Simplifying Algorithm Learning Using Serious Games

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Sahar Siraj Shabanah
Master of Science
George Washington University, 2001
Bachelor of Science
King Abdulaziz University, 1990

Director: Dr. Jim X. Chen, Professor
Department of Computer Science

Spring Semester 2010
George Mason University
Fairfax, VA

Dedication

In loving memory of my father Siraj and my grandparents Hussein and Sara.

To my beloved mother Zubidah Banawi.

To my soul mate and my husband Omer Zamzami.

To my lovely children Doaa, Hassan, Hussein, Ahmed, and Joud.

To my kind sister Samar and my brothers Mazen, Mohammad, and Manar.

Acknowledgments

It is difficult to write in a few paragraphs the thanks that I owe to all of the people who have helped me make it through my graduate study. Expressing gratitude carries the implicit risk of forgetting someone. To those not listed here, I humbly ask you to accept my apologies and to understand that such an injustice was not caused intentionally, but due to tricks played by memory.

Thanks to my family, especially my husband Omer Zamzami, my mother Zubidah, my children: Doaa, Hassan, Hussein, Ahmed, and Joud. No words would be enough. You know, more than myself, that all of this belongs to you. Thank you for believing in me, helping me get through the low points, rejoicing with me in the high points, and always being there. To my sister Samar and her children Yara and AbdulRahman, my brothers, Mazen and his wife Hatoon, Mohammad, and Manar, sister-in-law Izdihar and her daughters, my brother-in-law Mohammad and his wife Noha, thank you for all the incentive and support, be it from another room, another estate, or another continent.

So many people have helped and supported me, and made GMU such a special place. For a start, thanks to my advisor, Dr. Jim X. Chen. His encouragement, enthusiasm, and guidance kept me "going for it" despite some seemingly insurmountable hurdles. Thanks also my dissertation committee, Dr. Daniel Carr, Dr. Edward Wegman, and Dr. Harry Wechsler, for all of their comments, suggestions, and help. An extra thanks to Dr. Henry Hamburger, for accepting me into the PhD program, and for Dr. Stephen Nash and Dr. Daniel Menascé for helping me to re-enroll in the PhD program after a leave of absence. Thanks to Nooshe and Therese for their support, advice, and for knowing the answers to just about any question presented to them. Special thanks to Prof. Dennis Young for helping me edit this dissertation.

All my thanks to my friends, Malak Alnory, Hanan Zyan, and to the people of the Culture Mission of Saudi Arabia who helped and supported me during my study in USA.

Finally, my greatest gratitude to King Abdulaziz University in Jeddah, Saudi Arabia, for supporting me financially and emotionally throughout my research years, for having the confidence in me to succeed, and for giving me a chance. Special thanks to my friends and colleagues there, who have helped and supported me during my study.

Table of Contents

List of Tables

List of Figures

Abstract

SIMPLIFYING ALGORITHM LEARNING USING SERIOUS GAMES

Sahar Siraj Shabanah, PhD

George Mason University, 2010

Dissertation Director: Dr. Jim X. Chen

Data structures and algorithms are important foundation topics in computer science education. However, they are often complex and hard to understand. As a result, educational tools, such as algorithm visualization systems, are always needed to help students better learn and understand algorithms. The focus on graphics and sound instead of pedagogical aspects in the design of current algorithm visualization systems undermines effectiveness in teaching algorithms. In addition, most algorithm visualization systems lack features that encourage student engagement. This research addresses some required issues in creating algorithm visualization techniques by integrating learning theories and models in algorithm learning and by visualizing algorithms using computer games to fully engage students in the algorithm learning process. A new algorithm visualization and learning approach, namely Algorithm Visualization using Serious Games (AVuSG), has been introduced. It visualizes algorithms using educational or serious games to benefit from their popularity and engagement to motivate students who are learning algorithms. Moreover, it facilitates the students' assessment using the winning-losing criteria of computer games without the need for external questions. The conceptual framework of AVuSG visualizes the algorithm to be learned using three forms of representations: Text, Flowchart, and Computer Game. Moreover, it defines three types of learning processes: Viewing,Playing, and Designing, which learners can use to engage with any of the three forms of the produced algorithm visualizations.

Finally, AVuSG integrates learning theories with game design to introduce three learning models: Bloom Based, Gagne Based, and Constructivist Models, which can be adopted either by students to learn the algorithm or by the instructors to teach the algorithm depending on the learning objectives that they want to achieve.

To demonstrate AVuSG framework, a software system called Serious Algorithm Game Visualizer (Serious-AV) has been developed to provide a viewer and a designer for each algorithm representation form (Text, Flowchart, and Computer Game). Serious-AV is used on two levels: by the user interacting with the visualizations and by the developer creating these visualizations. The user views the algorithm text and flowchart using the Algorithm Text Viewer and the Algorithm Flowchart Viewer, respectively, and plays its game using the Algorithm Game Viewer. On the other hand, the developer uses the three development tools: Algorithm Text Designer, Algorithm Flowchart Designer, and Algorithm Game Designer to create each of those three algorithm representation forms. The Algorithm Game Designer is an integrated development environment tailored to create computer science educational games, namely an Algorithm Game, for the Windows platform to teach about specific algorithms and data structures. To visualize an algorithm, an Algorithm Game must have a game-play that simulates the behavior of the visualized algorithm and graphics to depict the features of its data structure. Several components and editors have been added to the Algorithm Game Designer to automate and simplify the visual development of algorithm games using as little code as possible. First, it is built on top of a game engine called SAVGEngine, which contains several modules that provide the basic functionality to the newly created game in addition to a repository of ready-to-use algorithm game components that can be altered and plugged in to the new game. Moreover, the Algorithm Game Designer includes an Algorithm Game Template to be used as a blueprint in the creation of a new algorithm game by providing basic game classes that implement the algorithm game basic architecture. Furthermore, the Algorithm Game Designer contains five visual editors: Properties, Assets, Screens, Classes, and Graphic Items Editors for creating all game content visually with no code, using a flexible, user-friendly graphical user interface (GUI). Lastly, it takes advantage of current software tools and libraries, such as XNA-GS, and VS Shell-Isolated Mode to simplify game design. At last, several algorithm visualizations, including texts, flowcharts, and algorithm games prototypes, have been developed using the developed systems.

Chapter 1: Introduction

An algorithm is defined as "a sequence of computational steps that takes a value, or set of values, as input and produces a value, or set of values, as output" [2]. Algorithms are the underlying theory for Computer Science and at the heart of every nontrivial computer application. Every computer scientist, every professional programmer, and every computer science student should know about basic algorithms that solve frequently occurring problems and generic data structures that make data organization and retrieval more efficient. Students deal with algorithms in almost every computer science course; for instance, in computer graphics students learn to render algorithms to draw objects, in networking they learn algorithms to solve network traffic congestion, and in database they learned algorithms to search or sort data. Accordingly, teaching algorithms is one of the main activities that takes place almost in every computer science class.

However, understanding algorithms is one of the most difficult aspects of the study of Computer Science because algorithms usually model complicated concepts, refer to abstract mathematical notions, or describe complex dynamic changes in data structures to solve relatively difficult problems. Consequently, teaching algorithms is a more demanding task facing computer science instructors, requiring much explaining and illustrating. Teaching aids other than chalkboard and view-graph are always needed to help students learn and understand algorithms better [3]. Researchers have been trying to find the best way to learn and teach algorithms. The human ability to realize graphic representations faster than textual representations led to the idea of using graphical artifacts to describe the behavior of algorithms to learners. One of the best known approaches, known as Algorithm Visualization, uses graphics, sounds, and animations to communicate how algorithms work [4].

1.1 Problem Statement

Since the production of the movie Sorting Out Sorting [5], a great number of algorithm visualization systems have been built with the promise of improving algorithm learning. Examples of such systems are BALSA [6], ANIMAL [7], ALVIS [8], and Algorithm Explorer [9].

However, regardless of the large number of existing algorithm visualization systems, their promise as helpful educational tools is mostly unsatisfied. Many researchers who conducted experiments to determine the efficiency of existing algorithm visualization systems in teaching algorithms have reported unpromising results. For example, Badre et al. [10], in their evaluation of algorithm visualization systems as instructional aids, found that there is no evidence that algorithm visualization systems really enhance the understanding of algorithms, despite the learners' interest to use these systems. Moreover, a study of algorithm visualization systems efficiency concluded that algorithm animations have unreliable effects on student understanding of algorithms [11]. In addition, Byrne et al. [12] state that "the benefits of animations are not obvious." In a meta-study of over twenty empirical experiments conducted to assess existing algorithm visualization systems, Hundhausen et al. [13] describe algorithm visualization systems as having "failed to catch on in mainstream computer science education," and conclude that "studies in which students merely viewed visualizations did not demonstrate significant learning advantages over students who used conventional learning materials."

The failure of current algorithm visualization systems as effective educational tools results for two main reasons. First, the developers of most algorithm visualization systems usually focus on graphics and sound as opposed to pedagogical issues in their designs [14]. Second, a small number of algorithm visualization systems allow learners to interact effectively with their displayed visualizations despite the importance of such engagement. In fact, algorithm visualization systems become very successful when engaging students in the process of learning [13], since "what the students do, not what they see, has the greater impact on learning" [15]. Moreover, the more learners interact with algorithm visualization systems, the better they understand the algorithms these systems visualize [16].

1.2 The Proposed Approach

Computer Games are software systems that involve interaction with a user interface to generate visual feedback on a computer or a video device and utilize many elements, such as fun, play, winning/losing, and competition. Computer games that involve learning of certain knowledge are called Educational Computer Games [17]. Despite the frustration with educational computer games in the past, recently, they reemerged as Serious Games, which have serious purposes, such as training, advertising, simulation, and education [18]. Chamberlin [19] describes computer games as having many roles, such as "tutoring, amusement, exploring new skills, promoting self-esteem, drill and practice, or creating a change in attitudes."

This research introduces Algorithm Visualization using Serious Games (AVuSG), a novel approach for teaching and visualizing algorithms that addresses the shortness in current algorithm visualization systems in two ways: first, researchers in computer science education identify many pedagogical requirements for the success of an algorithm visualization system [20]; the most effective factor is the students' engagement, since there is an increasing correlation between the level of student learning and engagement in algorithm visualization systems [21]. Alternatively, computer games fully interact with players and encourage them to think and act. Furthermore, according to Rieber [22], "playing computer games involves active engagement" because computer games support all components of flow [23], as defined by Csikszentmihalyi [24]. In fact, recent research shows that computer games build content and activities into high levels of motivation and interest that are useful in reinforcing a wide variety of learning values [25]. Given that active engagement with algorithm visualization systems is far more important to learners than the graphics that they see, and given that computer games fully interact with players, AVuSG visualizes algorithms using computer games. AVuSG benefits from computer games active engagement, intrinsic motivation, popularity, and entertainment to maximally motivate and engage students in algorithms learning and enhance their comprehension. Second, since researchers emphasize the importance of learning theories and teaching aspects in the design of effective algorithm visualization systems, AVuSG uses learning theories and models to derive learning models for algorithms teaching [14].

1.3 Motivation

Is there a need for another Algorithm Visualization system? Over a twenty year period, many algorithm visualization systems have been produced. However, in their effort to build a wiki for current algorithm visualization systems, Shaffer et al. [26] searched and analyzed hundreds of visualization systems and found that "most existing algorithm visualizations are of low quality and the content coverage is skewed heavily toward easier topics." Still, positive about visualizations in general, Shaffer et al. [26] conclude, "while many good algorithm visualizations are available, the need for more and higher quality visualizations continues. There are many topics for which no satisfactory visualizations are available. Yet, there seems to be less activity in terms of creating new visualizations now than at any time within the past ten years."

Why using computer games to visualize algorithms? Besides active engagement, computer games have many features that motivate their use for education in general and for algorithms learning in particular, as explained in the following.

- **Computer games are popular.** Computer games have been broadly played all over the world by adolescents and young adults. In particular, most college students spend half the time reading that they spend playing computer games [27]. Therefore, the opportune and effective method for teaching today's students is to use computer games.

- **Computer games are based on intrinsic motivation.** Players spend considerable time in learning and playing computer games solely for the sake of playing. Therefore, playing computer games is an intrinsically motivating activity, in which people engage for no reward other than the interest and enjoyment that accompanies it. Malone [28] stresses the importance of Intrinsic Motivation in improving learning. Moreover, Dempsey et al. [29] state that "games result in significantly higher levels of motivation, reduce training time, and may improve retention of what is learned." As a result, the use of computer games can motivate students to learn algorithms and enhance their comprehension.

- **Computer games simplify evaluation.** Sivasailam and Thiagarajan [30] state that games

can be used as performance tests "for individual evaluation of transfer and application" of acquired knowledge; this use of games is "obviously superior to any paper-and-pencil test and is easier to administer." In short, student understanding of algorithms can be evaluated using computer games that simulate and imitate the behavior of those algorithms.

- **Computer games utilize entertainment.** Garvey [31] defines playing games as "pleasurable, spontaneous, and voluntary." Moreover, Gee [32] describes the knowledge earned by playing games as "the cycle of expertise" that gives people pleasure. Importantly, Chamberlin [19] adds, "attainment of educational outcomes can be combined with voluntary participation in play." Therefore, the use of computer games can convert an unpleasant and tedious operation of algorithm learning into an enjoyable and interesting experience.

- **Computer games add emotional element.** Events that are accompanied by intense emotions result in long-lasting learning, such as training games, simulations, and role plays that add an emotional element to learning [33].

1.4 Research Questions

How the pedagogical requirements can be satisfied when teaching algorithms using computer games? What are the specifications of the educational computer games that efficiently visualize and teach algorithms to new generations? How computer games can be designed and developed to efficiently visualize algorithms? What methodologies and technologies can be used to produce prototypes for these games? This research attempts to answer these questions by focusing on exploring new methods and new approaches for designing and implementing educational computer games for algorithm learning and visualization.

1.5 Outline of the Research

The main purpose of this research is to explore the educational impact of an alternative form of active engagement produced by computer games technology, and by the integration of learning

theories to teach algorithms. Such a goal is made concrete by particular tasks, including:

1. Comparing several algorithm visualization systems according to their level of engagement and presenting an alternative form of active engagement produced by computer games technology; identifying problems and unexplored opportunities of current algorithm visualization systems technologies and tools, and justifying why a new algorithm visualization system is needed.

2. Summarizing the history of computer games usage in education; surveying various game development technologies, such as application program interfaces, visual creation tools, game engines, and domain-specific languages.

3. Specifying the conceptual framework of the AVuSG approach by defining:

   (a) The forms of algorithm visualizations the approach has produced.

   (b) The learning processes the learners can use to engage with each one of these algorithm visualizations forms.

   (c) The educational models the approach has deployed to teach algorithms using games.

4. Developing an algorithm visualization system that implements the AVuSG approach by providing the following:

   (a) Viewers that display all the forms of algorithm visualizations produced by the approach to the learner.

   (b) Designers that simplify the creation of all the forms of algorithm visualizations produced by the approach for the developer.

5. Implementing a new game development environment that has several components to simplify game creation, including the following:

   (a) Game Template, which be used as a blueprint for creating new games.

(b) Game Engine, which encapsulates all the basic, common functionality for creating computer games.

(c) Game Components Repository, which has built-in, ready-to-use game components that can be used to develop various games, such as game assets, screens, graphic items, data structures, and algorithms playing methods.

(d) Code Editors Basic and Script Editors that help in coding the game.

(e) Developer Graphics Editors, which support the visual creation of different game components, such as game properties, assets, graphic items, classes, and screens.

6. Implementation of real-world examples that illustrate how algorithm games can be effectively designed and developed, using the game development tool that have been built.

1.6 Game Development Challenges

Computer game development is a hard and complex operation that requires experience in computer graphics and animation. Game developers face unique challenges and scenarios, which turn game development into a very atypical domain when compared to software development in general. Specifically, when creating a computer game, the developer must write code to handle graphics devices, sound cards, input devices, memory, and processor. Therefore, game development tools have been created to help game designers in designing efficient games. In the beginning, many programming libraries that allow game programmers to directly access the machine hardware using abstract code have been released; they are known as Multimedia Application Program Interfaces (APIs). However, the abstraction level of these APIs is general for computer games development, and developers may interact with them only by writing complex programs. Consequently, visual game creation tools, which help users to create complete games with no programming at all, appeared as an alternative. They provide graphical and easy-to-use interfaces for creating sprites, entities, sound, screens, and other resources for the game. However, their lack of support for programming languages prevents them from being adequate for real-world games. Although some of these tools provide visual programming constructs and others include script languages, the former can be used

only in simple cases and the latter require users to learn a new language, which is error-prone inside an unstable codification environment. Finally, game engines have been produced, which are reusable APIs that gather common game development foundations. They provide developers with a programmatic interface in which game behavior can be specified, but they lack the support for visual development. Some game engines provide visual tools, but these tools have been focused on creating a specific portion of the game rather than a complete game and their output must be consumed programmatically.

In this research, I have developed a new integrated development environment, namely Algorithm Game Designer, which has been tailored to create computer educational games for the Windows platform that takes advantage of the previously described game development tools in its design, while avoiding some of their pitfalls. First, to benefit from multimedia API, the designer has been built on top of XNA Game Studio. However, it is a complete, stand-alone IDE, which supports the C# programming language by providing debugging and compiling operations to the designers. Moreover, it includes a game engine that supports the development of the games by providing modules to handle input, sound, rendering, and other game functions. Nevertheless, to provide visual development, it has five visual editors to support the creation and loading of the game entities and sprites visually, without any coding. It also includes a game template that provides the game content and basic game screens in addition to the Base Game and Play Game classes that compose the game skeleton. Finally, it includes a repository of built-in game content items that can be loaded directly into the game using the graphical editors.

1.7 Dissertation Contributions

The main contribution of this research is a detailed study, illustrated with real examples, of how educational computer games can be used to improve algorithm learning. Many contributions are encompassed by such a study:

1. **The introduction of a novel approach for visualizing algorithms.** This research introduces

Algorithm Visualization using Serious Games (AVuSG), a new algorithm visualization approach that uses serious or computer games to overcome the shortness in current algorithm visualization systems. It benefits from the players' desire to win, love to compete, and to be entertained resulting from playing games to motivate and help students better learn algorithms. It facilitates the students' assessment by using the winning-losing paradigms of computer games without the need for external questions [34].

2. **The presentation of an alternative form of active engagement.** Active Engagement Taxonomy defines five active engagement forms for the user with any algorithm visualization system: this includes viewing, responding, changing, constructing, and presenting [35]. This research presents "playing" games as a new form of active engagement that maximally engages students through repetition, challenge, and enjoyment in addition to including all active engagement forms of Active Engagement Taxonomy [36].

3. **The investigation of a new research area in educational games literature.** A comprehensive survey of the usage of computer games in education has been carried out in this research, as shown in Section 2.3, leading to the conclusion that no computer game has been designed either to teach or to visualize algorithms. Therefore, by investigating the means and methods for the usage of computer games in algorithm learning and visualization, this research has initiated a new research area into educational games literature and in algorithm learning and visualization techniques.

4. **The integration of learning theories and models in algorithm visualization.** The AVuSG introduces three learning models that can be adopted to teach algorithms by using computer games, each one of which integrates a learning model or theory depending on the learning goals to be achieved. These three models are Bloom Based Model, Gagne Based Model, and Constructivist Model.

5. **The definition of the specification of the computer game that visualizes an algorithm.** By describing its attributes, type, structure, design elements, and architecture, an Algorithm Game may be specifically defined.

6. **The development of several viewing and development tools that support the AVuSG approach.** The developed tools were integrated in one system, namely Serious Algorithm Games Visualizer (Serious-AV), which contains two main sub-systems:

   (a) Serious-AV Viewer that supports the viewing of the Algorithm Text, Flowchart, and the playing of Algorithm Game by providing three viewers: Algorithm Text Viewer, Algorithm Flowchart Viewer, and Algorithm Game Viewer.

   (b) Serious-AV Designer that supports the creation of the Algorithm Text, Flowchart, and Game by providing three designers: Algorithm Text Designer, Algorithm Flowchart Designer, and Algorithm Game Designer.

7. **The development of an innovative environment for visual game development.** A complete portrayal of how to create and provide an integrated development environment (IDE) for visual game development has been given by introducing Algorithm Game Designer. It has been tailored to create computer games by deploying several game development technologies in one environment as follows:

   (a) Supporting visual creation for all types of game content without any coding, using five graphical editors. The game assets, such as textures, fonts, models, sound, and effects, are loaded into the game directly through the Assets Editor. The game properties, such as the level number and number of lives, are set visually using the Properties Editor. The game graphic items, such as ball, card, and paddle, are designed and added to the game using the Graphic Items Editor. The game screens, such as menu and playing, are designed using the Screen Editor. In addition, the game classes are defined using the Classes Editor.

   (b) Containing a Game Content Repository that provides ready-to-use game content, which can be altered and plugged in to the new game, such as data structures, algorithms, graphic items, game screens, and game assets.

   (c) Including a game engine called Serious Algorithms Visualization Game Engine (SAV-GEngine), which provides a reusable methodology for creating games. Overall, the

engine is responsible for various major modules that encapsulate all functionality for creating games, such as graphics, sound, input, game screens, physics, and storage managers.

(d) Providing advanced game developers with XNA & C# coding, debugging, and compiling operations support.

(e) Automating the game development process by implementing each game as a Finite State Machine (FSM) that transitions between many game screens. Using the Screen Editor, the developer can create all game screens and define the transition order between them visually without coding.

(f) Containing an Algorithm Game Template, which provides each newly created game with its architecture by defining basic components, such as the base game class and default game screens.

8. **The introduction of a learning environment for computer games design.** The Algorithm Game Designer can be used to teach students how to design computer games in general. Its game engine has been designed as an educational game engine by providing documentation and source code, and using the same language (C#) in both its implementation and accessing its functionality. Moreover, the visual editors help students to concentrate on game design aspects instead of content development.

9. **The implementation of several prototypes for algorithm games.** Validation of the game development process in the presented environment is achieved through the implementation of real-world scenarios and prototypes for algorithm games. Moreover, these algorithm games prototypes illustrate how algorithm games can be effectively used to visualize algorithms.

10. **The employment of new technologies.** This research has deployed new technologies in the implementation of the developed systems, such as Microsoft Visual C# 3.0, .NET Framework 3.5, XNA Game Studio 3.1, and Visual Studio Shell-Isolated Mode 2008.

In summary, the work presented by this research is a valuable contribution to achieve a smooth integration between computer games, learning theories, and algorithm visualization to improve algorithm learning.

1.8 Dissertation Organization

The remainder of this dissertation is organized in nine chapters as follows:

Chapter-2 reviews various forms of software visualization, analyzes current algorithm visualization systems, presents a comprehensive history of computer games usage in education, and surveys some of important educational games and various game development technologies. A discussion about how computer games are suitable for algorithm visualization is also provided.

Chapter-3 explains the theoretical background of the research by exploring different learning theories and models, motivation theories, and the theory of algorithms and data structures.

Chapter-4 details the proposed approach; an algorithm visualization approach named Algorithm Visualization using Serious Games (AVuSG) is specified. Its conceptual framework is defined and its software systems design requirements are described.

Chapter-5 overviews computer games genres, game design methodologies, and then defines Algorithm Games and their properties.

Chapter-6 shows some prototypes for computer games that were designed to teach algorithms.

Chapter-7 presents Serious Algorithm Game Visualizer (Serious-AV), which has been developed to implement the proposed approach.

Chapter-8 details the Algorithm Game Designer, which is the algorithm game development environment, as well as related components, such as its visual editors, code editors, and a game engine.

Chapter-9 presents complete case studies for algorithm games that were developed using the Algorithm Game Designer.

Chapter-10 gives an evaluation and investigates the future work that can be carried out from this research.

Chapter-11 summarizes this work and explains how the research questions have been answered.

Chapter 2: Related Research

This research proposes an algorithm visualization technique that uses educational computer games to simplify algorithm learning. Three related research areas are considered in this chapter: algorithm visualization, educational computer games, and computer games development techniques; systems form each area have been reviewed and analyzed. The first section explains Software Visualization and its classifications as background information for algorithm visualization systems. The second section reviews some significant algorithm visualization systems. The third section outlines a historical background for educational computer games and evaluation of several important educational computer games. Lastly, the fourth section compares the different game development technologies and systems. The last subsection of each of the last three sections shows how this research is different from currently reviewed systems or approaches.

2.1 Software Visualization (SV)

Visualization is defined in the Oxford English Dictionary as "the process of forming a mental picture or vision of something not actually present to the sight" [37]. In computer science, Software Visualization refers to the process of developing software systems that "use the crafts of typography, graphic design, animation, and cinematography with modern human-computer interaction and computer graphics technology to facilitate both the human understanding and effective use of computer software." Software visualization can be classified into two main categories: Program Visualization and Algorithm Visualization (Figure 2.1). Visual Programming (VP) is the process of using visual techniques to specify a program, while Programming by Demonstration (PbD) determines the specification of programs using user-demonstrated examples; both are considered as software visualization forms [1].

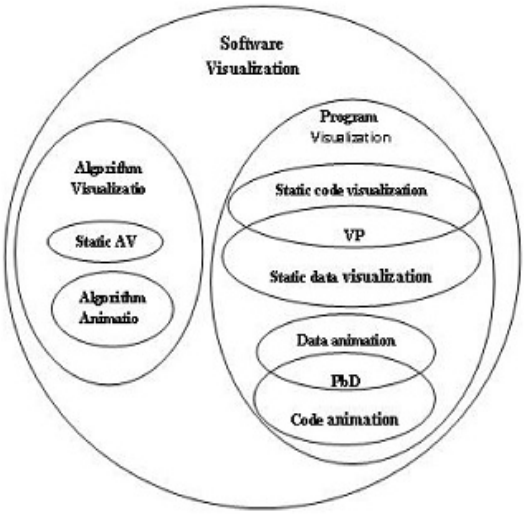

Figure 2.1: Software Visualizations Categories [1]

2.1.1 Program Visualization (PV)

Program visualization is defined as "the use of graphical artifacts to represent both the static and dynamic aspects of programs" [4].

Static Program Visualization

Static program visualization is the use of fixed diagrams and graphics to visualize either program code or data.

- Code Visualization: producing some kind of pretty-printing or program map for the code. For example, SEE Visual Compiler (1990) is a program visualizer that takes a C source code as an input and produces as its output high quality typeset presentations on a laser printer (Chapter 4 of [1]) [38].

- Data Visualization: constructing diagrams of data structure showing its content using boxes and arrows. For example, BlueJ (2003) is a data visualization tool (Figure 2.2) that visualizes abstraction entities of object orientation and allows direct interaction between them to illustrate the object oriented concepts [39].

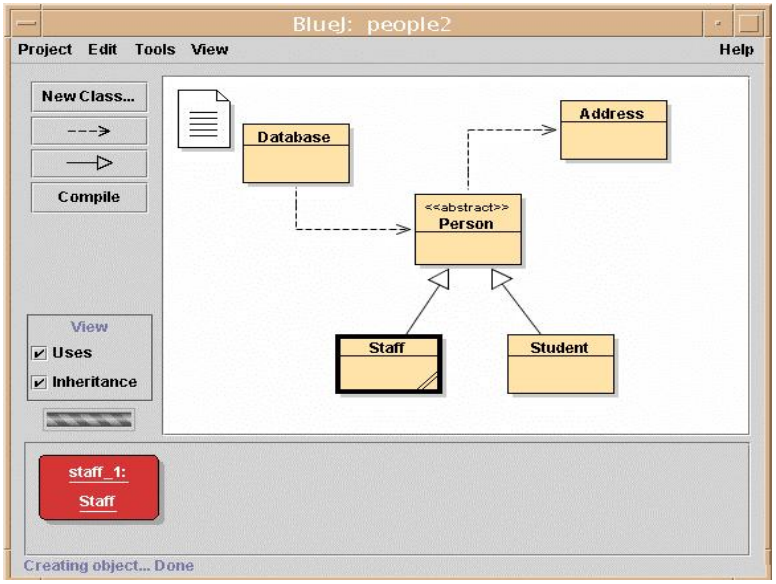

Figure 2.2: BlueJ–Class Structure

Dynamic Program Visualization (Program Animation)

Program animation is the display of a dynamic graphical representation of a program execution to enhance understanding of the actual implementation of a program code or data [40].

- Code Visualization: highlighting lines of code as they being executed. Jeliot (1997) is an example of a code visualization tool that illustrates the execution of input-output, assignment, selection, and loop statements. It is a Java programming environment that automatically animates Java source code, displays every instruction, and creates stack frames for method calls [41]. Jeliot2000 is a newer version that separates the source code from the animation, and uniformly animates all variables, text input-output, expression evaluation, and control decisions [42].

- Data Visualization: showing diagrams of arrows and contents that are changing dynamically as the program runs. JGrasp (2004) is an interactive development environment (IDE) for Java that allows the creation of dynamic data structure visualizations through data structure "viewers" that automatically updated to reflect changes in the data structure as the program runs. Dynamic visualizations can be produced for many data structures, such as linked lists, heaps, and red-black trees. For example, the JGrasp dynamic visualization of a red-black tree

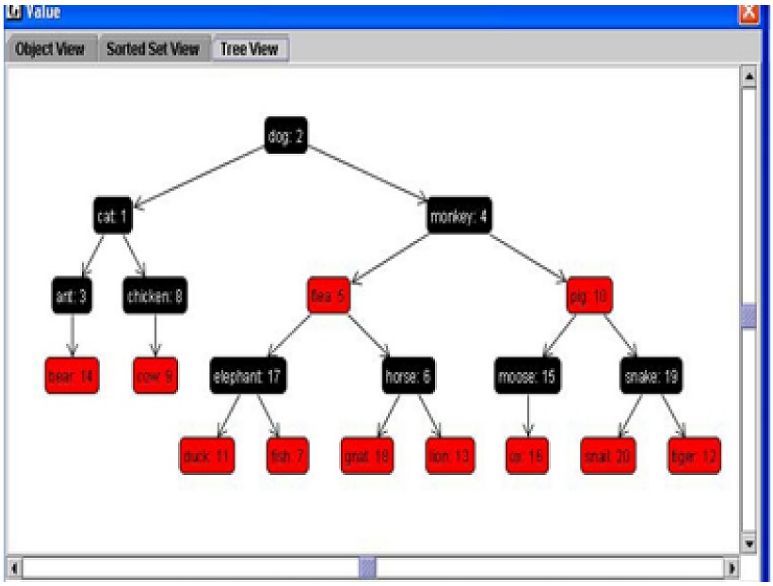

Figure 2.3: JGrasp–Red-Black Tree Visualization

(Figure 2.3) shows how the tree grows, shrinks, and changes colors as a program is stepped through and explained [43].

2.1.2 Algorithm Visualization (AV)

Algorithm visualization is the process of "visualizing all the states of the data structures during the execution of an algorithm, showing the object structures (components, sub components, and their values) that are required to understand the logic and the behavior of the algorithm, omitting data types that have no affect on the algorithms behavior, including enough frames to explain the operations the algorithm performs" [44]. Algorithm visualization can be divided into static and dynamic algorithm visualization.

Static Algorithm Visualization

Static Algorithm Visualization is the process of using a series of fixed or still diagrams to describe the behavior of algorithms. For example, Flowcharts, which consist of boxes and arrows that describe the behavior of algorithms, are considered as static algorithm visualizations.

Dynamic Algorithm Visualization (Algorithm Animation)

Dynamic Algorithm Visualization is the process of using animations to communicate how an algorithm works. Mark Brown defines an algorithm animation environment "as a mean for exploring the dynamic behavior of algorithms that makes possible a fundamental improvement in the way we understand and think about them. It presents multiple graphical views of an algorithm in action, exposing properties that might otherwise be difficult to understand or even remain unnoticed" [45].

2.2 Review of Algorithm Visualization Systems

Over the last two decades hundreds of algorithm visualization systems have been implemented, and provided freely to educators, each with its own philosophy and range of application. First, this section analyzes some of algorithm visualization systems according to the type of engagement they provide for their users. Next, it reviews some web-based, hypermedia, and specific domain algorithm visualization systems in addition to algorithm visualization support systems.

2.2.1 Algorithm Visualization Systems–Active Engagement Taxonomy

Naps et al. [35] define six different forms of learner engagement with an algorithm visualization technology. The first form of engagement is "no viewing," meaning no visualization technology is used at all. The remaining five forms construct the Active Engagement Taxonomy: viewing the visualization passively, responding by answering questions related to the presented visualization, changing the visualization data or some other features, constructing visualizations of algorithms under study, and presenting the visualization to an audience for feedback and discussion. Next, some examples of current algorithm visualization systems have been reviewed by classifying them according to the type of engagement that they support.

Sorting Out Sorting (SOS) is the first well-known algorithm visualization system, which has been produced by Ronald Baeker in 1981. It is a 30-minutes color, sound film that uses animations to explain three sorting algorithms: insertion, exchange, and selection sorts. The film uses bars and single-digit numbers to represents data items and colors to distinguish between sorted and unsorted

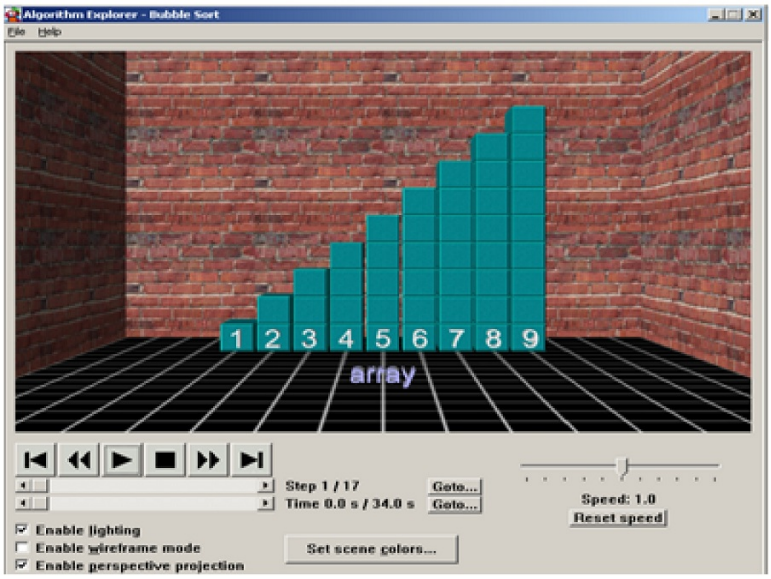

Figure 2.4: Algorithm Explorer–Bubble Sort Visualization

items [5]. However, Brown Algorithm Simulator and Animator (BALSA), which was created in 1984 by Mark Brown, is the first real-time, interactive algorithm visualization system that allows learners to interact with the visualized algorithm. It provides the user with some kind of control over the visualization using two types of primitives: 1) Display primitives that allow the user to create, size, and position the algorithm; 2) Interpretive primitives that allow the user to start, stop, slow down, or even run an algorithm backward [6]. In 1988, BALSAII extended BALSA with step and break points and a number of other features [46]. Later, Zeus was created by Brown in 1991 to demonstrate the utilization of color graphics and computer-generated sounds [47] [48]; then, followed by Zeus3D in 1993 to support 3D graphics [49]. A recent 2007 algorithm visualization system Algorithm Explorer (Figure 2.4), uses three-dimensional objects to represent common data structures and animations to visualize algorithms. It consists of three main components: 1) User Interface that enables the viewer to control how the visualization played and displayed on the screen; 2) Development Interface that help the developer to construct and configure new visualizations; 3) API that represents data blocks and arrays as 3D boxes (Block World), graphs as spherical nodes (Graph World), and recursion and general function tracing (Stack World) [9]. SOS allows learners to view the visualization passively while BALSA and Algorithm Explorer provide the user with some

kind of control over the visualization viewing process. Viewing is the most common form of engagement that existing algorithm visualization systems provide for their users, other less supported engagement forms are responding, changing, constructing, and presenting [35].

In 1991, Arin Korhonen built a visualization system that supports responding by asking viewers to answer questions related to the presented visualization. It consists of two parts: 1) Trakla Server that automatically generates algorithm exercises and assesses students returned answers; 2) Graphical Editor that presents the exercise text graphically, offers interactive tools to solve it, and generates the answer using the required textual format [50]. WWW-Trakla (2000) is the successor of Trakla that allows students to solve the problems using a graphical editor, (Tred), on a web page [51]. In 2001, Matrix (Figure 2.5) was designed to overcome the limitations in Trakla graphical part. It is a general purpose framework for building concept visualizations using visual algorithm simulation, which allows the user to manipulate data structures directly, as an algorithm would do, by using the graphical user interface facilities without writing any code. Moreover, using Matrix, users can visualize their own code (Java classes) implemented using a library of implementations for all supported fundamental data types (FDT) [52]. Trakla2 (2003) is a newer version of Trakla that supports the creation and publishing of interactive exercises for data structures and algorithms. It is also based on Matrix to build visual algorithm simulation exercises and give immediate feedback on learners' performance by evaluating the accuracy of these exercises [44]. MatrixPro (2004) was built to provide instructors with an easy usage of the functionalities of Matrix with lecture support and easy demonstration [53]. Created in 2005, Java Hosted Algorithm Visualization Environment (JHAVE) is not an algorithm visualization system. Rather, it is a support environment for a variety of algorithm visualization systems, giving these systems a drawing context. It provides many features to the visualizations, such as a standard set of VCR-like controls that let students step through the algorithm's visual display. In addition, it has information and pseudo-code windows where visualization designers can author statically or dynamically generated content to help explain the significance of what the student sees in the algorithm's graphical rendering. Moreover, it provides stop-and-think questions, which can be designed in a variety of formats to pop up at the algorithm's key stages, to support the responding category of engagement [16].

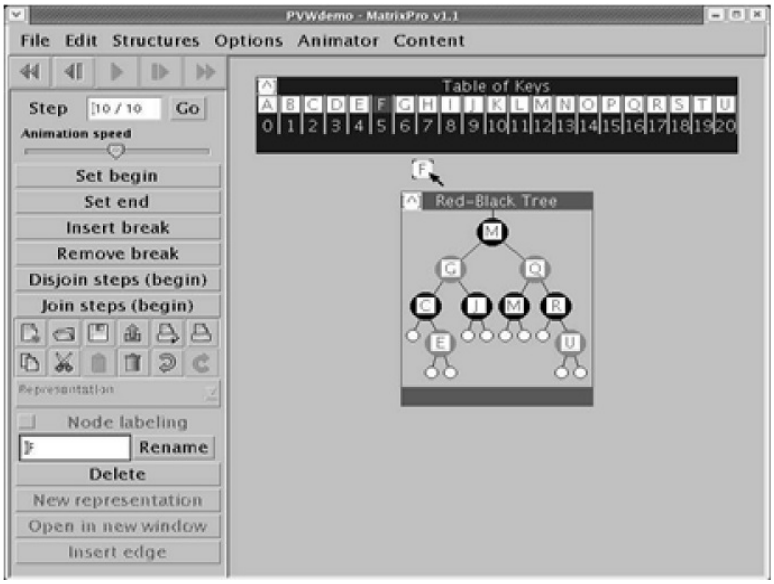

Figure 2.5: MatrixPro

Some algorithm visualization systems allow their viewers to change the visualization data or some other features. For example, GeoWin (2002) is a visualization system that allows the user to manipulate a set of actual geometric objects through the interactive interface [54]. Another system, CATAI (2002), enables the user to invoke some methods on the running algorithm. In terms of method invocations, it is possible to directly access the content of the data structures or to execute a piece of code encapsulated in an ordinary method call [55].

Tango (1989) and its successor XTango (1992) were developed by John Stasko to support the constructing of algorithm animations using API calls into a C implementation of the algorithm. However, such animations require explicit placement and low-level setup of every component of the animation [56] [57]. In 1993, Stasko developed two more systems: Polka to visualize parallel algorithms [58] and Polka3D to explore the use of 3D graphics [59]. In 1997, Samba, then JSamba, was introduced as interactive animation interpreters for Polka. They support the developing of animations via text scripts by including a number of parametrized ASCII commands that performed different animation actions. Thus, programs written in any programming language could be animated simply by having them output Samba commands at interesting event points [60]. ANIMAL was developed by Robling's in 2002 to support building animations using a graphical editor. It

generates a text script that can then be modified by hand. It can be used to visualize many representations through the composition of low-level drawing and animation operations not on the abstract data types represented in the animation. ANIMAL animations can be generated without actually implementing the algorithm in code. It has specialized support for displaying code or pseudo-code in the animation. It also has a sizable online repository of animations that can be retrieved through the interface itself [7]. Algorithm Visualization Storyboarder (ALVIS) is an interactive environment that was built by Hundhausen in 2004 to enable students to quickly construct rough, unpolished (low fidelity) visualizations in much the same way they would do so with simple art supplies, and interactively present those visualizations to an audience. In ALVIS, users create storyboards by using a graphics editor to cut out and sketch visualization objects, which they lay out in the "Storyboard View" by direct manipulation. They then specify, either by direct manipulation or by directly typing in Spatial Algorithmic Language for StoryboArding (SALSA) commands how the objects are to be animated over time [61] [8]. ALVIS-Live! (2005) is a newer version of ALVIS that implemented "What You See Is What You Code" model to facilitate learner-constructed algorithm visualizations. In this model, the line of algorithm code currently being edited is reevaluated on every edit, leading to the dynamic update of an accompanying visualization of the algorithm [62].

A project called Algorithm Studio, which has been designed by Hundhausen in 2004, supports presenting by allowing students to present their own visual solutions, constructed using ALVIS, of algorithm design problems to their instructors in one-to-one sessions; then, to the entire studio. Finally, at the end of the semester, the students must present their solutions for an extra set of design problems to a jury of instructors [63].

2.2.2 Algorithm Visualization Systems–Other Classifications

Web Based Algorithm Visualization Systems

The advantages of the World Wide Web triggered the development of several web-based algorithm visualization systems, such as Mocha (1996), which is an algorithm visualization system that uses a client-server approach where users point a web browser at a web page containing a Java applet. The applet starts an algorithm on a remote server; the algorithm transmits interesting events via TCP to

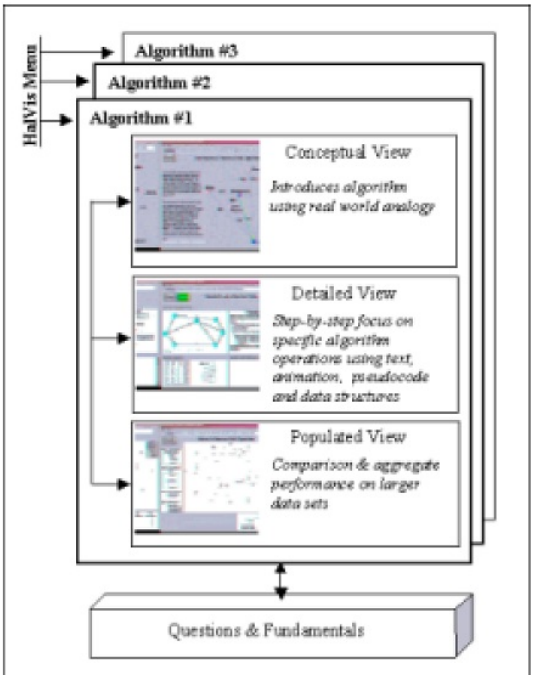

Figure 2.6: HalVis Architecture

the applet, and the applet generates the appropriate animations [64]. Another web-based algorithm visualization system is Collaborative Active Textbooks (CAT), also created in 1996, runs both the algorithm and the animations as applets on the user's web browser; these applets were written in a scripting language [65]. JCAT (2001) system replaces the scripting language of CAT with Java and offers 2D and 3D animation libraries [66].

Hypermedia Algorithm Visualization Systems

Developed in 2000, Hypermedia Algorithm Visualizations (HalViS) is an algorithm visualization system that uses the "Asymmetrix Toolbook" to produce algorithm visualization (Figure 2.6) that consists of five views: 1) Conceptual View: describes the algorithm in general terms; 2) Detailed View: includes a textual description and a depiction of the algorithm's behavior; 3) Populated View: provides an animated view of the algorithm; 4) Questions View: measures comprehension; 5) Fundamentals View: contains information about basic concepts common to virtually all algorithms [67]. Another algorithm visualization system that uses hypermedia is Structured Hypermedia ALgorithm EXplanation (SHALEX), which has been created in 2006. It explains an algorithm textually and

visually using a bottom-up and top-down structure. The algorithm structure is represented as an acyclic digraph with nodes representing abstractions and edges representing operation dependencies [68].

Specific Domain Algorithm Visualization Systems

Some algorithm visualization systems visualize specific domains algorithms, such as Gasp and GaspII (1997), which are both designed to visualize graphs algorithms [69] [70], JFlap and Pate (2000), which are both visualize FA (Finite Automata) algorithms [71], and AI-Search (2001), which visualizes AI (Artificial Inelegance) algorithms [72].

Support Systems for Algorithm Visualization

Conceptual Keyboard (2005) was built by Baloian et al. [73] to support the interaction between the learner and the data structure of the algorithm in an algorithm visualization system.

2.2.3 AVuSG Presents Alternative Form of Active Engagement

As described in Section 2.2.1, Naps et al. [35] define six different forms of learner engagement with a visualization technology. The first form of engagement is "no viewing" meaning no visualization technology is used at all, while the remaining five are the active forms of engagement: viewing, responding, changing, constructing, and presenting. Viewing is the basic form of active engagement that has been supported by all algorithm visualization systems. Students watch the displayed visualizations passively and only engage with them using some basic viewing controls, such as play and rewind. Exemplars of systems that only support viewing are SOS [5], BALSA [6], and Algorithm Explorer [9]. To support the other active forms of engagement, algorithm visualization systems have provided additional features to students. For example, to support responding, some systems ask students questions related to the presented visualization, such as JHAVE [16]. Other systems support changing by allowing students to change the visualization data or some other features, such as GeoWin [54]. The systems that support constructing encourage students to build and construct

their own visualizations for the algorithms under study, such as ANIMAL [7] and ALVIS [8]. Finally, systems that support presenting provide tools that help students to present visualizations to an audience for feedback and discussion, such as Algorithm Studio [63]. Similarly, computer games support viewing, responding, changing, constructing, and presenting, too. Specifically, most computer games come in two modes: demonstration (demo) and playing. In demo mode, players view how the game is played passively, while in the playing mode they respond to the game events. Also, several computer games provide options for their players to change some features of their components, such as their color, sound, and graphics. Other games allow their players to construct new components in the game environment and present, those to other players, such as the World of Craft game [74]. Moreover, according to Rieber [22] "play is usually voluntary, holds intrinsic motivation, involves active engagement, and contains a make-believe quality." Therefore, this research introduces "play" as a new form of active engagement that maximally engages students through repetition, challenge, and enjoyment in addition to combining all five forms of active engagement that have been presented by Active Engagement Taxonomy.

2.3 Computer Games in Education

The use of computer games in education has progressed through four main stages. Computer games were first used for education purposes by the military. After showing success in military, computer games have been used to teach business related topics. Afterward, educational computer games have been used to teach school related subjects and basic skills. Nowadays, educational computer games have emerged as serious games that have been used in various areas, such as teaching politics and health training.

2.3.1 Military Educational Computer Games

The first organized effort to use games for direct educational purpose in western culture was started at 1824 when the German military used Kriegsspiel game to simulate strategy on the battlefield [75]. In 1998, the US military started the use of computer games in training with a modified version of the commercial game Doom that was called Marine Doom. Currently, the number of computer

games used by the military for low level training, such as Bottom Gun for Torpedo Firing is very small while the most used games are for team tactics, decision-making, and conflict resolution, such as Delta Force2, Guard Force, and Joint Force Employment [76]. The military is considered a true supporter of educational computer games, because over the years it developed and tweaked a lot of computer games, such as America's Army, Full Spectrum Warrior, Close Combat Marines, and Full Spectrum Command [76].

2.3.2 Business Educational Computer Games

In 1955, a simulation called Monopologs, which required players to perform as inventory managers in a simulated U.S. Air Force supply system, was the origin of business simulation games. Inspired by the reported success of Monopologs, the first useful business game, Top Management Decision Simulation, was developed by the American Management Association in 1956. Then in 1957, the Business Management Game was developed for McKinsey & Company. In addition, the same year reveals the first use of a business game in a university classroom when the University of Washington used a simulation developed by Schreiber in a business policy course. Since 1958, the number of business games has grown rapidly [77], and they have been used for teaching a variety of business-related topics, such as running a company or managing a business. Exemplars of such games are Railroad Tycoon for running a Railroad company, Oil Emporium for running an oil business, Car Tycoon for building up a car company, Pizza Tycoon for making a pizza empire, and Capitalism II where the player learns how to lead a worldwide company in addition to the basics of the market [18].

2.3.3 Educational Computer Games

"*Edutainment*" is a term derived from "education" and "entertainment" to refer to embedding instruction with some form of entertainment to teach viewers. The idea of combining fun with learning led to the appearance of educational media. The first use for educational media was in 1980 when Silent Instructional Films were used in schools museums of New York City. During the 1910's the use of instructional films expanded to schools to teach science concepts. However, the instructional

films market crashed at 1930 because of many problems, such as lack of seriousness, technical difficulties, copyright issues, and a conflict between commercial interest and the curriculum [78].

In 1933, after the invention of television, the University of Iowa have produced the first educational television program to teach college students. Later, the Sesame Street program have been launched in 1969 to teach children and prepare them for school. Sesame Street paved the way for the appearance of many educational programs, such as Dora the Explorer, Barney, Play House Disney, and many other programs. Furthermore, many television stations, such as Public Television, Discovery Channel, History Channel, and Animal Planet have specialized in educational content. While the evaluations of the effectiveness of educational programs are controversial, educational media gain support for preparing children for school [79].

The interactive ability of computers motivated the use of computer in instruction and led to the development of the first CAI system in 1960's [80]. Computer Assisted Instruction (CAI) refers to drill-and-practice simulation or tutorial activities offered as stand-alone activities or with limited guidance from teachers. The use of computers in school for educational purposes expanded through the 1980's and 1990's under many terms, such as Computer-Based Education (CBE), Computer Supported Collaborative Learning (CSCL), Computer-Based Instruction (CBI), and Multimedia Learning. Multimedia Learning refers to the encoding of the content in different modalities, such as text, pictures, video, and sound. Although many researches showed that CAI has positive effects on education, CAI has always been criticized because of the drill-and-practice approach and the extrinsic motivation [80].

The shortness in drill-and-practice educational systems contributed to the appearance of educational computer games, which benefit from the intrinsic motivation feature of computer games to drive students to learn through playing [28]. The development of educational computer games started in research centers with research-based educational computer games. However, the success of research-based games, when presented as commercial products, led to the production of educational computer games (edutainment) by many commercial companies. Nevertheless, the failure of edutainment games to maintain good reputation has contributed to the appearance of new types of

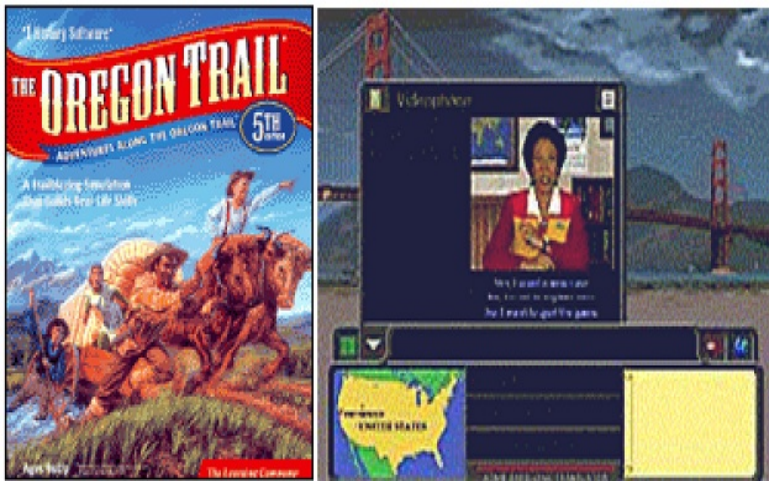

Figure 2.7: Oregon Trail Game

educational computer games. These are commercial computer games that are not designed, specifically, with educational goals, yet they have been used by many schools to teach students some general topics.

Research-Based Educational Computer Games

These are titles that have been designed by researchers as a result of new ideas or to prove some learning theories so they mostly contribute new innovation; however, they do not spread as commercial products because of limited budgets. The first attempt in using games in an educational product was in 1960 when the University of Illinois built a computer system named Plato [81] that was designed especially for general education using games. Between the 1970s and 1980s, many computer games were developed for Plato, such as Empire (a tactical multiplayer game where players captained a starship and fire at each other), Panther (a tank war game), FreeCell (a solitaire card game), and Avatar (a text and graphics-based multiuser game) [82]. The first known educational game is Oregon Trail (Figure 2.7), which was produced in 1971 to teach students about US history by following the footsteps of the early American settlers and helping students make decisions concerning the journey. The player assumes the role of a wagon leader guiding his party of settlers from Missouri to Oregon. The original game was developed by the MECC research center then released as a commercial product by the Learning Company [83] [84].

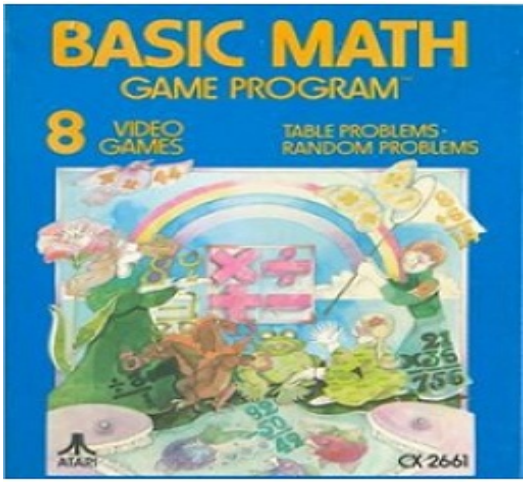

Figure 2.8: Basic Math Game

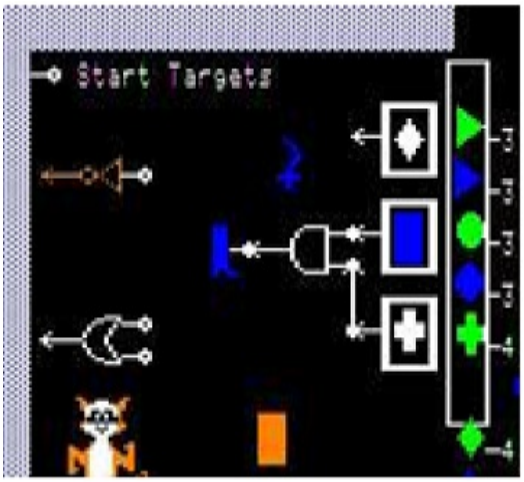

Figure 2.9: Rocky Boots Game

Commercial Educational Computer Games "Edutainment"

Most edutainment games have been developed by commercial companies to teach students basic skills, such as reading and some k-12 curriculum subjects, such as math, science, social studies, and language arts. Early educational computer games that have been released as commercial products includes: Basic Math (Figure 2.8) was produced by Atari in 1977 to teach math by giving players a set of ten basic math problems, such as simple addition, subtraction, multiplication, or division [85]. Electric Company Math Fun (1979) teaches math by allowing players to control gorillas to compete against each other by answering questions [86]. Rocky Boots (1982) uses an interactive graphical simulation (Figure 2.9) to teach simple digital logic circuits for upper-grade-school students [87].

Adventure games were extrapolated from the first computer games that were developed for education. In an adventure game, the player must work for the knowledge, solve meaningful puzzles, and travel through a meaningful game world. Over the years, many adventure games were developed, such as Snooper Troops (1982), which facilitates better problem-solving skills in children by allowing them to play detectives and solve different assignments [88]. Another game, In Search of the Most Amazing Thing (1983), requires players to explore a science-fiction universe with their long-lost uncle Smoke Bailey [89]. Carmen Sandiego was developed in 1985 to teach players about geography and history by solving some crime cases. The first title in the series Where in the World is Carmen Sandiego? was developed by Brøderbund Software while the last The Secret of the Stolen

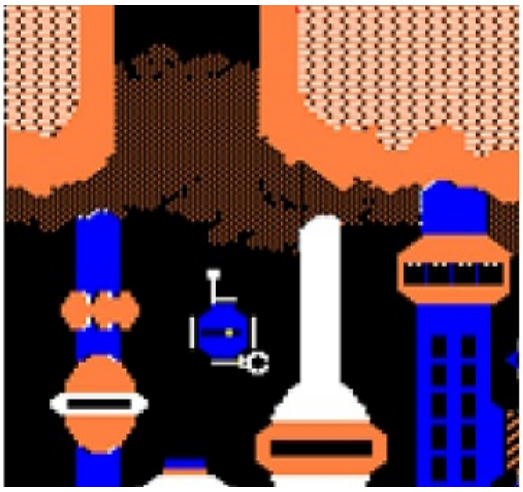

Figure 2.10: Robot Odyssey Game

Figure 2.11: Incredible Machine

Drums (2004) was developed by Bam! Entertainment using real-time 3D computer graphics [90] [91].

In 1984, Seven Cities of Gold was released as the first educational game with an edutainment title. It is an adventure game where the player assumes the role of a late-15th century explorer from Spain, setting sail to the new world to explore the map and interact with the natives to win gold and please the Spanish court. It was very successful title and paved the way for many edutainment titles that followed [92]. In the same year, Robot Odyssey (Figure 2.10), which was released by the Learning Company, constituted an adventure and puzzle game in which the player must escape from Robotropolis, a dangerous underground robotic city, by programming robots to solve various mind-bending puzzles [93]. Mavis Beacon (1987) was another successful edutainment title by the Learning Company. It lets the player set up different races against time as she types the best she has learned [94]. Incredible Machine (1993) was an interesting title (Figure 2.11) from Sierra Entertainment with general objective to create a series of Rube Goldberg devices by arranging a given collection of objects in a needlessly complex fashion to perform some simple tasks [95]. Typing of the Dead (2000) was another typing trainer by Sega in which characters have keyboards strapped to them, and the player needs to type out the words attached to the zombies in order to kill them [96].

Many edutainment games have been centered upon preschool and early school children to cover

Figure 2.12: Disney Toon Website

Figure 2.13: The Grand Prix Website

basic areas like arithmetic, reading, and spelling. Of these titles, usually released in series by several companies, the most noticeable one is The Learning Company, which produced several famous titles, such as Reader Rabbit (1989) teaches children from infancy through second grade basic skills [97]. Gizmos & Gadgets (1993) to teach children simple machines [98], Zoombinis (1995) teaches problem solving, math, and science using puzzles [99], and Clue Finders (1998-2002) teaches the third through ninth grades curriculum subjects [100]. Another company is the Knowledge Adventure, which released Jump Start series in 1994. The series is designed for a variety of ages and combine adventure, puzzles, and problem solving. In general, its games designed for younger children tend to include more activities than those for older children [101]. A relatively recent series is the Math Missions (2003) by Tom Snyder Company, which makes players build basic math and problem-solving skills while exploring different activities [102].

Since the Internet started connecting the whole world, online games have become more and more popular. EduProfix (2003) is a racing educational online game in which the player must answer correctly the given questions to get more seconds to win the race [103]. GCompris (2003) is an educational software suite comprising of many activities for children aged two to ten, including some computer games and covering many subjects, such as computer discovery, algebra, science, and geography [104]. Disney Toontown (Figure 2.12) a 3D online game has been launched by Disney in 2003 for children ages seven and older, teaches them many basic and critical thinking

skills [105]. Arcademic Skill Builders (2007) is a web site that offers many online games, which incorporate features of arcade games and educational practices to motivate and teach students basic skills in math and reading. One of its games is the Multiplication Grand Prix (Figure 2.13) in which the players move cars by answering some multiplication questions [106].

Despite their dominance in the educational computer games market, edutainment titles started to lose their good reputation over the years, for several reasons: they used the game-play of famous titles, with some changes in graphics only without contributing new ideas; they produced titles that have been seen as low-budget titles aimed at early school children, but did not compete with commercial titles [18].

Commercial Computer Games Used for Education

Many commercial computer games, which have been designed with no educational goals, turned to be useful in education because of their strong motivation. Exemplars are SimCity (1989) about starting a city and maintaining its growth as a mayor, Lemmings (1990) about arranging misguided Lemmings in the right way, SimEarth about the development of the earth from prehistorical times until today, and Civilization (1991) about managing different civilization areas, such as technology, trade, and city management [18]. Other game, the Big Brain Academy (2007) by Nintendo, features a variety of puzzles, mathematical questions, and Sudoku puzzles, all designed to help keep certain parts of the brain active [107].

2.3.4 Serious Games

Serious games are computer games that have been developed for purposes other than entertainment, such as training, advertising, simulation, or education. The failure of edutainment computer games to innovate in preceding years, in addition to the advanced developments in the technological abilities of computer games hardware and software, and the increased number of people playing computer games each year, all encouraged researchers to re-examine the use of computer games for serious purposes other than entertainment.

To encourage the development of serious games, the "Serious Games Initiative" was launched

Figure 2.14: The Hephaestus Game

Figure 2.15: The Sole Survivor Game

in 2002 by the Woodrow Wilson International Center for Scholars. Initially, it supported the development of serious games that address policy and management issues then extended to other fields by adding more focus sub-groups, such as Games for Change, which focuses on social issues and social change, and Games for Health, which addresses health care applications [108].

Research-Based Serious Games

Since 2001, many research projects started to focus on serious games development. The Games to Teach project was launched at 2001 by Henry Jenkins at MIT in collaboration with Randy Hinrichs at Microsoft Research [109]. The project ended in 2003 after producing 15 interesting frameworks for computer games to teach college-level math, science, and engineering concepts. One example of these games is the Hephaestus game (Figure 2.14), a multiplayer Xbox Online game to teach engineering and physics concepts by requiring the player to build robots to colonize the planet Hephaestus. Another game is Sole Survivor (Figure 2.15), which teaches psychology by applying psychology experiments on the player in an alien vessel. The Periodista, another game the project produced, teaches Spanish by requiring the player to go on photographic missions in Spanish-speaking countries [110]. Another learning and teaching project is The Quest Atlantis (QA) (2001) that uses a 3D multi-user environment to immerse children, ages 9-12, in educational tasks. Building on

Figure 2.16: The Making Games Project

Figure 2.17: The Immune Attack Game

strategies from online role-playing games, QA combines strategies used in the commercial gaming environment with lessons from educational research on learning and motivation. It allows users to travel to virtual places to perform educational activities (known as Quests), talk with other users and mentors, and build virtual personae [111] [112]. Between 2003 and 2006, The Making Games project (Figure 2.16) was conducted by London Knowledge Lab to develop prototype software for authoring computer games to allow students at schools to learn how to design their own games. The project was completed in 2006 [113]. In 2007, the Immune Attack (Figure 2.17), a single-player serious game, was developed by the Federation of American Scientists. It combines a realistic 3D depiction of biological structure and function with educational technologies for teaching immunology to high school students and college freshmen. In the game, the players must train various immune system components, such as blood cells to protect the human body through the game-play [114].

Commercial Serious Games

In addition to the research-based serious games, several serious games have been developed as commercial products. Colonize with Bots (CoLoBoT) (2002) is a serious game (Figure 2.18) that teaches programming languages by requiring players to program robots to prepare fictional planets for human race colonization by establishing basic infrastructure on the surface and eliminating any

Figure 2.18: Colobot Game

Figure 2.19: Plant Game

dangers [115]. The Genomics Digital Lab: Plant (2007) is an interactive serious game (Figure 2.19), which simulates a plants lab and allows players to alter the environment to figure out how to make the dying plant survive and thrive using the information learned in the game [116]. Global Conflict: Palestine (2007) is a serious game (Figure 2.20) to educate about the Palestinian and Israeli political conflict. The player acts as a freelance journalist who must collect information about the conflict from inside the game to write an article for a newspaper. At the end of the game, the submitted story by the player reflects his point of view about the conflict [117].

2.3.5 New Research Area for Educational Games

Over the years, educational computer games (edutainment) have been developed to teach many school subjects, such as math, science, language arts, and programming. Later, serious games have been used in many new areas other than education, such as politics, health, and social sciences [118] [119]. However, as the previous sections show no computer game has been designed either to teach or to visualize algorithms. Therefore, using computer games to visualize algorithms, in this research, adds a new dimension to educational computer games usages and paves the way for more research in this new area.

Figure 2.20: Global Conflict–Palestine Game

2.4 Game Development Technologies

A major evolution in game development technologies and techniques has occurred since its early days. This section presents a brief summary of some computer games development tools.

2.4.1 Multimedia Application Program Interfaces (APIs)

The Multimedia Application Program Interfaces are programming libraries that support the creation of games and other multimedia applications (Figure 2.21) and can be divided into:

1. Low-level Libraries that can be used to directly access the machine hardware (graphics devices, sound cards, input devices, etc.). Examples include DirectX by Microsoft [120] and OpenGL by Silicon Graphics [121], both multimedia libraries that provide 2D and 3D graphics rendering and acceleration features.

2. High-Level Libraries, such as Microsoft XNA Game Studio, which is a set of tools based on Microsoft Visual C# that allow students and hobbyists to build games for both Microsoft Windows and Xbox 360. It also includes the XNA Framework, which is a set of managed libraries based on the Microsoft .NET Framework that are designed for game development [122].

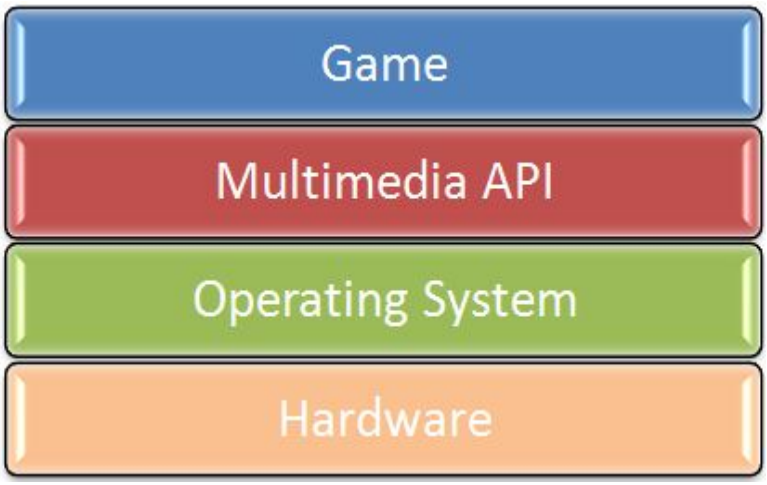

Figure 2.21: Multimedia (APIs) and Game Development

Multimedia Application Program Interfaces (APIs) set a new stage in game development by empowering programmers with more abstraction to experience an easier game development process. However, they provide only features that are generic for computer games development, but do not offer the abstraction level desired to design specific games. Additionally, interaction with such APIs can only be done programmatically, not visually. Even for XNA Game Studio, which provides some game-related APIs, the designer still must implement many operations and functions related to the game.

2.4.2 Visual Game Creation Tools

With the intention to simplify game development and make it more accessible to a broader range of communities, visual game creation tools are created (Figure 2.22). They help users to create complete games with no programming at all by providing graphical and easy-to-use interfaces for creating sprites, entities, sound, screens, and other resources to the game. Examples of such tools are:

1. Game Maker: an Integrated Development Environment (IDE) for creating (2d) computer games. All elements of a game should come together: graphics, sound, music, gameplay, etc. It is simple to use, with drag-and-drop like features and capabilities. However, considerable power is retained with the use of a built-in scripting language [123].

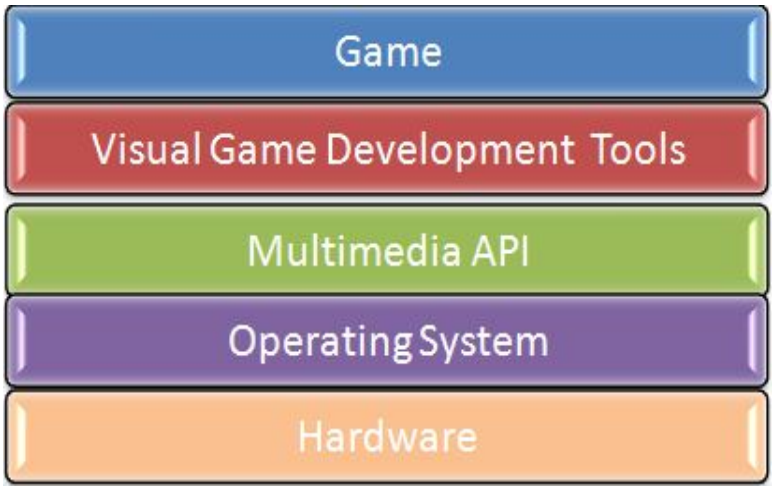

Figure 2.22: Visual Game Creation Tools and Game Development

2. Game XML: a collection of XML specifications that describe and script computer simulation engines. Developed by the XML Game Consortium (XGC), it is an on going project to create a reusable, standards-based architecture that can be applied toward computer games and simulations [124].

Visual game creation tools were certainly a great achievement in order to help beginner or amateur game designers and programmers to accomplish their tasks. However, they are not adequate for real-world games, since they lack the support for real programming languages. Some of these tools provide visual programming constructs for use in simple cases, but not when more elaborated behaviors are desired. Other tools provide simple (script) programming languages for advanced users; however, this means these users are required to learn a new language, which is error-prone inside an environment that was not originally conceived for codification.

2.4.3 Game Engines

Game engines are used to automate game-specific functions like loading and rendering animations, playing digital sounds, controlling input devices like joysticks, and simulating physics-based special effects [125]. Specifically, a game engine is a reusable API that gathers common game development foundations (entity rendering, world management, game events handling, etc.) and provides to developers a programmatic interface in which game behavior can be specified (Figure 2.23). A

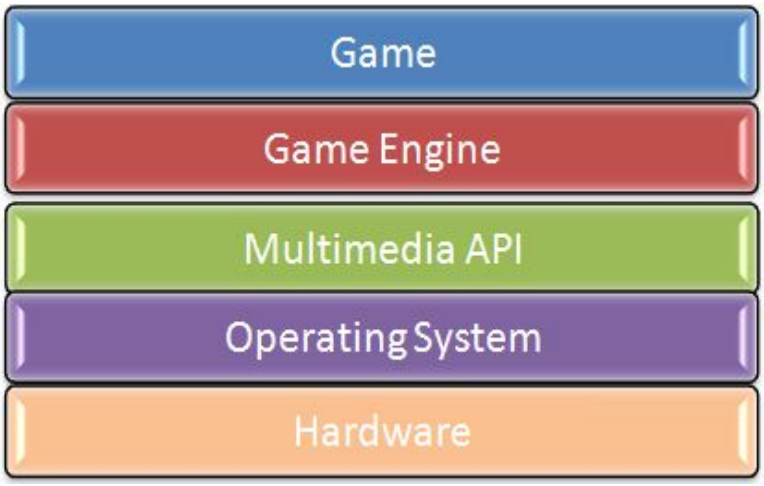

Figure 2.23: Game Engine and Game Development

comprehensive list of game engines can be found in the DevMaster.net engine database, such as 3DGameStudio, C4 Engine, Crystal Space, and Panda3D [126]. Examples of game engines currently available to game developers are:

1. OGRE (Object-Oriented Graphics Rendering Engine): one of the most popular free and open-source 3D engines. It is a scene-oriented, flexible 3D engine written in C++ [127].

2. Torque: a fully featured commercial game engine. Its features include object-oriented design, built-in tools (World Editor, GUI Editor, Terrain Editor, etc.), a rendering engine, a networking system, artificial intelligence (in the form of finite state machines), a physics engine, a particle engine, support for sound (including 3D sound), video, lighting, textures, shadows, scene management, animation, meshes, special effects (sky, water, decals, fog, etc.), and fonts [128].

3. GEDI: provides a 2D game design in C++, and is designed with educational use in mind. Source code can be downloaded and is currently available without license, but GEDI is still in the early phases of development [129].

Game engines are the state-of-the-art tools in computer games development. However, some disadvantages can be related to them. For example, where some game engines provide game programmers with visual tools to help them accomplish some tasks, such as level editors and sprite editors,

these tools are focused on specific aspects of the game development process, but can not be targeted at the creation of a complete game, and their output must consumed programmatically in the game. Moreover, due to the inherent complexity of game engines, the learning curve for mastering these tools is somewhat high. Subsequently, using a game engine may involve considerable costs, such as acquisition costs, training costs, customization costs, and integration costs.

2.4.4 Domain-Specific Languages (DSL)

A Domain-Specific Language (DSL) is a programming language geared towards solving problems in a specific domain. Some game development tools use DSLs to provide a higher level of abstraction in the game design process, such as:

1. Sharpludus Software Factory: an extension to the Visual Studio .NET development environment aimed at computer games development industrialization in the .NET Platform. It supports the design of 2D adventure games using a visual designer based on a domain-specific language, semantic validators, and code generators [130].

2. Cannibal Game Development Platform: an IDE for developing computer games that integrate a DSL language [131] with a commercial game engine called Cannibal Engine [132].

The use of (DSL) is a new trend in game development technology. However, it is in the early experimental stages with some prototypes targeting specific game genres and needs more research and investigating.

2.4.5 Algorithm Game Development Tool

This research introduces a new integrated development environment, namely Algorithm Game Designer, tailored to create computer science educational games for Windows platform that takes advantage of the previously described game development tools in its design, while avoiding their pitfalls (Figure 2.24). First of all, to benefit from multimedia API, the designer has been built on top of XNA Game Studio, which has been built on DirectX API. However, it is a complete, stand alone IDE that supports the C# programming language by providing debugging and compiling operations

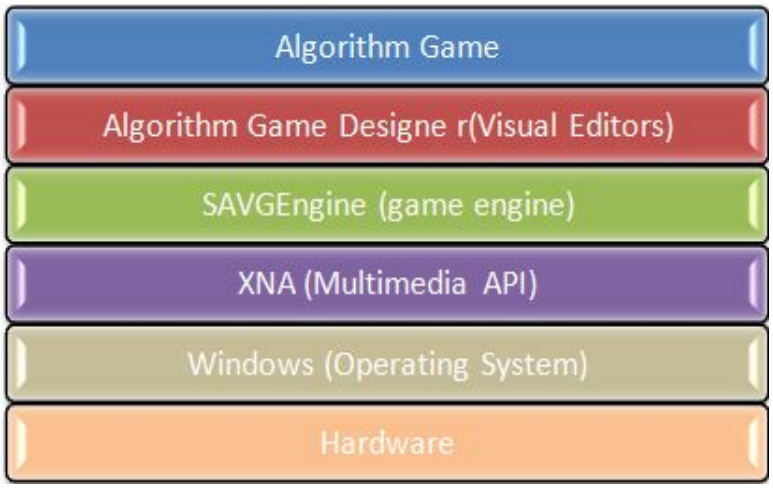

Figure 2.24: Algorithm Game Designer and Game Development

to the designers. Moreover, it includes a game engine that supports the development of computer games by providing modules to handle input, sound, rendering, and other game functions. Nevertheless, to support visual development, it has five visual editors to support the creation and loading of the game entities and sprites, visually without any coding. It also provides a game template that provides the game content and basic game screens in addition to the Base Game and Play Game classes that provide the game skeleton. Finally, it includes a repository of built in game content items that can be loaded directly into the game using the graphical editors.

Chapter 3: Theoretical Background

This chapter defines the theoretical background that has been used in conducting this research. The first section explains three educational theories: Behaviorism, Cognitivism, and Constructivism, in addition to Motivation theory, which has been used in algorithm games design. The second section defines two learning models: Bloom's Taxonomy of Educational Objectives and Gagne's Model of Instruction, which are used to construct the learning models of the AVuSG approach. The last section gives an overview of algorithm genres and data structure types and operations.

3.1 Learning Theories

Learning theories provide a conceptual framework for interpreting the examples of learning and suggest where to look for solutions to practical problems. In general, learning theories fall into three main categories: behaviorism, cognitivism, and constructivism.

3.1.1 Behaviorism

Behaviorists treat the learner as a black box that receives environment stimulation and responses by changing the environment in ways that increase or decrease the likelihood of the same response in the future. Behaviorism focuses on a new behavioral pattern being repeated until it becomes automatic [133]. Several critical concepts of learning can be defined in the context of Behaviorism theory, such as Classical Conditioning, Operant Conditioning, and Laws of Exercise and Effect.

Classical Conditioning

Classical conditioning is defined by Ivan Pavlov as the process of associating a previously neutral stimulus (a stimulus with no response) with an unconditioned stimulus (a stimulus that produces

some observable response without prior learning known as an unconditioned response) to evoke a conditioned or learned response [133].

Operant Conditioning

According to Burrhus Frederic Skinner, learning occurs as a result of consequences, since consequences largely determine whether a person will repeat a behavior or not. Consequences that tend to strengthen the recurrence of a particular behavior are called reinforcement. Any action that decreases the behavior is punishment [133].

Laws of Exercise and Effect

According to Thorndike, the law of exercise states that repetition is crucial to learning, and the law of effect states that providing a reward after a response can strengthen it.

Behaviorist Theory Influence on Computer Games

The behaviorist theory is apparent in most educational computer games or edutainment games. In these games, the player practices a specific area through repetition while receiving rewards after each proper response; such games may be easiest to design [133].

3.1.2 Cognitivism

Cognitive theorists argue that learning is a more complex process that utilizes problem-solving and insightful thinking in addition to repetition of a stimulus-response chain. In contrast to behavioral learning theory, cognitive learning theory explains learning by focusing on mental process [134]. Memory and Schema models are some concepts that have been defined by the Cognitive theory.

Memory Model

According to Atkinson and Shiffrin [134], the human memory (Figure 3.1) has three main components:

1. Sensory Registers: receive stimuli from the environment (sights, sounds, smells, etc.) and hold it for a very short time. By paying attention to or focusing on, certain stimuli can enter short-term memory.

2. Short-Term Memory: is limited by the number of bits of information that it can hold at any one time: five to nine separate items is the maximum. The duration ranges from about 20 to 30 seconds at the most. Information in short-term memory must continually be rehearsed or it will be forgotten.

3. Long-Term Memory: stores a large amount of information for a long time. There are three categories of long-term memory:

    - Semantic memory for meaning that stores information in the form of a proposition, images, and schema.
    - Episodic memory for information tied to a particular place and time.
    - Procedural memory for how to do things.

Schema Model

Schemes are the basic building blocks of thinking; they are organized systems of actions or thought that allow humans to mentally represent the objects and events in the world. According to Piaget [134], information is stored in long-term memory in a network of connected facts and concepts that provide a structure for making sense of new information. When encountering new concepts, learners attempt to assimilate them into the existing schema and decide what should be done. If what is encountered is different from what is understood, disequilibrium is said to occur. Learners will try to reduce disequilibrium by developing new schema or adapting old ones until equilibrium is restored.

Cognitivist Theory Influence on Computer Games

Most games that need internal mental processing to play involve cognitive learning; some examples are adventure games, strategy games, and all forms of puzzle games.

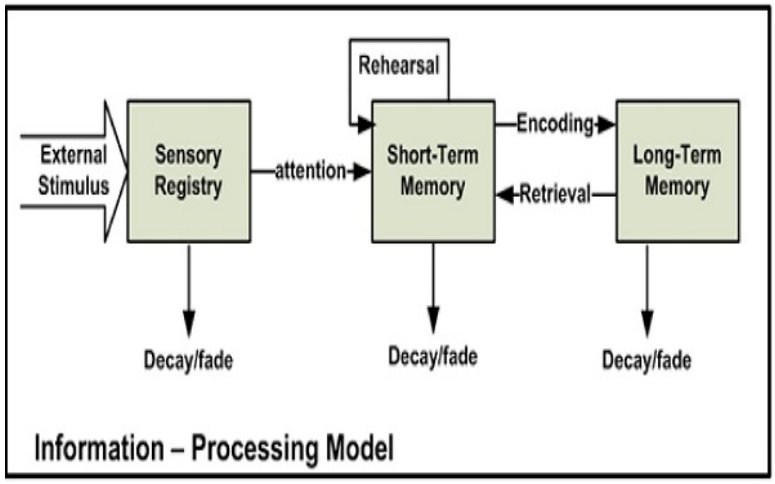

Figure 3.1: Information Processing Model

3.1.3 Constructivism

Constructivism states that human beings actively construct knowledge for themselves through their engaged interaction with the environment, previous experiences, and prerequisite knowledge that will enable them to construct meaning for new information. Moreover, different people may learn different things from the same information.

Social Development Theory

The Social Development Theory of Vygotsky is one of the foundations of constructivism. Vygotsky focused on the connections between people and the sociocultural context [135]. He asserts three major themes:

- Social interaction plays a fundamental role in the process of cognitive development.
- Anyone who has a better understanding of a concept or task is known as the More Knowledgeable Other (MKO).
- Learning occurs in a Proximal Development Zone, which is the distance between performing a task under guidance or independently.

Constructivism Impacts on Learning

Constructivism influenced learning in three ways:

- Curriculum: using curricula customized to the student's prior knowledge instead of standardized curriculum.
- Instruction: making connections between facts, fostering new understanding in students, tailoring teaching strategies to student responses and encouraging students to analyze, interpret, and predict information.
- Assessment: eliminating grades and standardized testing and assessment becomes part of the learning process so that students may judge their own progress.

Constructivist Computer Games

Constructivist approaches are promoted by Bailey [136] through encouraging students to build their own games with independent learning. Issing's [137] suggestions reflect constructivist approaches with learner-oriented study, imaginative education in environment, active learning, and open study. Computer-based constructivist approaches should comprise "unpredictability and offer a relaxed set of control for the user, creating a space in which teaming can occur" [19]. However, developing games and simulations that reflect more interactive and constructivist approaches can be difficult for a game designer [138].

3.1.4 Motivation Theories

Motivation is concerned with the factors that stimulate or inhibit the desire to engage in a behavior. Malone and Lepper [28] distinguish between extrinsic and intrinsic motivation as follows:

- Extrinsic Motivation: supported by factors external to the activity, i.e., when people perform a task because some force, either external to them (money, rewards, grades, and punishment) or internal to them (a value or belief that impacts their self-worth) drives them to perform.

- Intrinsic Motivation: arises directly from doing the activity, i.e. what people will do without external inducement. Intrinsically motivating activities are those in which people will engage for no reward other than the interest and enjoyment that accompanies them. Intrinsic motivation occurs when people are passionate about a task and perform it for the sheer pleasure of doing it.

Factors to Enhance Motivation

There are several factors that enhance the motivation of the learner. Some of these factors are individual in the sense that they operate even when a learner is working alone. Other interpersonal factors play a role only when someone else interacts with the learner [139].

Individual Factors to Enhance Motivation:

- Challenge: people are best motivated when they are working toward personally meaningful goals whose attainment requires activity at a continuously optimal (intermediate) level of difficulty.
- Curiosity: something in the physical environment attracts the learner's attention or there is an optimal level of discrepancy between present knowledge or skills and what these could be if the learner engaged in some activity.
- Control: people have a basic tendency to want to control what happens to them.
- Fantasy: learners use mental images of things and situations that are not actually present to stimulate their behavior.

Interpersonal Factors to Enhance Motivation:

- Competition: learners feel satisfaction by comparing their performance favorably to that of others.
- Cooperation: learners feel satisfaction by helping others achieve their goals.

- Recognition: learners feel satisfaction when others recognize and appreciate their accomplishments.

3.2 Models of Learning

Models of learning and instruction attempt to examine all the elements that contribute to learning and to organize these in a systematic way that can easily be applied to learning situations. Models of learning are valuable because they provide ways to think about the process of learning, to analyze learning problems, and to plan instruction. Two main learning models can be distinguished: Bloom's Taxonomy of Educational Objectives and Gagne's Model of Instruction.

3.2.1 Bloom's Taxonomy of Educational Objectives

Bloom [140] defined the following objectives for any learning process:

- Knowledge: is defined as the remembering of previously learned material. This may involve the recall of a wide range of material, from specific facts to complete theories, but all that is required is the bringing to mind of the appropriate information. Knowledge represents the lowest level of learning outcomes in the cognitive domain.
- Comprehension: is defined as the ability to grasp the meaning of material. This may be shown by translating material from one form to another (i.e. interpreting charts and graphs).
- Application: refers to the ability to use learned material in new and concrete situations.
- Analysis: refers to the ability to break down material into its component parts so that its organizational structure may be understood.
- Synthesis: refers to the ability to put parts together to form a new whole.
- Evaluation: is concerned with the ability to judge the value of material.

3.2.2 Gagne's Model of Instruction

Gagne [141] described eight phases that occurred during the learning process as follows:

1. Attention: learning is not likely to occur in the absence of attention. Attention is essential for getting information into the working memory and keeping it active there.

2. Expectancy: during this phase, the learner develops an expectancy that something desirable will happen as a result of the proposed learning process. The result is a motivation to engage in the subsequent phases of the learning process.

3. Retrieval of Relevant Information to Working Memory: the learner retrieves from long-term memory the structures that will be helpful in learning new information or solving problems that have been encountered.

4. Selective Perception: during this phase, the learner focuses attention on the essential features of the instructional presentation.

5. Encoding: entry of Information into Long-Term Storage. During this phase, the learner encodes the information–that is, transfers the information into long-term memory by relating it to information that is already stored there.

6. Responding: during this phase, the learner retrieves and actively uses the information that has been stored in long-term memory. The learner demonstrates through an active performance that the learning has taken place.

7. Feedback: during this phase, the learner determines the degree to which the performance during the previous phase was satisfactory. When the feedback indicates acceptable performance, this usually serves as reinforcement to the learner.

8. Cueing Retrieval: during this phase, the learner practices recalling or applying the information after it has been initially learned in order to enhance retention of the information or to transfer the learning beyond its original context to a new application.

Depending on the previous nine events Gagne, Briggs, and Wager [141] have determine nine events of instruction to be:

1. Gaining attention.
2. Activating motivation.
3. Stimulating recall of prerequisite learning.
4. Presenting stimulus material.
5. Providing learning guidance.
6. Eliciting the performance.
7. Providing feedback.
8. Assessing the learner's performance.
9. Promoting retention and transfer.

3.3 Algorithms and Data Structures Overview

This section defines algorithms and data structures, and then explains data structure basic operations and algorithms classifications.

3.3.1 Data Structures

A data structure in computer science is a way of storing data to be used efficiently. Formally, a data structure is a representation of a finite data set [2]. Data Structures examples are Array, List, Linked list, Doubly linked list, Stack, Queue, Hash table, Graph, Heap, Tree, Binary Search tree, Red-Black tree, etc.

Data Structure Basic Operations

Queries and Modifying are the two kinds of operations that can be applied on a data structure:

1. Queries operations that get information about the data structure are as follows:

    - Search (data structure, key): searches for a given key in a data structure.
    - Sort (data structure): sorts elements of a data structure.
    - Minimum (data structure): finds the element with the minimum value in a data structure.
    - Maximum (data structure): finds the element with the maximum value in a data structure.
    - Successor (data structure, element): finds the element that succeeds the given element in a data structure.
    - Predecessor (data structure, element): finds the element that precedes the given element in a data structure.

2. Modifying operations that change the data structure are as follows:

    - Insert (data structure, element): inserts an element into a data structure.
    - Delete (data structure, element): deletes an element from a data structure.

3.3.2 Algorithms

An algorithm is a sequence of computational steps that transform the input into the output [2]. Algorithms can be classified according to the problem-solving approach that they use or the problems that they solve.

Algorithms use a similar problem-solving approach:

- Recursive Algorithms: convert the problem into sub-problems, then solve each one using recursion.
- Backtracking Algorithms: return a solution if found or recur through the problem with every possible choice until solution or failure is reached.

- Divide and Conquer Algorithms: divide the problem into smaller sub-problems of the same type, and solve these sub-problems recursively, then combine the solutions to the sub-problems into a solution to the original problem.
- Dynamic Programming Algorithms: find the best solution of multiple exist solutions. Examples are Knapsack and Activity Selection Problem.
- Greedy Algorithms: get the best solution at the moment, without regard for future consequences. By choosing a local optimum at each step, it will end up at a global optimum. Examples are Prim's and Dijkstra's algorithms.
- Branch and Bound Algorithms: a tree of sub-problems is formed.
- Brute Force Algorithms: try all possibilities until a satisfactory solution is found.
- Randomized Algorithms: use a random number at least once during the computation to make a decision.

Algorithms solve similar problems:

- Sorting Algorithms: Bubble Sort, Selection Sort, Insertion Sort, Shell Sort, Merge Sort, Heap Sort, Quick Sort, Bucket Sort, etc.
- Linear-Time Sorting: Counting Sort, Radix Sort, Bucket Sort, etc.
- String Matching: Naïve String Matching, Knuth-Morris-Pratt, Boyer-Moore, etc.
- Graph Algorithms: Breadth First Search (Bfs), Depth First Search (Dfs), Topological Sort, Strongly Connected Components, Generic Minimum Spanning Tree, Kruskal'S, Prim'S, Single Source Shortest Path, Dijkstra'S, etc.
- Searching Algorithms:
  - List Search: Linear Search, Binary Search, etc.
  - Tree Search: Breadth First Search, Depth First Search, etc.

- Informed Search: Best-First Search, A*, etc.

Chapter 4: Algorithm Visualization using Serious Games (AVuSG)

Once the requisite background in algorithm visualization, learning theory, educational models, and educational games has been provided, this chapter describes the Algorithm Visualization using Serious Games (AVuSG) approach. AVuSG illustrates how computer games can be used in algorithm learning and visualization. The first section explains the conceptual framework of AVuSG components, including algorithm representation forms, learning process, and learning models. The second sections gives a brief description of the software systems that must be implemented to illustrate AVuSG. (A detailed description of these systems implementations is given later in Chapter 7.)

4.1 AVuSG Conceptual Framework

The AVuSG conceptual framework consists of three main components: algorithm representation forms, learning processes, and learning models.

4.1.1 Algorithm Representation Forms

The algorithm representation form (visualization) is a visual depiction of the algorithm behavior that has been produced by a visualization system to visualize that algorithm. AVuSG produces four different forms of representations or visualizations for the algorithm to be learned:

1. Algorithm Text: a written description of the algorithm behavior that shows its basic idea and how it works. The algorithm text is written as steps using a pseudo-code (Figure 4.1) or ordinary English (Figure 4.2).

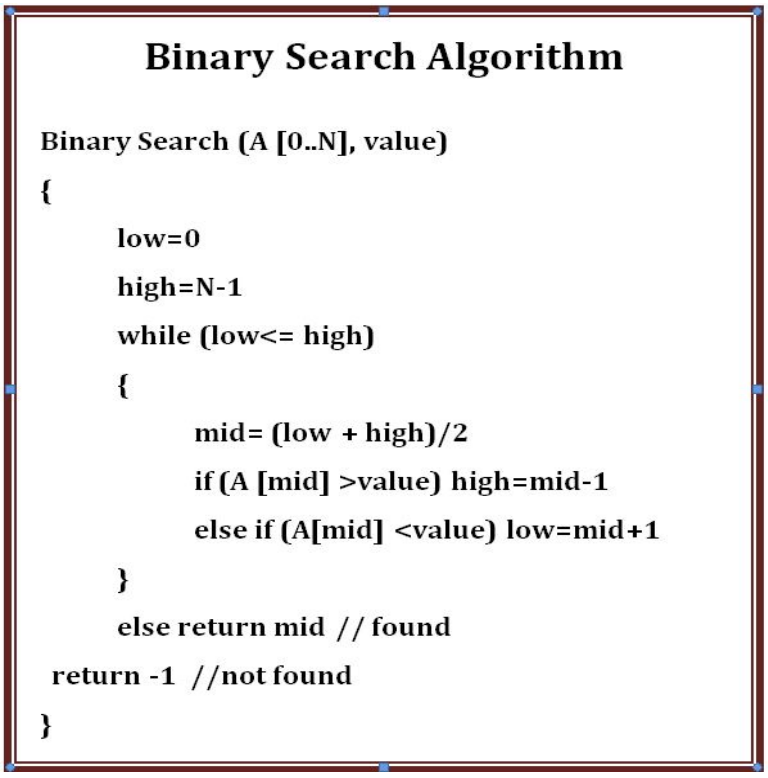

Figure 4.1: Pseudo-code of Binary Search Algorithm

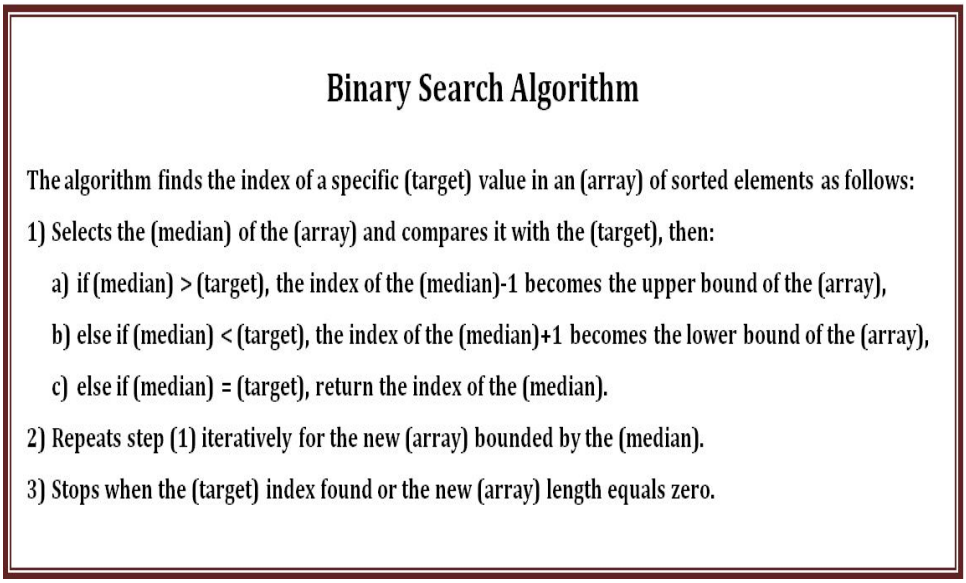

Figure 4.2: Description of Binary Search Algorithm Steps

2. Algorithm Flowchart: a graph depicting a static visualization of the algorithm functionality. It uses several shapes to describe the different states of the algorithm activities (Figure 4.3), such as making a decision, inputting data, and performing a calculation. It also uses arrows to define the algorithm behavior.

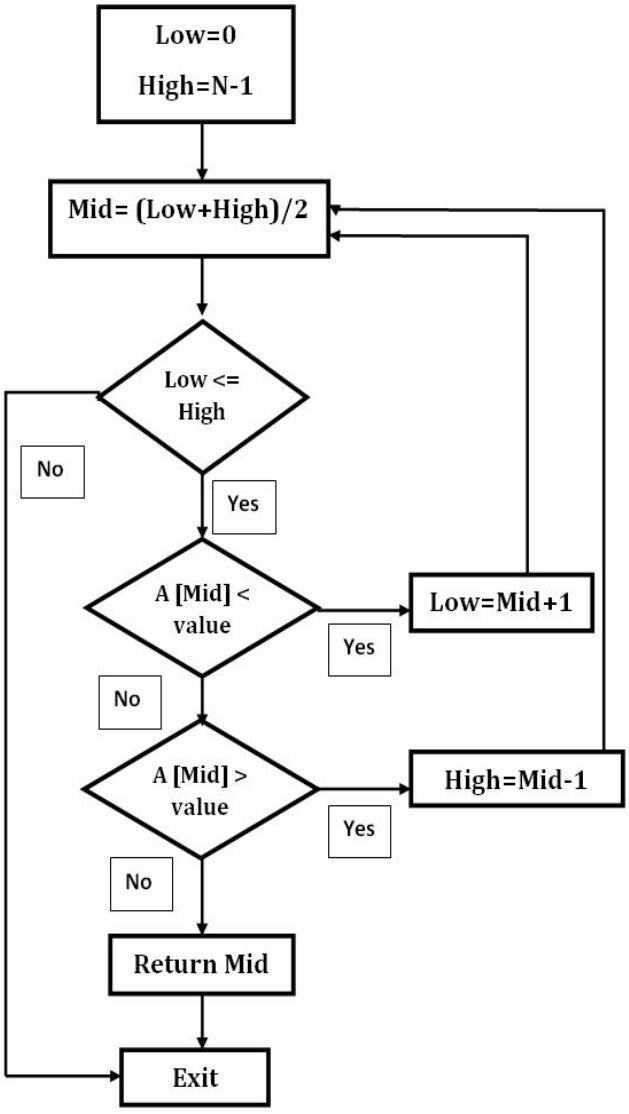

Figure 4.3: Flowchart of Binary Search Algorithm

3. Algorithm Game: an educational computer game with a game-play that simulates the behavior of the algorithm and graphics that depict the features of its data structure. (Algorithm games are described in more detail in Chapter 5.)

4. Algorithm Game Demonstration: a dynamic visualization of the algorithm operations using the self-running demo of the algorithm game.

4.1.2 Learning Processes

A learning process describes the type of activity the learner must perform to accomplish a learning objective. AVuSG defines three types of learning processes involved in learning an algorithm:

1. Viewing Process: during this process, the learner watches the algorithm text, flowchart, and game demo.

2. Playing Process: during this process, the learner plays the algorithm game that simulates the algorithm.

3. Designing Process: during this process, the learner develops the algorithm text, flowchart, and game.

4.1.3 Learning Models

In general, a learning model describes the organization of learning processes in a systematic way that can easily be applied to learning situations. AVuSG introduces three learning models that can be adopted to teach algorithms using computer games. Each of these learning models organizes the three previously described learning processes (viewing, playing, and designing) using a learning model or theory.

1. The Bloom Based Model

This model achieves the learning objectives of Bloom Taxonomy (Section 3.2.1). Figure 4.4 shows the flow between the model learning processes, while its detailed steps are explained as follows:

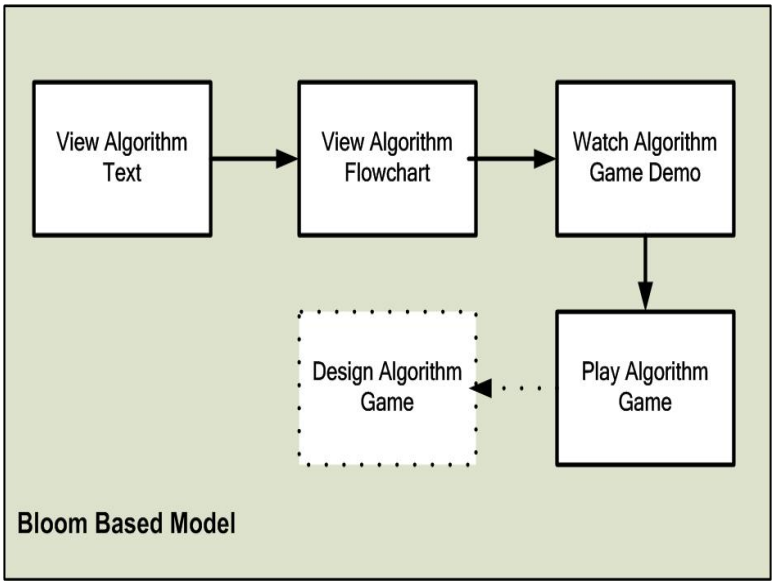

Figure 4.4: Bloom Based Model

1. The learning starts in the viewing process, where learners watch the algorithm text to build their knowledge.

2. Then, learners view the algorithm flowchart and game demo to comprehend how the algorithm works.

3. Next, in the playing process, learners apply their knowledge of the algorithm to play the algorithm game.

4. The game-play is divided into several game moves; each move simulates one algorithm step. While playing the game, learners first analyze the algorithm to determine the needed steps to win each game move, then synthesize all moves to win the whole game.

5. Learners can easily evaluate the speed, complexity, and efficiency of an algorithm by playing its related game. For example, if the game is hard to win, this means the algorithm is complex. If the game data is not large, this means the algorithm is suitable for small size data and so on.

6. Optionally, in the designing process learners analyze the algorithm and its played game to synthesize a new algorithm game design. Then, learners evaluate their designs for the new

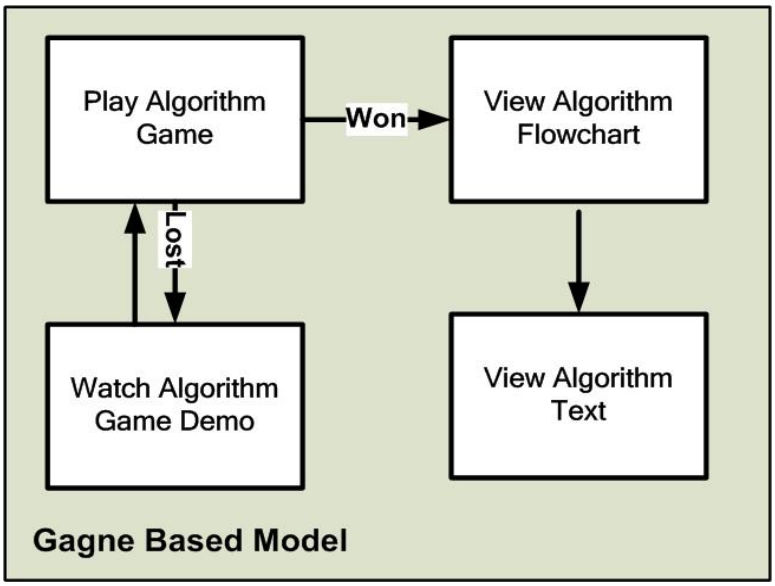

Figure 4.5: Gagne Based Model

algorithm game.

2. The Gagne Based Model

The steps of this model simulates the events of Gagne Model of Instruction (Section 3.2.2). Figure 4.5 shows the flow between the model learning processes, while its detailed steps are explained as follows:

1. The learning starts by playing the algorithm game that was created by the instructor, to gain the learners' attention and activate their motivation. If they lose the game, learners are provided with guidance through an algorithm game demo. The playing process builds and stimulates the required prerequisite information related to the algorithm.

2. After winning the game, learners are presented with the algorithm text and flowchart in the viewing process to learn the algorithm.

3. Eliciting and assessing the performance of learners can be achieved using the winning/losing criteria of the algorithm game.

4. Promoting retention and transfer can be acquired in the playing process.

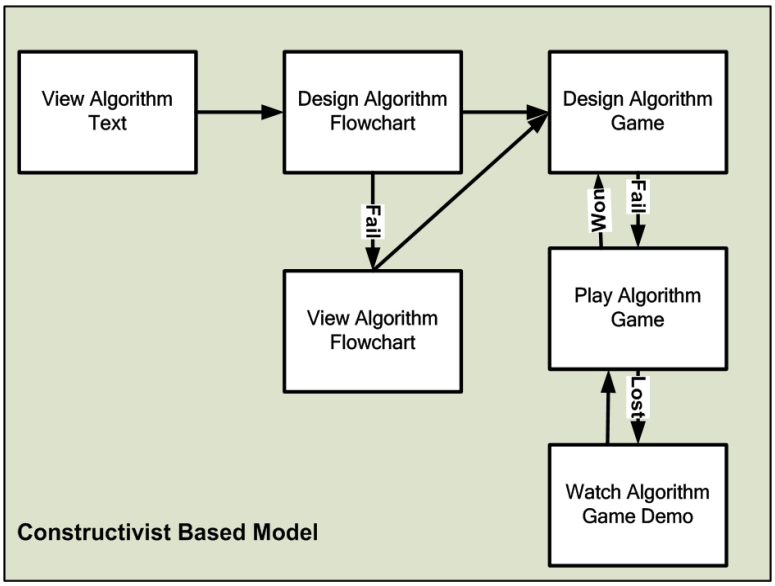

Figure 4.6: Constructivist Model

3. The Constructivist Model

In this model instructors encourage students to learn algorithms by making their own algorithm visualizations. Figure 4.6 shows the flow between the model learning processes, while its detailed steps are explained as follows:

1. Learners start learning by viewing the algorithm text to get an idea about how the algorithm works.

2. Learners then go to the designing process to design the flowchart of the algorithm.

3. If successful in building the flowchart, learners go to next step and start to design and develop their own algorithm games.

4. However, if learners fail any of the design processes, they can always watch the algorithm flowchart, game demo, and play the algorithm game that has been built by their instructors.

The model deploys the constructivist theory (Section 3.1.3) to affect learning in three ways:

1. Curriculum: for the same algorithm, each student builds his own algorithm game and flowchart based on his prior knowledge.

2. Instruction: students analyze, interpret, and predict information about the algorithm to design its flowchart and game.

3. Assessment: students learn the algorithm by playing its game and judge their own progress through the winning/losing criteria of the game. Therefore, the assessment becomes part of the learning process.

4.2 AVuSG Software Systems

The conceptual framework of the AVuSG approach presents three learning processes (viewing, playing, and designing), which the learner performs on three forms of algorithm visualizations–text, flowchart, and game. Therefore, six systems are needed, as follows:

1. Algorithm Text Viewer: a system to support the viewing process of the algorithm text.

2. Algorithm Flowchart Viewer: a system to support the viewing process of the algorithm flowchart.

3. Algorithm Game Viewer: a system to support the playing process of the algorithm game and the viewing of its demo.

4. Algorithm Text Designer: a system to support the designing process of the algorithm text.

5. Algorithm Flowchart Designer: a system to support the designing process of the algorithm flowchart.

6. Algorithm Game Designer: a system to support the designing process of the algorithm game.

The above six systems have been implemented and integrated in one system, namely Serious Algorithm Games Visualizer (Serious-AV), which is described in more detail in Chapter 7.

Chapter 5: Algorithm Games

This chapter introduces and defines Algorithm Games, which are computer games that have been developed to teach and visualize algorithms. The first section specifies algorithm games by defining them and describing their genres and attributes. The second section details some concepts related to computer games design in general, while the third section illustrates how these terms can be applied to design our algorithm games.

5.1 Algorithm Game Specification

In this section, algorithm games are defined using concepts related to instructional and simulation games. In addition, computer games genres are surveyed to define the genres that can be used to design algorithm games. Finally, the characteristics of educational games are used to derive the algorithm games attributes.

5.1.1 Algorithm Game Definition

Thiagarajan [142] defines an Instructional Game as an activity that has been designed specifically for achieving learning objectives, and a Simulation Game as a "correspondence between aspects of the game and selected aspects of the reality." In more detail, "the rules of a game may reflect real-world processes, and the game artifacts may represent real-world products" [142]. In fact, simulation games replicate a model of reality as observed by the designer. Accordingly, we can use the definitions of the instructional and simulation games to define our algorithm games as follows: an algorithm game is an instructional game that has been designed to teach and visualize an algorithm by simulating the behavior of the algorithm and by using graphics that portray its data structure features.

5.1.2 Algorithm Game Attributes

Thiagarajan [142] suggests five important attributes for instructional games to be effective and productive in transferring information:

1. Conflict: players may compete or cooperate with each other to overcome the game obstacles and win.

2. Control: the rules of how the game is played differ in difficulty and inflexibility from game to another.

3. Closure: reveals how the game ends; efficient games must apply several conditions for closure, allowing diverse groups of players to win.

4. Contrivance: elements of the game that induce or reward players to increase their degree of enjoyment and playfulness.

5. Competency: assist players to develop their skills in many areas, can be used to improve players knowledge or problem-solving skills.

Malone theory [143] describes four game properties that increase the motivation of the players:

1. Challenge: includes granting the user uncertain outcomes, obvious and personally meaningful goals, and varying and appropriate difficulty levels.

2. Fantasy: includes extrinsic fantasy, in which the user's actions determine what happens in the game, and intrinsic fantasy, in which the fantasy provides feedback to the user as well.

3. Curiosity: yields from environments that are neither too complicated nor simple, with appropriate graphics, music, and animation. Surprise and increasingly complex tasks also encourage curiosity.

4. Control: giving the player full control on the game and some of its features.

Considering Malone and Thiagarajan game attributes, the following common attributes and features for algorithm games can be defined:

1. The algorithm game can be created either by designing a totally new game or by modifying the game-play of an existing game to simulate the algorithm steps. For example, I have modified the Pong game to visualize the binary search algorithm by making one player use a ball and a paddle to hit a set of boxes instead of two players hitting the ball against each other (Section 6.1).

2. The algorithm game must be simple, not complicated, and focused on simulating the algorithm, so students do not lose concentration and become distracted. However, it must include challenging tasks and goals to enhance the self-esteem of the student.

3. The algorithm game should challenge the player by setting clear goals with appropriate difficulty levels and by giving clear and encouraging feedback. For example, all algorithm game prototypes that have been developed in Chapter 6 have several levels with increasing difficulties. In particular, if the data structure in the game has N elements at first level, it will have (Level Number * N) elements in the succeeding levels. However, the level time is also increased, using a similar formula (Current Level Time= Level Number * First Level Time), to create appropriate difficulty levels for the game.

4. The information in the algorithm game should be complex and unknown to increase player curiosity.

5. The game must give the most control to players by providing many options to customize it and increase player imagination and fantasy.

6. The algorithm game graphics must depict the features of the data structure of the visualized algorithm. For example, if the data structure is an array, a deck of cards can be used to visualize it, since the cards have values and can be arranged sequentially to simulate the array elements; for the tree data structure, an actual tree with leaves that represent the numbers can be used.

7. The game-play of the algorithm game must simulate the behavior of the algorithm that game is visualizing. For example, the bubble sort game (Section 6.2) does not allow the player

to uncover more than two cards in the same time or swap non-adjacent cards. In the binary search game (Section 6.1) if the player hits a non-middle block, he loses one point.

8. To simplify the student assessment, the algorithm game must keep a record for each player's progress in the game. Therefore, each algorithm game prototype that has been described in Chapter 6 includes a Player Report screen, which displays the time, date, score, and the playing results (won/lost) of the player entered name.

9. To support competition, algorithm game must allow learners to compare their performance with each other. Therefore, each algorithm game prototype that has been described in Chapter 6 includes a High Scores screen that displays the five highest scores achieved by all players.

5.1.3 Algorithm Games Genres

Game genre distinguishes one type of game-play from another. Genres are mainly focused on style of interaction and thus provide a good basis to find out whether the interaction type influences the general structure of a game. This section gives an overview of the most important game genres by presenting a high-level overview of the different game genres [135]. The most important game genres are summarized in Table 5.1. This table reveals surprisingly few really different game genres, and most of the games encountered will fit into one of these genres (sometimes in combination) [144]. Thiagarajan [142] specifies several game genres that can be used in designing instructional games such as card, verbal, solitaire, quiz, and board games. In particular, an algorithm game can be designed of any game genre depending on the type of the data structure and the algorithm it visualizes. For example, Chapter 6 gives some prototypes for algorithm games that have been designed as board games, such as bubble sort game and insertion sort game in addition to algorithm games that have been designed as ball and paddle games, such as binary search game.

5.2 Computer Game Design

A computer game features some kind of world, objects and characters in this world with different kind of properties and behaviors, and rules that make up the game and control how these objects

Table 5.1: Computer Games Genres

Game Genre	Description	Examples
Ball and Paddle	A player controls a paddle object that moves back and forth on a single axis.	Pong game.
Binball Games	A player makes a ball hit various parts of a play field to gather up points.	
Fighting Games	Players fight with each other or with a computer controlled characters.	
Maze Games	Players must navigate through a playing field, which is entirely a maze.	
Shooter Games	The focus is primarily on combat involving projectile weapons, such as guns and missiles.	First-person shooter: the screen is the player's own vision. Third-person shooter: avatars that are fully visible to the player.
Simulation Games	Vehicle simulation: Sports simulation: simulates sports Card simulation: simulates game cards Construction and management simulation: city-building, business simulation, etc. Life simulation: biological simulation, pet-raising, etc.	Flight, racing, space flight, etc. football, basketball, etc. solitaire game.
Adventure Games	Players solve various puzzles by interacting with people or the environment.	
Role-playing Game	The game-play is centered around one or more avatars, with characteristics that evolve over the course of the game, and take the place of the gamer's own skill in determining the outcome of the actions taken by the player.	
Strategy Games	The game-play requires/ careful and skillful thinking and planning in order to achieve victory.	Real-time strategy, War games.
Board Games	Adaptation of existing board games or games which are similar to board games in their design and play even if they did not previously exist as board games.	Backgammon, Othello, Checkers, Chess.

interact with each other.

5.2.1 Educational Games Design

At the end of her research on educational games effects, and after reviewing many theories on designing such games, Chamberlin [19] gives ten suggestions to be considered when designing games for learning purposes:

1. Interface design is a key consideration: game interfaces should provide user-guided learning, help when required, and navigation assistance through the game.

2. Games should incorporate feedback throughout play: feedback support player advancement during the game. The use of scores and rewards can support player need for excitement.

3. Environments and characters are important: use of fancy graphics, professionally produced animation, and sounds are important to users, but games do not have to overdo it to be appropriate.

4. Games should engage users with activity: player involvement and decision-making within the game should be increased.

5. Build challenge into the game-play: provide rising complexity levels, increased problems to solve, or competition, without exceeding the players' abilities.

6. Offer users control throughout activities: contribute not only to user engagement, but also permit players to change the play environment to meet their needs.

7. Build on user's familiarity of other games, characters, and content: while familiarity increases player's satisfaction, also novel and unusual material are encouraged.

8. Recognize the importance of variety: players should be granted different types of opportunities for utilizing their different skills and interests.

9. Repeat educational information: instructive information should be given in various places with several learning approaches.

10. Involve users in design process: this is achieved by regular testing, infrequent discussions, or by benefiting from players as design associates.

These suggestions are taken into account when designing our algorithm games to teach algorithms, as shown in the following sections.

5.2.2 Game Elements

When it comes to game design, two types of elements can be identified: external and internal elements. External elements are related to the design of the game appearance. According to a research about the game industry conducted by Microsoft [122], there are several important elements that affect the appearance of games: meshes and materials, audio, video, animation, user interface, and worlds/levels. Internal elements are related to the internal or core design and implementation of the game. Current research in game design [145] distinguishes the following four elements:

1. The interaction with the player determining how to control the game and what feedback is provided.

2. The objects that make up the game world with specific behaviors and properties.

3. The rules that govern the core mechanics of the game and determine what the player can and cannot do.

4. A storyline is not present in all games, but in some genres it takes a prominent place, such as action, adventure, and RPG.

5.2.3 Game Design Document

Prepared by game designers, the design document contains information about the core elements that make a game. One of the important parts of any design document is the game mechanics, which describes the game-play of the game. It describes how the game is played, the flow of the game, and provides detailed information on the movement of every object in the game. A second section of the game design document gives details on the story of the game, or the levels the user has to

go through. Another section of the design document is the setting of the game, which is supported by the story, the art work (graphics), the video, and the sound used throughout the game. All game design documents contain some details on how the game should feel, what the overall mood is, and at least some impression sketches or concepts art of the game. Besides defining the game mechanics, the story and setting, the interaction with the user must be determined in the game design document. How the user controls the game world, what are the GUI elements within the game, like Head-Up-Displays (HUD), menus, and help screens must all be specified. These specifications mostly define how the user controls the system surrounding the game: saving, loading, (re)starting the game, changing some settings, and so on [131].

5.2.4 Game General Design Elements

In a crash course about game design, Packard [146] defines eight general game design elements that can be used to describe a game prototype:

1. Game Idea: a game must have an idea on which it is based.

2. Game Goal: a game must have a clearly defined goal. This goal must be expressed in terms of the effect that it will have on the player.

3. Game Topic: the topic is the means of expressing the goal, the environment in which the game will be played. It is the concrete collection of conditions and events through which the abstract goal will be communicated.

4. Game Start: what are the game startup screens? What are the startup parameters? Where are the characters? What messages, if any, are on screen, and where? Is there introductory music?

5. Game Levels: how does the difficulty increase? How does a level end? How does the player know what level he is on?

6. Game Milestone Events: this refers to points of the game at which the player is rewarded or penalized. Milestone Events are gauges to let players know they are on the right (or wrong)

track, and will encourage (or discourage) them to keep going.

7. Game End: what happens when the player loses? What happens when the player wins? What happens when the player gets a high score? Where does the player go when the game is over? How does the player start a new game?

8. Game Exit: what does player see when he decides to exit the game?

5.2.5 Game Appearance Design

Game Graphics Design

Game graphics are everything that contributes to the visual appearance of the game, such as fonts, (2D) Sprites, and (3D) Models. Some games do not require any fonts at all; instead, they use word textures for whatever messages they need, or use the normal bios print routine. The graphics that build the game world can be classified into three types:

1. Background objects that do not move and are not drawn on top of something else during the game. Each object is defined by name, description, size, and position.

2. Foreground objects that don't move and are drawn on top of everything else, such as Get Ready Image, Game Over Image, and animated logo. Each object is defined by name, description, size, and position on the screen.

3. Character objects that move and/or animate on the screen. They can be 2D Sprites or 3D Models. Each object is defined by name, description, color, size, texture, personality traits, and movement options. Game charterers may be divided into:

    - A Player Character (PC): a fictional character who is controlled by the player.
    - A Non-Player Character (NPC): a character whose actions are not under the player's control, such as bystanders, competitors, bosses, or may exist to aid the player's progress in the game.

Game Sounds Design

An event is anything that causes playing one of the game digital samples, such as the beginning of the game, losing a life, or getting a high score. The game goal-related events, such as GameOver and HighScore events, are the milestone events, while any events that are not related to the game goals are the game-related events. There are two types of game sounds:

1. Musical Sounds are played at milestone events, and are not affected by any other game-related events. Based on the type of milestone events that triggers them, they fall into three categories:

   (a) Theme Music: five to ten or more seconds setting the tone of the game. Theme Music is played during major milestone events, such as TitleSequence, StartNewGame, NewLevel, and YouDie.

   (b) Background Music: begins at milestone events, like StartGame, or Intermission, and continues to play throughout the event.

   (c) Musical Tag: very short samples, usually two seconds or less, occur at minor milestone events, like HitHighScore.

2. Sound Effects (SFX) are related to game events where something happens in the game that is not really milestone-related, but is important enough that a sound effect needs to play for it, such as Hitting a Ball.

Game Screens Design

A game screen is a collection of visual and audio components that describe the state of the game at any one time during the game life cycle. Exemplars of game screens are:

- Title Screen: players see this when they first start the game.
- Self-running Demo: a demonstration of the game that is played without player interference.
- About This Game Screen: gives information about the game.

- Game Selection Screen (Main Menu): displays available options to the player on game startup, how the player gets there, and how he gets out?
- Game Play Screen: displays the common game parameters, such as scores, number of levels, and other information to the player.
- Game Level Screen: displays one level of the game.

The game screen design is a plan for how and where things will be placed on the screen. There are usually many things that need to be placed on the screen, such as the player's score, number of lives left, game logo, level number, etc. Each game screen design must include the following items:

- Screen Title: name of the screen.
- Screen Description: what information needs to be on it?
- Screen Layout: what goes where, and how big it is in pixels?
- Screen Exit: what happens when this screen is exited?
- Player Controls: whatever the player can do on this screen, and how he does it. How he calls up any menus, what he does to interact with the game.
- Menus, Choices, and Functions: if this screen has special menu options, what they are, how the player changes those options, and how the game will let him know of those changes.
- Feedback Systems: what player information is displayed on the screen, where it is, how the player will access it.

Game Assets

The game assets are the game files that make up the game content. These files include:

1. Texture Files: define the 2D sprites of the game.
2. Model Files: define the 3D models of game.

3. Effect Files: specify the appearance of the game models.

4. Font Files: define the properties of the fonts that are used in the game.

5. Sound Files: determine the sound clips that are used in the game.

5.2.6 Game Mechanics Design

Game Mechanics describe the flow of the game, the movement and drawing order of every object in the game, and needed procedures to achieve game goals. The heart of the game mechanics is the game loop. The game loop continually updates the state of the game based on user input, in-game conditions, and any other applicable condition and renders it. This involves drawing to the screen, playing appropriate audio, rumbling a controller, and providing any other form of output to the user. Besides the game loop, the game mechanics include the game user interface design, the game screens structure, and the game-play.

Game Properties

What are the game name, number of levels, number of the player lives, level time? How are the current level number, remaining lives, player score, and level timer calculated?

User Interface Design

A user interface is the mechanism a game uses to get information to and from the player; generally, it consists of two parts:

1. User Controls to be used by the player to affect the game, including character movement or actions, pull down menu choices, and options screen controls.

2. Feedback System that conveys information to the player, such as his Score Display, Number of Lives, Level Number, Sound Effects (SFX), and Visual Effects (FX).

The Game Input/Output is the player's tactile contact with the game. An excellent game offers the player a large number of meaningful options in addition to the choice of several input devices, such

as keyboard, joystick, gamepad, or mouse. The output structure includes graphics that convey the game information while supporting the fantasy of the game in addition to sounds that tell the player what is going on in the game.

Game Screens Structure

A Finite State Machine (FSM) is a software machine that has a limited number of states that it can be transitioned in and out of. Like the FSM, the game transitions between many screens, at the start of the game the Title screen is displayed followed by the Main Menu screen. Then, depending on the player choice, the Play or Options screen is displayed. When the player wins or loses the game, the related screen is displayed. Therefore, a game can be represented as a finite state machine where each game screen corresponds to the state the game is in. A stack can be used to store the different game screens to simplify the transition between them. Using the stack Push() operation, a game screen like Pause can be easily displayed on top of other screens and a screen can be removed by a Pop() operation [147].

Game-Play

The game-play explains how the game is played, what the controls are, what the game objects allowed movements are, and how the player is going to achieve the game goals.

5.3 Algorithm Game Design

Based on game design document, game general design elements, game appearance design, and game mechanics design sections, this section specifies an Algorithm Game Prototype that serves as the design document for an algorithm game by describing its key design elements. Each Algorithm Game Prototype has three sections: algorithm game general design elements, algorithm game appearance design, and algorithm game mechanics design.

5.3.1 Algorithm Game General Design Elements

The algorithm game design elements that describe an algorithm game prototype are as follows:

1. Game Idea: each algorithm game idea visualizes an algorithm steps and its data structure.

2. Game Goal: involves learning the visualized algorithm.

3. Game Topic: shows the algorithm and its data structure features.

4. Game Start: each algorithm game starts by displaying one game level that includes its graphic items. The level HUD default attributes are Level <= Maximum Level Number, Score= 0, Timer= Maximum Level Time, and Lives= Maximum Player Lives.

5. Game Levels: describes how the difficulty increases, how a level ends. Each completed level must achieve a learning sub-goal. Moreover, each level has a specific playing time. Each algorithm game has several levels.

6. Game Milestone Events: many reward and penalized points must be placed for the game. The algorithm game milestone events are the starting of a new level and losing of one life.

7. Game End: the algorithm game ends when the player either loses all his lives or completes all game levels successfully. If the player loses the game, Lost screen and Main Menu screen are displayed, respectively. Otherwise, if the player wins the game, Won screen, High Scores (if the Player Score is high) screen, and Main Menu screen are displayed respectively.

8. Game Exit: the algorithm game can be exited from the Main Menu screen Exit button, the X-button of the gamepad, the Escape button of the keyboard, or from the close (or X) button on the game window. Before closing the game window, the Credits screen is displayed.

5.3.2 Algorithm Game Appearance Design

Graphics Design

The Algorithm Game Graphic Items define the game entities or objects that make the game world. Each graphic item is either a 2D Sprite or a 3D Model with attributes, such as size, position, color, texture, and name in addition to behaviors like render, move, and update. Examples of graphic items of a typical algorithm game are as follows (Figure 5.1):

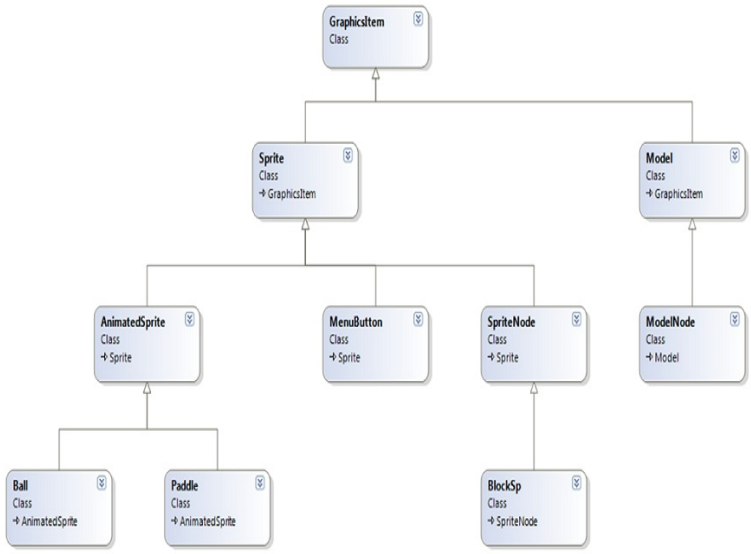

Figure 5.1: Graphic Items Class Architecture

- Node: animated item used to visualize one node of the algorithm data structure, such as Card, Box, and Domino.
- Collections: a set of similar nodes that have been organized according to specific rules. A collection is used to visualize an algorithm data structure, such as a Deck of Cards, Map of Cities, and Pack of Dominoes.
- Playing Tools: animated items used to play the game, such as Ball, Paddle, and Shooter.
- Buttons: non animated graphic objects used in the screens design.

Game Assets

Texture, fonts, models, effects, and sound files.

5.3.3 Algorithm Game Mechanics Design

Input Design

Input design handles the game input from the mouse, keyboard, and gamepad. For each player input event in the game, the required feedback must be implemented.

Game Properties

Some of the game parameters are initialized at the start of the game, the default parameters are Game Name, Maximum Levels Number, Maximum Player Lives, and Maximum Level Time. Other game attributes are calculated and displayed to show the game current state, the default parameters are Player Remaining Lives, Current Level Timer, Player Score, and Current Level Number.

Game-play

Game-play is responsible for implementing the playing rules of the game according to the visualized algorithm behavior. The general game-play of an algorithm game is as follows:

1. While playing, if (Current Level Timer == 0), the current level ends.
2. If current level is lost, the Player Remaining Lives is decreased by one.
3. If (Player Remaining Live > 0), the current level is repeated;
    (a) else the game ends and the player loses the game.
4. If current level is completed successfully, the Current Level Number is increased by one.
5. If (Current Level Number < Maximum Levels Number), a new level is displayed;
    (a) else the player wins the game and the game ends.

Screens Design

Each algorithm game has a set of default game screens that are explained in the following:

1. Title Screen: displays the game title, logo, and name.
2. Player Name Screen: displays an on screen keyboard at the game start to input the player name.
3. Main Menu Screen: displays the game main menu options and handles the player choices.

4. Start Level Screen: displays the start of one game level for the player including the level number.

5. Play Screen: displays the game for the player to play, including a Head-Up-Display (HUD) that shows the game properties, such as Level Number, Remaining Lives, Game Name, Player Score, and Level Timer.

6. Help Screen: displays instructions of how the game is played for the player.

7. Pause Screen: allows the player to stop and resume the game at any time.

8. Exit Screen: allows the player to stop and end the game at any time.

9. Won Level Screen: displays "Level Completed" message when the player wins one level and asks the player to continue the game; then, displays the next level for the player.

10. Lost Level Screen: displays the Remaining Player Lives when the player loses one level and asks the player to continue the game; then, repeats the same level for the player.

11. Won Game Screen: displays a won message when the player wins the game.

12. Lost Game Screen: displays a lost message when the player loses the game.

13. Options Screen: displays the game options for the player.

14. Game Demo Screen: displays the self- running demo of the game.

15. High Scores Screen: shows the highest five scores achieved during the game by all players.

16. Player Report Screen: displays a progress report for the player for the last eight times the player played the game, including Game Result (Lost, Won), Score, Level Number, Play Date, and Play Time.

17. Credits Screen: shows the game developers' names.

Chapter 6: Algorithm Games Prototypes

This chapter describes the design of several algorithm games prototypes. At the beginning, each section describes the algorithm behavior, which has been simulated by the described prototype. Then, the prototype components, such as the general game design, game appearance design, and game mechanics design, are detailed.

6.1 Binary Search Game

The binary search game is based on the original Pong game (Figure 6.1), where the player hits a ball character who moves around, each one containing enemies (robots) which can be shot by the main character. Binary Search, however, extends the original Pong game to simulate the binary search algorithm by adding an array of blocks, background music, and a more interesting rule set, as it will be presented throughout this section.

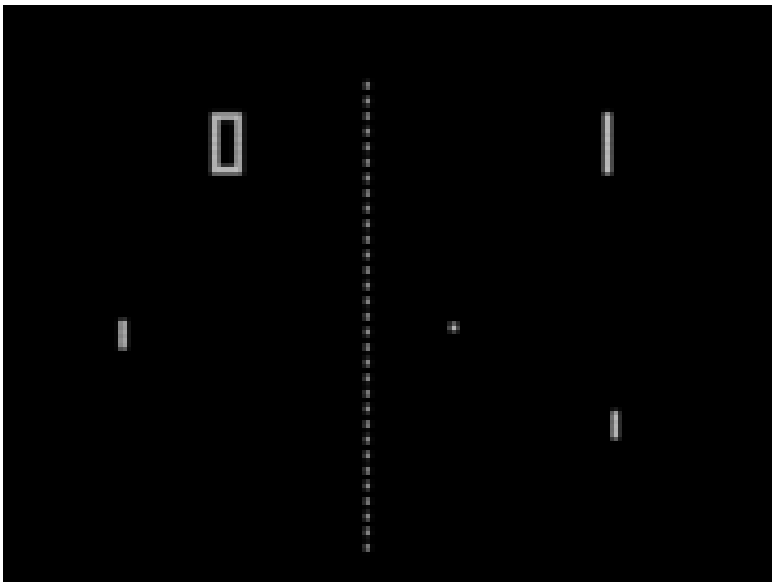

Figure 6.1: Pong Game

6.1.1 Pong Game

Pong is a two-dimensional sports game which simulates table tennis. The player controls an in-game paddle by moving it vertically across the left side of the screen, and can compete against either a computer controlled opponent or another player controlling a second paddle on the opposing side. Players use the paddles to hit a ball back and forth. The aim is for a player to earn more points than the opponent; points are earned when one fails to return the ball to the other side [148].

6.1.2 Binary Search Algorithm

The binary search algorithm (Algorithm 6.1) finds the index of a specific value (target) in a sequential list of sorted elements (array) by selecting the middle element (median) of the array and comparing it with the target:

1. If (median > target), the index of the median-1 becomes the new upper bound of the list;
2. else if (median < target), the index of the median+1 becomes the new lower bound;
3. else if (median = target), return the index of the median.

The algorithm pursues this strategy iteratively for the new list bounded by the middle element. It reduces the search span by a factor of two each time, and soon finds the target value or else determines that it is not in the list.

6.1.3 Binary Search Game Prototype

First–General Game Design Elements

1. Game Idea: hitting an array of blocks with a ball using a paddle.
2. Game Topic: the game is a modification of Pong game.
3. Game Goal: simulating the binary search algorithm.
4. Game Start: the game starts by displaying one game level that includes a set of blocks with their values hidden, Ball, and Paddle. Besides the default attributes of the algorithm game HUD, the level HUD includes Search Number=random value and Search Index=’ ’.

Algorithm 6.1 Binary Search

```
BinarySearch(Array[0..Size-1], Value, Low, High)
Low =0
High =Size - 1
while (Low <=High) do
  Mid = (Low + High)/2
  if (Array(Mid) >Value) then
    High =Mid - 1
  else if (Array(Mid) <Value) then
    Low =Mid + 1
  else
    return Mid{Found}
  end if
end while
return -1 {Not found}
```

5. Game Levels: the game has several levels; at each new level, the number of blocks is increased to make the game more challenging.

6. Game End: default settings of the algorithm game.

7. Game Milestone Events: default settings of the algorithm game.

8. Game Exit: default settings of the algorithm game.

Second–Game Appearance Design

- Game Graphic Items: 2D sprites, each with name, texture, size, and position.
  - Set of Blocks: each block has a value to represent one element of the array.
  - Ball: used to hit one block of the set to reveal its value.
  - Paddle: used to move the ball toward the blocks.
- Game Assets:
  - Textures: Ball, Paddle, and Block.
  - Sounds: LostBall, HitBall, TimeEnds, LostLife, LostGame, and WonGame.
  - Fonts: Arial.

Third–Game Mechanics Design

- Game Properties: the default properties are initialized as follows the Maximum Level Number=3, Maximum Player Lives=3, and Game Name=Binary Search. Besides the default, the calculated properties include the Search Number.
- User Interface–Input Design: supporting keyboard, mouse, and Xbox gamepad.
- Game Screens Design: the game screens are all default algorithm game screens.
- Game-Play: in addition to the general playing rules that have been included in each algorithm game, the specific playing rules that simulate the algorithm steps are as follows (Figures 6.2-6.7):

    1. The player hits one block of the array with the Ball, using the Paddle.
    2. If the hit block is in the middle:
        (a) The Player Score is increased by five points.
        (b) If (Search Number > hit block value), the player plays on the right section of the array;
            i. else if (Search Number < hit block value), the player plays on the left section of the array;
            ii. else if (Search Number = hit block value) or the Search Number is not found, the level ends.
    3. Else if the hit block is not in the middle, the Player Score is decreased by one point.
    4. If the level time ends without hitting all middle blocks in the set, the player loses the level.
    5. At each new level, the number of the blocks in the set, Level Timer, and Level Number are increased.

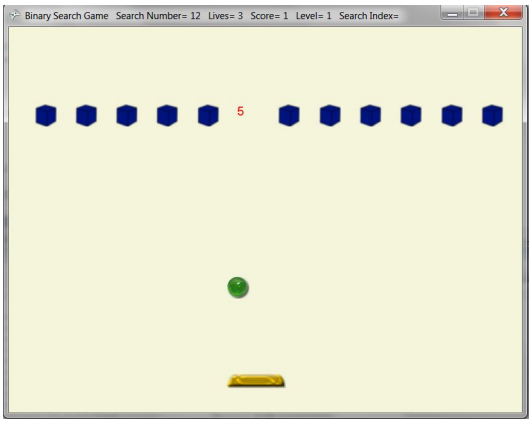

Figure 6.2: Binary Search–Mid=5<12, Right

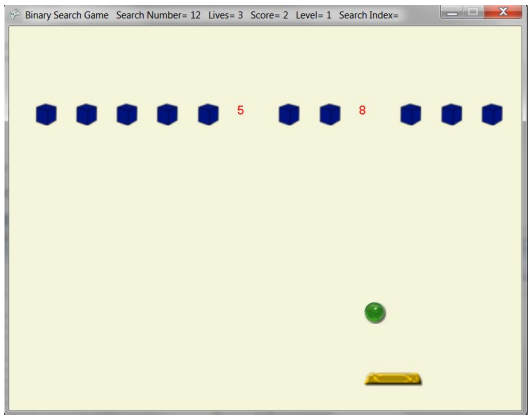

Figure 6.3: Binary Search–Mid=8<12, Right

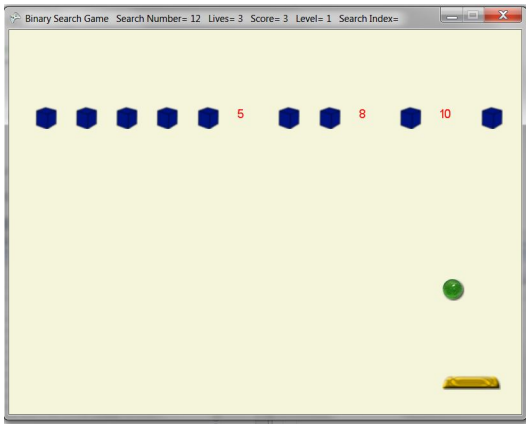

Figure 6.4: Binary Search–Mid=10<12, Right

Figure 6.5: Binary Search–Mid=11 <12

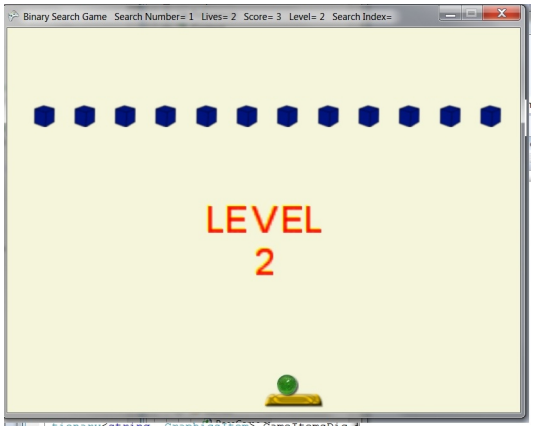

Figure 6.6: Binary Search–Level 2

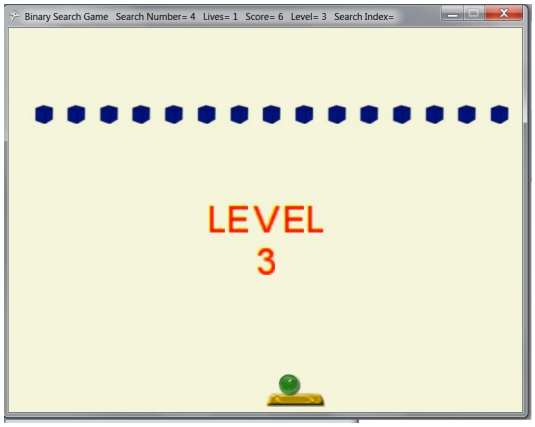

Figure 6.7: Binary Search–Level 3

6.2 Bubble Sort Game

6.2.1 Bubble Sort Algorithm

The bubble sort algorithm sorts an array of N elements by looking at adjacent elements in the array and putting them in order. It looks at the first two elements, swaps them if necessary; then it looks at the second and third elements, swapping if necessary; then it looks at the third and the fourth and so on. After one pass through the array, the largest item will be at the end of the array. Similarly, for another complete pass through the array, the second largest item will be in position. If N passes are made, all of the array will be sorted.

6.2.2 Game Design Prototype

First–General Game Design Elements

1. Game Idea: sorting a deck of cards in a fixed time.
2. Game Topic: card board game, such as solitaire.
3. Game Goal: simulating bubble sort algorithm.
4. Game Start: the game starts by displaying one game level that includes a deck of unsorted, covered cards. The level HUD includes all default attributes of the algorithm game HUD.
5. Game Levels: the game has several levels; at each new level, the number of cards is increased to make the game more challenging. However, the timer value also increased to create appropriate difficulty levels.
6. Game End: default settings of the algorithm game.
7. Game Milestone Events: default settings of the algorithm game.
8. Game Exit: default settings of the algorithm game.

Second–Game Appearance Design

- Game Graphic Items:
  - Deck of cards, each card having a value.
  - Swap button.
- Game Assets:
  - Textures: 52 cards, 13 for each card suit, Swap button, and card cover textures.
  - Sounds: TimeEnds, LostLife, LostGame, and WonGame.

Third–Game Mechanics Design

- Game Properties: all default properties, which are initialized as follows: Maximum Level Number=3, Maximum Player Lives=3, Maximum Level Time=1 minute, and Game Name=Bubble Sort in addition to default calculated properties.
- User Interface–Input Design: supporting keyboard, mouse, and Xbox gamepad.
- Game Screens Design: the game screens are all default algorithm game screens.
- Game-Play: in addition to the general playing rules that have been included in each algorithm game, the specific playing rules that simulate the algorithm steps is as follows (Figures 6.8-6.13):
  1. The player uncovers two adjacent cards to see their values.
  2. The player must compare the cards and swap them, if they are not sorted.
  3. If the player makes a correct swap (cards were not in order before swapping), the Player Score is increased by five points; otherwise, it is decreased by one.
  4. If the level time ends without sorting all cards in the deck, the player loses the level.
  5. At each new level, the number of the cards in the deck, Level Timer, and Level Number are increased.

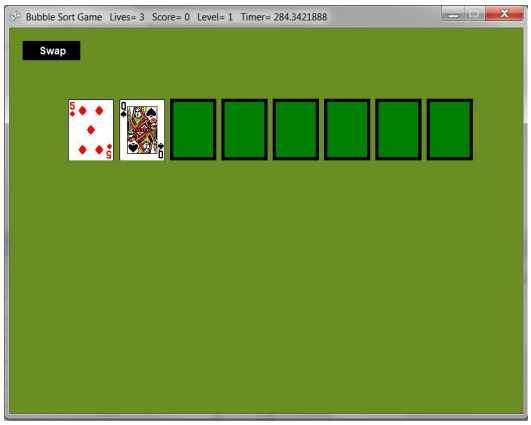

Figure 6.8: Bubble Sort–Sort [Queen]

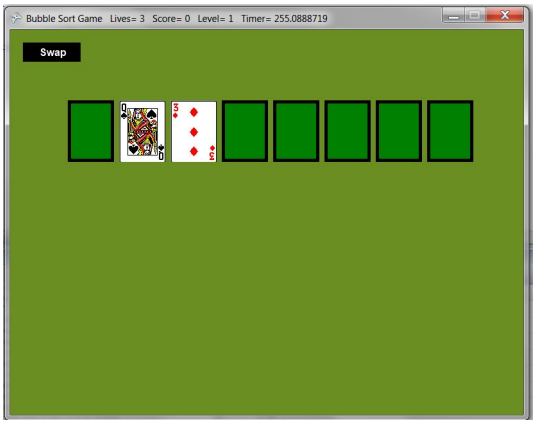

Figure 6.9: Bubble Sort–[Queen] Swap 1

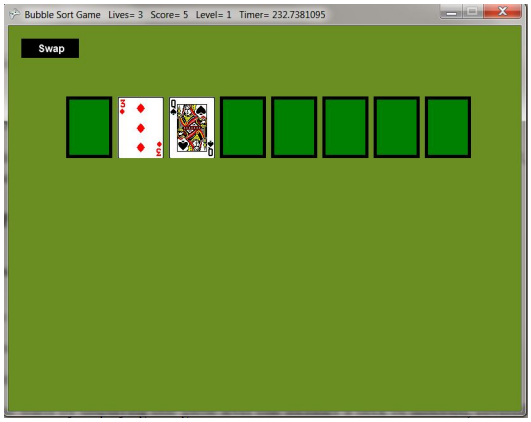

Figure 6.10: Bubble Sort–[Queen] Swap 2

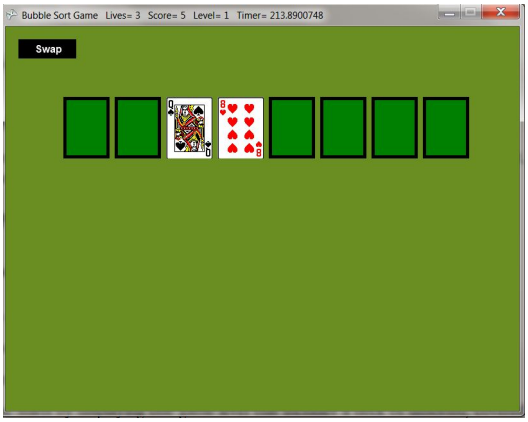

Figure 6.11: Bubble Sort–[Queen] Swap 3

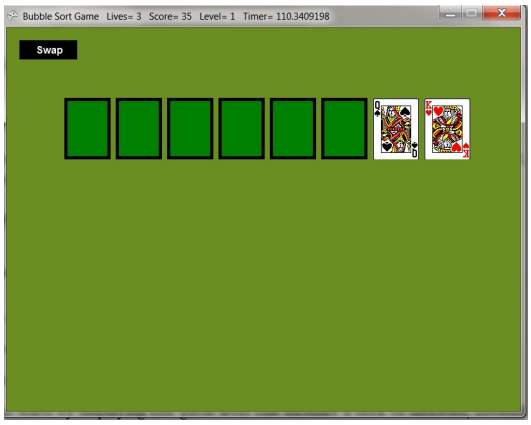

Figure 6.12: Bubble Sort–[Queen] In Place

Figure 6.13: Bubble Sort–Level Completed

6.3 Selection Sort Game

6.3.1 Selection Sort Algorithm

The selection sort algorithm sorts an array of numbers as follows: it finds the minimum value in the list, swaps it with the value in the first position, and repeats for the remainder of the list, excluding the swapped elements at the beginning.

6.3.2 Game Design Prototype

First–General Game Design Elements

1. Game Idea: sorting a deck of cards in a fixed time according to the algorithm rules.
2. Game Topic: card board game, such as solitaire.
3. Game Goal: simulating the selection sort algorithm.
4. Game Start: the game starts by displaying one game level that includes a deck of unsorted, covered cards. The level HUD includes all default attributes of the algorithm game HUD.
5. Game Levels: the game has several levels; at each new level, the number of cards is increased to make the game more challenging. However, the timer value also increased to create appropriate difficulty levels.
6. Game End: default settings of the algorithm game.
7. Game Milestone Events: default settings of the algorithm game.
8. Game Exit: default settings of the algorithm game.

Second–Game Appearance Design

- Game Graphic Items:
  - Deck of cards, each having a value.
  - Swap button.

- Game Assets:
  - Textures: 52 cards, 13 for each card suit, Swap button, and card cover textures.
  - Sounds: TimeEnds, LostLife, LostGame, and WonGame.
  - Fonts: Arial.

Third–Game Mechanics Design

- Game Properties: all default properties, which are initialized as follows: Maximum Level Number=3, Maximum Player Lives=3, Maximum Level Time=1 minute, and Game Name=Selection Sort in addition to default calculated properties.
- User Interface–Input Design: supporting keyboard, mouse, and Xbox gamepad.
- Game Screens Design: the game screens are all default algorithm game screens (Figures 6.14-6.19).
- Game-Play: in addition to the general playing rules that have been included in each algorithm game, the specific playing rules that simulate the algorithm steps is as follows (Figures 6.20-6.31):
  1. The player chooses the smallest card value and inserts it in its correct sorting place on the left.
  2. If the player inserts the selected card in its correct place, the Player Score is increased by five points; otherwise, it is decreased by one.
  3. If the level time ends without sorting all the cards in the deck, the player loses the level.
  4. At each new level, the number of the cards in the deck, Level Timer, and Level Number are increased.

Figure 6.14: Selection Sort–Title Screen

Figure 6.15: Selection Sort–Player Name

Figure 6.16: Selection Sort–Main Menu

Figure 6.17: Selection Sort–High Scores

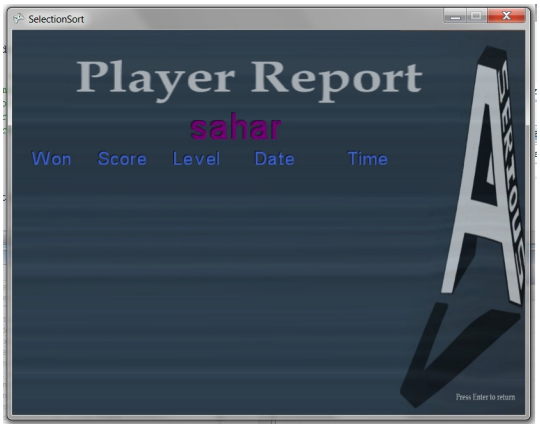

Figure 6.18: Selection Sort–Player Report

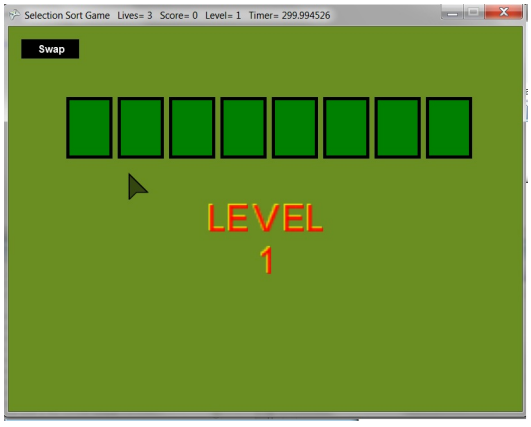

Figure 6.19: Selection Sort–Level 1

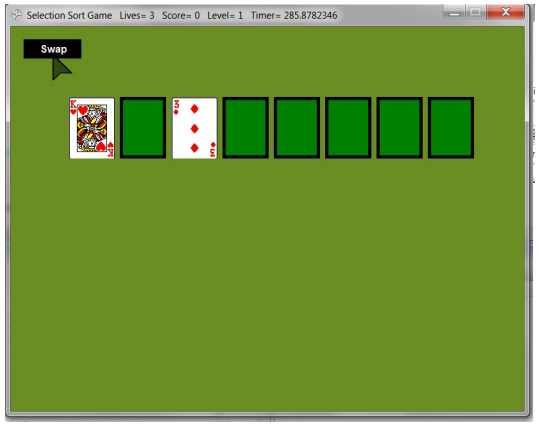

Figure 6.20: Selection Sort–Sort 1st Min[3]

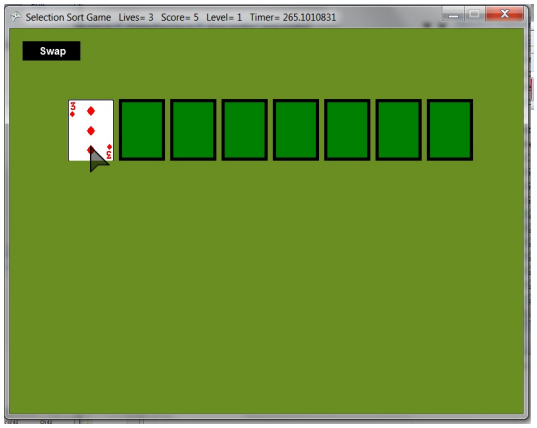

Figure 6.21: Selection Sort–Card [3] In Place

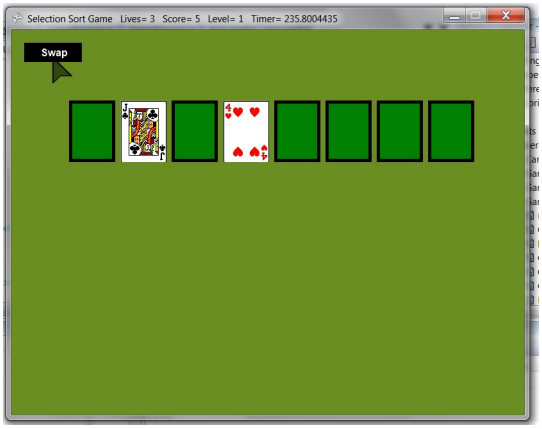

Figure 6.22: Selection Sort–Sort 2nd Min[4]

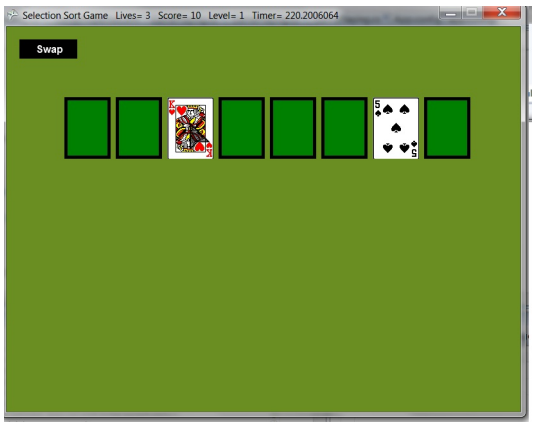

Figure 6.23: Selection Sort–Sort 3rd Min[5]

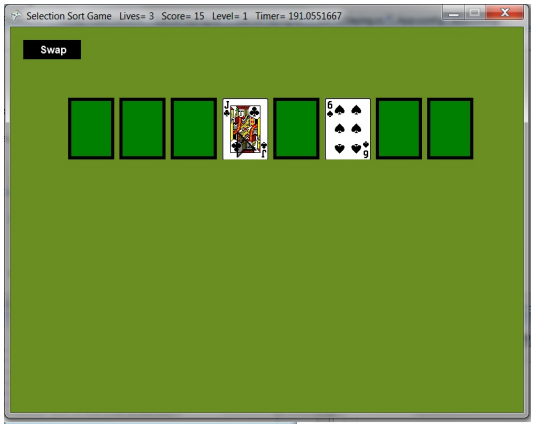

Figure 6.24: Selection Sort–Sort 4th Min[6]

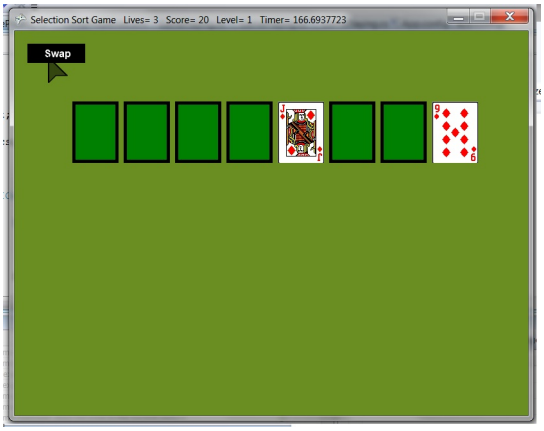

Figure 6.25: Selection Sort–Sort 5th Min[9]

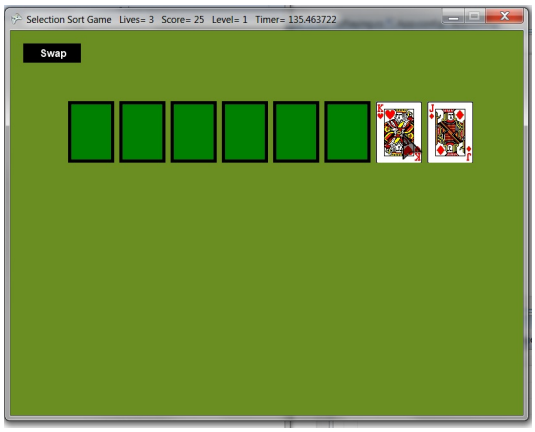

Figure 6.26: Selection Sort–Sort Last Min[J]

Figure 6.27: Selection Sort–Sorted Cards

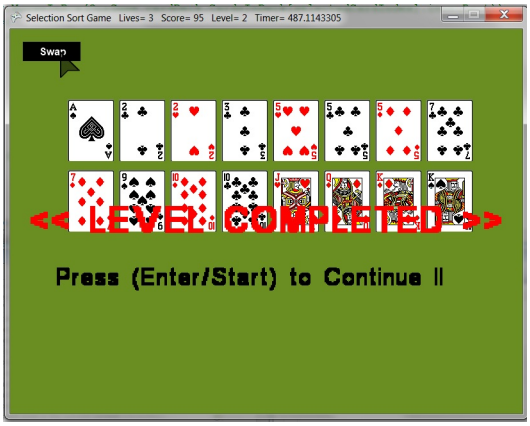

Figure 6.28: Selection Sort–Level Completed

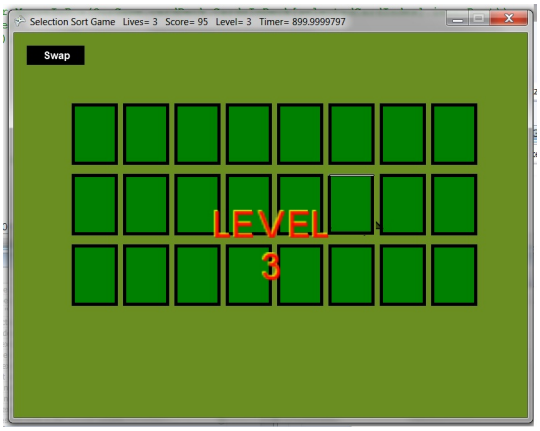

Figure 6.29: Selection Sort–Level 3

Figure 6.30: Selection Sort–Game Exit

Figure 6.31: Selection Sort–Game Pause

6.4 Insertion Sort Game

6.4.1 Insertion Sort Algorithm

The insertion sort algorithm sorts an array of numbers, as follows: it removes an element from the input data, inserts it into the correct position in the already sorted list until no input elements remain, and repeats for remainder of the list (excluding the elements in the already sorted list).

6.4.2 Game Design Prototype

First–General Game Design Elements

1. Game Idea: sorting a deck of cards in a fixed time.
2. Game Topic: card board game, such as solitaire.
3. Game Goal: simulating the insertion sort algorithm.
4. Game Start: the game starts by displaying one game level that includes a deck of unsorted, covered cards. The level HUD includes all default attributes of the algorithm game HUD.
5. Game Levels: at each new level, the number of cards is increased to make the game more challenging. However, the timer value also increased to create appropriate difficulty levels.
6. Game End: default settings of the algorithm game.
7. Game Milestone Events: default settings of the algorithm game.
8. Game Exit: default settings of the algorithm game.

Second–Game Appearance Design

- Game Graphic Items:
  - Deck of cards, each having a value.
  - Swap button.

- Game Assets:
  - Textures: 52 cards, 13 for each card suit, Swap button, and Card cover textures.
  - Sounds: TimeEnds, LostLife, LostGame, and WonGame.
  - Fonts: Arial.

Third–Game Mechanics Design

- Game Properties: all default properties, which are initialized as follows: Maximum Level Number=3, Maximum Player Lives=3, Maximum Level Time=1 minute, and Game Name=Insertion Sort in addition to default calculated properties.
- User Interface–Input Design: supporting keyboard, mouse, and Xbox gamepad.
- Game Screens Design: the game screens are all default algorithm game screens (Figures 6.32-6.37).
- Game-Play: in addition to the general playing rules that have been included in each algorithm game, the specific playing rules that simulate the algorithm steps is as follows (Figures 6.38-6.51):
  1. The player chooses one card at a time to be the key.
  2. The player uncovers the chosen card to see its value.
  3. The player must compare the card with all cards on the lift to insert it in its sorted place.
  4. If the player inserts a card in incorrect place, the Player Score is decreased by one point.
  5. If the player inserts a card in correct place, the Player Score is increased by five points.
  6. If the level time ends without sorting all the cards, the player loses the level.
  7. At each new level, the number of the cards in the deck, Level Timer, and Level Number are increased.

Figure 6.32: Insertion Sort–Title

Figure 6.33: Insertion Sort–Player Name

Figure 6.34: Insertion Sort–Main Menu

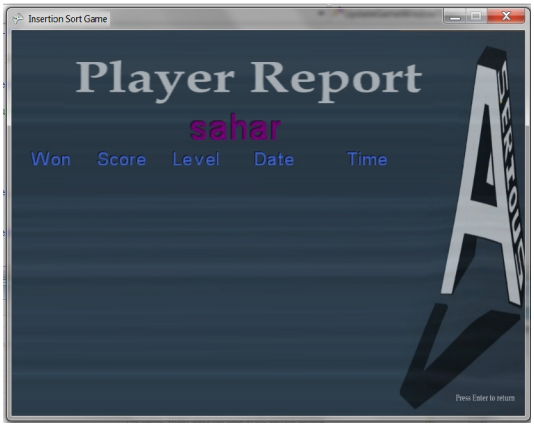

Figure 6.35: Insertion Sort–Player Report

Figure 6.36: Insertion Sort–High Scores

Figure 6.37: Insertion Sort–Game Pause

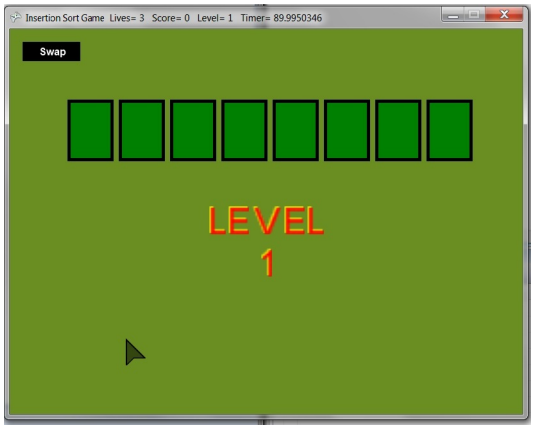

Figure 6.38: Insertion Sort–Level 1

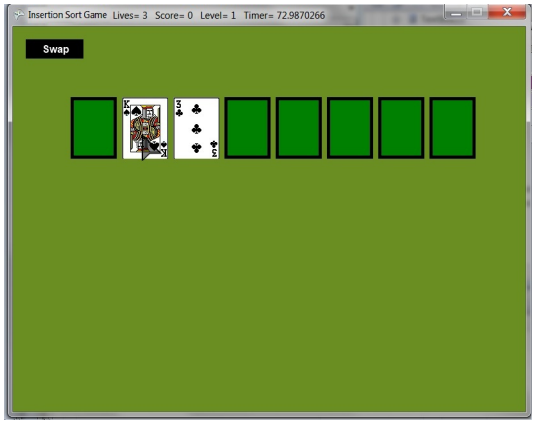

Figure 6.39: Insertion Sort–Insert Card[3]

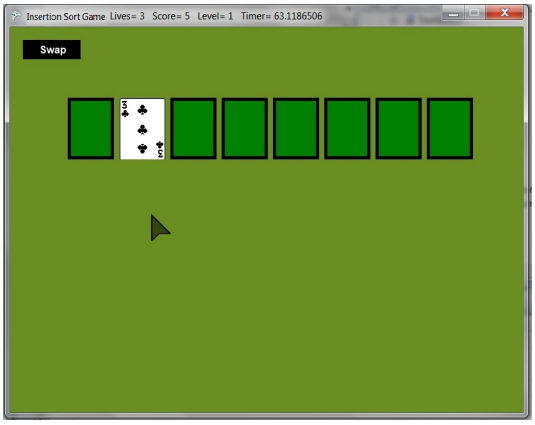

Figure 6.40: Insertion Sort–Card[3] Swap 1

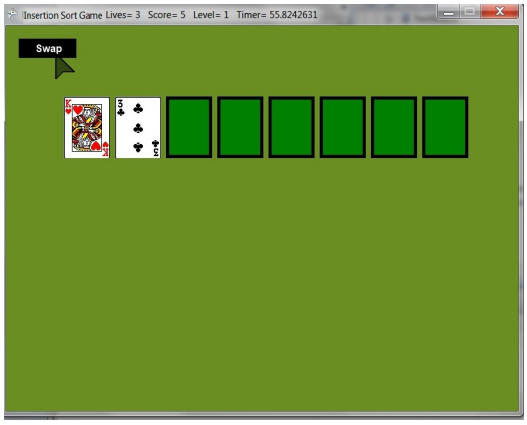

Figure 6.41: Insertion Sort–Card[3] Swap 2

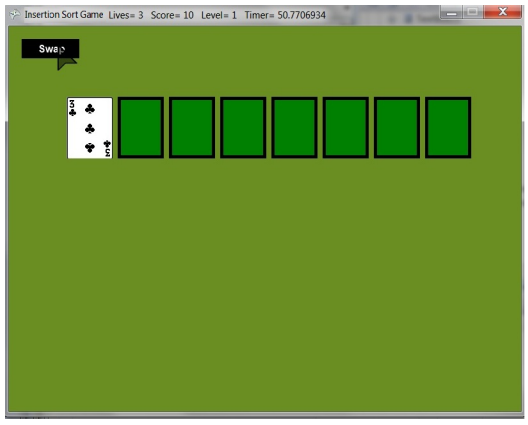

Figure 6.42: Insertion Sort–Card[3] In Place

Figure 6.43: Insertion Sort–Level Completed

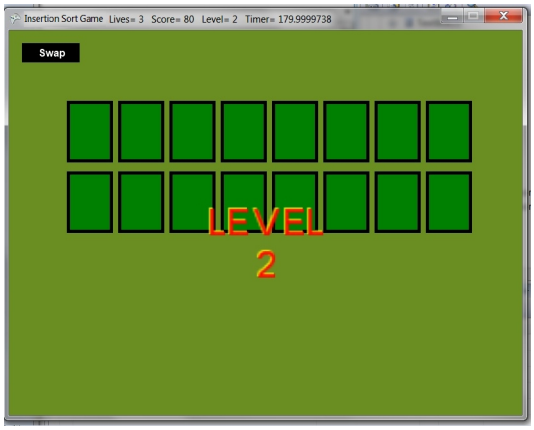

Figure 6.44: Insertion Sort–Level 2

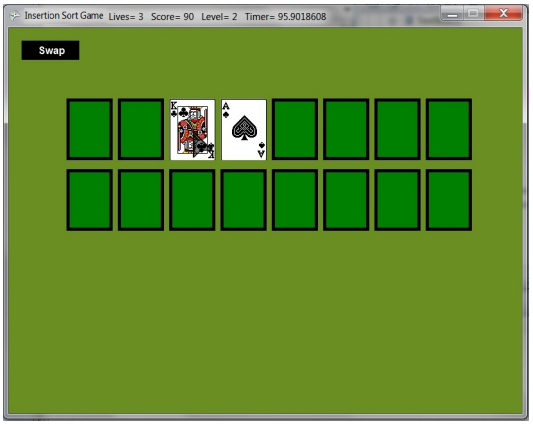

Figure 6.45: Insertion Sort–[Ace] Swap 1

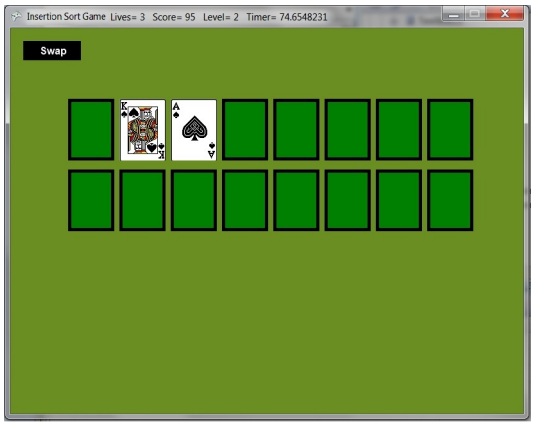

Figure 6.46: Insertion Sort–[Ace] Swap 2

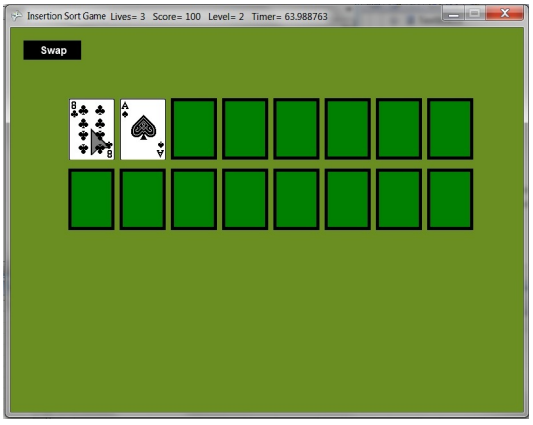

Figure 6.47: Insertion Sort–[Ace] Swap 3

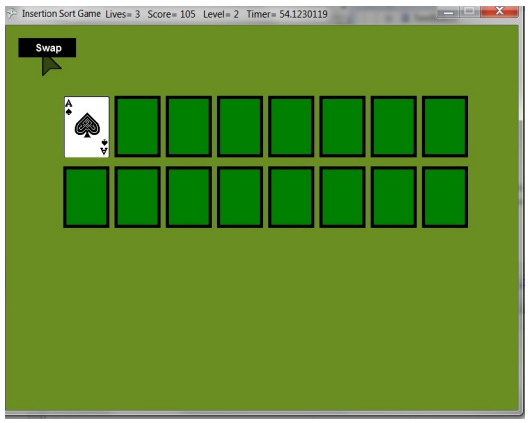

Figure 6.48: Insertion Sort–[Ace] In Place

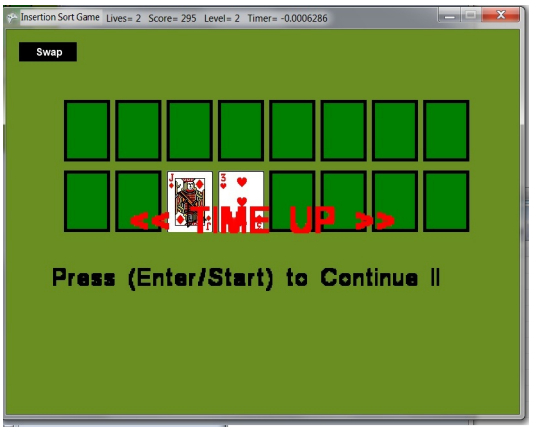

Figure 6.49: Insertion Sort–Time Ends

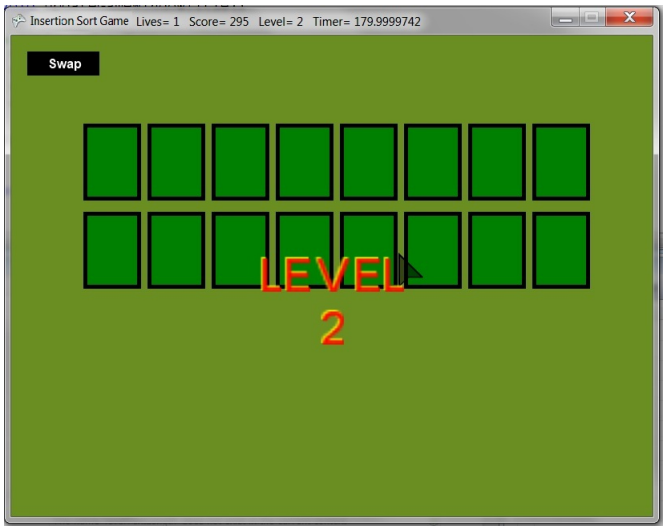

Figure 6.50: Insertion Sort–Life Lost

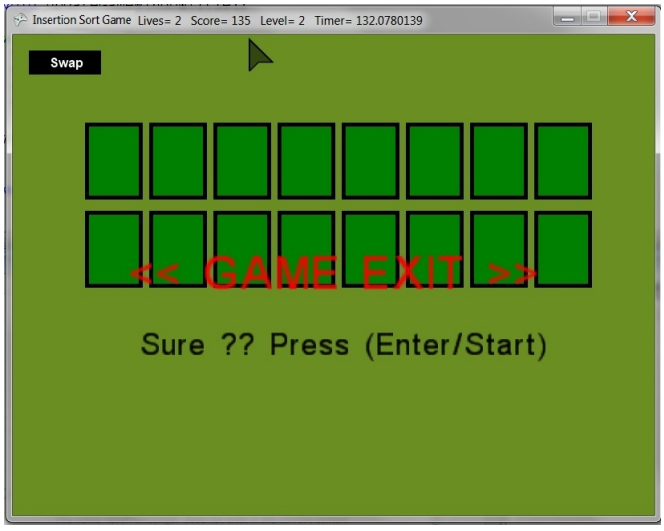

Figure 6.51: Insertion Sort–Game Exit

6.5 Linked List Game

6.5.1 Linked list

The linked list is a very important dynamic data structure. Basically, there are two types of linked lists: single-linked list and double-linked list. In a single-linked list every element contains some data and a link to the next element, which allows us to keep the structure. On the other hand, every node in a double-linked list also contains a link to the previous node. The operations on a single-linked list are traversal, inserting a node, and removing node algorithms.

Traversal Algorithm

Beginning from the head: 1) check, if the end of a list hasn't been reached yet; 2) do some actions with the current node, which is specific for particular algorithm; 3) current node becomes previous and next node becomes current. Go to the step 1.

Inserting Node Algorithm

There are four cases, which can occur while adding a node to a linked list, as shown in the following:

1. Empty list: when list is empty, which is indicated by (head = NULL) condition, the insertion is quite simple. Algorithm sets both head and tail to point to the new node.

2. Add first: new node is inserted right before the current head node, and can be done in two steps: a) update the next link of the new node to point to current head node; b) update head link to point to the new node.

3. Add last: new node is inserted right after the current tail node, which can be done in two steps: a) update the next link of current tail node to point to the new node; b) update tail link to point to the new node.

4. General case: new node is always inserted between two nodes, which are already in the list. Head and tail links are not updated in this case. Such an insert can be done in two steps: a)

update "previous" node link to point to the new node; b) update the new node link to point to "next" node.

Removing Node Algorithm

There are four cases, which can occur while removing a node from a linked list, as shown in the following:

1. When the list has only one node, which is indicated by the condition that the head points to the same node as the tail, the removal is quite simple. Algorithm disposes the node, pointed by head (or tail) and sets both head and tail to NULL.

2. Remove first: first node (current head node) is removed from the list. It can be done in two steps: a) update head link to point to the node next to the head; b) dispose removed node.

3. Remove last: last node (current tail node) is removed from the list. It can be done in three steps: a) update tail link to point to the node before the tail. In order to find it, list should be traversed first, beginning from the head; b) set next link of the new tail to NULL; c) dispose removed node.

4. General case: node to be removed is always located between two list nodes. Head and tail links are not updated in this case. Such a removal can be done in two steps: a) update next link of the previous node to point to the next node relative to the removed node; b) dispose removed node.

6.5.2 Game Design Prototype

First–General Game Design Elements

1. Game Idea: the main idea of the game is building a chain of connected dominoes according to the linked list add and remove algorithms.

2. Game Topic: dominoes board game.

3. Game Goal: simulating linked list operations algorithms.

4. Game Start: the game starts by displaying one game level that includes one uncovered domino. The level HUD includes all default attributes of the algorithm game HUD.

5. Game Levels: the game has several levels; at each new level, the number of the dominoes in the pack is increased to make the game more challenging.

6. Game End: default settings of the algorithm game.

7. Game Milestone Events: default settings of the algorithm game.

8. Game Exit: default settings of the algorithm game.

Second–Game Appearance Design

- Game Graphic Items:
  - A pack of dominoes with their values covered.
- Game Assets:
  - Textures: 28 dominoes textures.
  - Sounds: TimeEnds, LostLife, LostGame, and WonGame.
  - Fonts: Arial.

Third–Game Mechanics Design

- Game Properties: all default properties, which are initialized as follows: Maximum Level Number=3, Maximum Player Lives=3, Maximum Level Time=1 minute, and Game Name=Single Linked List in addition to default calculated properties.
- User Interface–Input Design: supporting keyboard, mouse, and Xbox gamepad.
- Game Screens Design: the game screens are all default algorithm game screens.
- Game-Play: in addition to the general playing rules that have been included in each algorithm game, the specific playing rules that simulate the algorithm steps is as follows:

1. Depending on their values, the player must build a chain of dominoes as fast as possible in a fixed time, where adjacent dominoes must have the same values from each side.
2. Continuously, the player removes one domino from the pack with a different value and adds it to the dominoes chain. The player must add and delete dominoes until reaching the required combination.
3. All dominoes are connected with each other using a connector; if a domino is added or deleted improperly, it falls, and the Player Score is decreased by one point; otherwise, when the domino is added correctly the Player Score is increased by five points.
4. If the level time ends without building the chain, the player loses the level.
5. At each new level, the number of the dominoes in the pack, Level Timer, and Level Number are increased.

6.6 Binary Search Tree Game

6.6.1 Binary Search Tree

The binary search tree is a data structure, which meets the following requirements: it is a binary tree so each node has at most two children; each node contains a value, which is less than the values of its right subtree and greater than the left subtree values. The binary search tree operations are search, add a value, and remove a value algorithms.

Search Algorithm

1. Starting from the root, check whether value in current node and searched value are equal. If so, value is found.
2. Otherwise, if searched value is less than the node's value; if current node has no left child, searched value does not exist in the BST; otherwise, handle the left child with the same algorithm.

3. If a new value is greater than the node's value; if current node has no right child, searched value does not exist in the BST; otherwise, handle the right child with the same algorithm.

Add Value Algorithm

1. Search for a place to put a new element.
2. Insert the new element to this place.

Remove Value Algorithm

1. Apply the search algorithm to find the parent of the node that has the value to be deleted.
2. Then, there are three cases, which are described below:
   (a) Node to be removed has no children. In this case, the algorithm sets corresponding link of the parent to NULL and disposes the node.
   (b) Node to be removed has one child. In this case, the node is cut from the tree and algorithm links single child (with its subtree) directly to the parent of the removed node.
   (c) Node to be removed has two children. In this case, the algorithm finds a minimum value in the right subtree, replaces the value of the node to be removed with found minimum. Now, right subtree contains a duplicate, so the algorithm applies remove to the right subtree to remove the duplicate.

6.6.2 Game Design Prototype

First–General Game Design Elements

1. Game Idea: to build the binary search tree given values one after another as fast as possible.
2. Game Topic: building blocks environment.
3. Game Goal: to simulate add, search, and remove operations of the binary search tree.

4. Game Start: the game starts by displaying one game level that shows one node of the tree with a given value. The level HUD includes all default attributes of the algorithm game HUD.
5. Game Levels: the game has several levels; at each new level, the number of the tree nodes is increased to make the game more challenging.
6. Game End: default settings of the algorithm game.
7. Game Milestone Events: default settings of the algorithm game.
8. Game Exit: default settings of the algorithm game.

Second–Game Appearance Design

- Game Graphic Items:
  - Nodes of the tree, each node having a value.
- Game Assets:
  - Textures: Node.
  - Sounds: AddNode, RemoveNode, TimeEnds, LostLife, LostGame, and WonGame.
  - Fonts: Arial.

Third–Game Mechanics Design

- Game Properties: all default properties, which are initialized as follows Maximum Level Number=3, Maximum Player Lives=3, Maximum Level Time=1 minute, and Game Name=Binary Search Tree in addition to default calculated properties.
- User Interface–Input Design: supporting keyboard, mouse, and Xbox gamepad.
- Game Screens Design: the game screens are all default algorithm game screens.
- Game-Play: in addition to the general playing rules that have been included in each algorithm game, the specific playing rules that simulate the algorithm steps are as follows:

1. The player must build a binary search tree as fast as possible by inserting a new node in its correct place in the tree.
2. If the player adds the node in its correct place, the Player Score is increased by five points; otherwise, the Player Score is decreased by one point.
3. If the level time ends without building the tree, the player loses the level.
4. At each new level, the number of the nodes, Level Timer, and Level Number are increased.

Chapter 7: Serious Algorithm Game Visualizer (Serious-AV)

In this chapter, we explain the implementation of Serious Algorithm Game Visualizer (Serious-AV), a visualization system that has been designed and developed to demonstrate the AVuSG framework by providing several viewers and designers. First we explain the design of the system; then, give its real implementation.

7.1 Serious-AV Components

Serious-AV is an educational gaming system that produces three types of algorithm visualizations: text, flowchart, and computer game. It has two main sub-systems (Figure 7.1): Serious-AV Viewer and Serious-AV Designer.

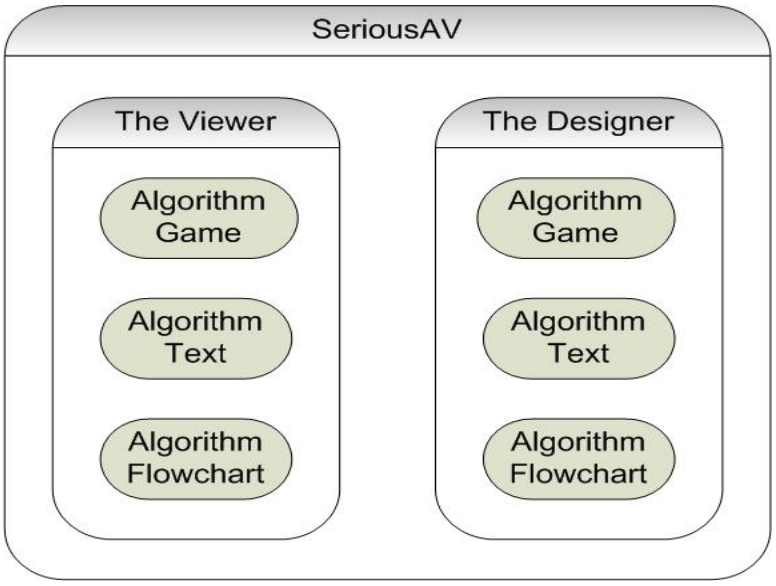

Figure 7.1: Serious-AV Components

1. Serious-AV Viewer consists of three viewers:

(a) The Algorithm Text Viewer shows the text of the algorithm.

(b) The Algorithm Flowchart Viewer presents the flowchart of the algorithm to the learner.

(c) The Algorithm Game Viewer displays the algorithm game and its self-running demo.

2. Serious-AV Designer consists of three designers:

(a) The Algorithm Text Designer is used to develop the algorithm text.

(b) The Algorithm Flowchart Designer is used to design the algorithm flowchart.

(c) The Algorithm Game Designer is used to design the algorithm game and its self- running demo.

As Figure 7.2 illustrates, the developer uses a Serious-AV Designer to produce an algorithm visualization form, then exports it to the corresponding Serious-AV Viewer to be viewed later by the learner.

Figure 7.3: Serious-AV Main System

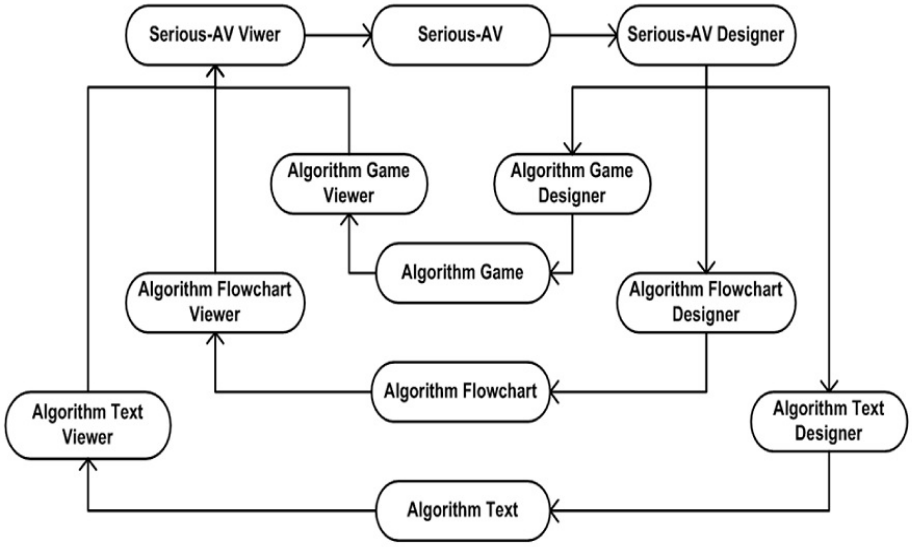

Figure 7.2: Serious-AV Architecture

7.2 Serious-AV Viewer

Serious-AV Viewer has three viewers to enable the learner to view the algorithm text, flowchart, and game. As Figure 7.4 shows, to view an algorithm the user must choose its name from the Algorithms List, then click on a button to watch the corresponding algorithm visualization.

Figure 7.4: Serious-AV Viewer Buttons

7.2.1 Algorithm Text Viewer

This viewer (Figure 7.5) shows the algorithm text to the learner. It provides the learner with many options to manipulate the viewed text, such as copy, cut, paste, and print features.

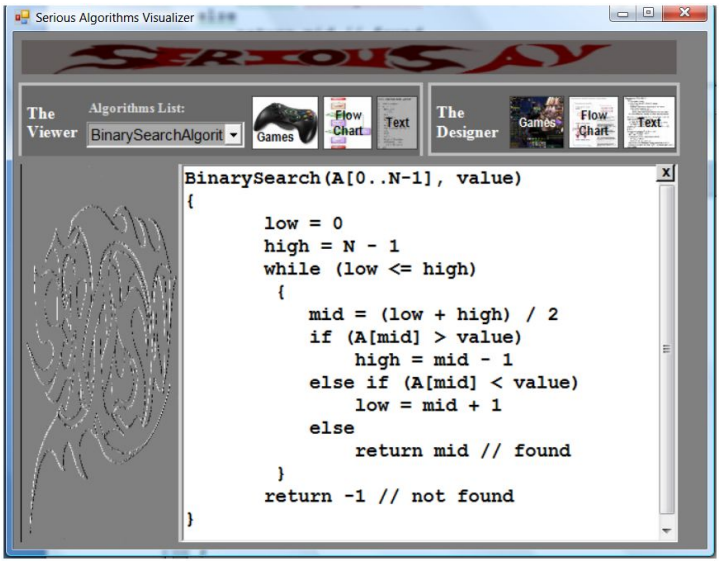

Figure 7.5: Algorithm Text Viewer

7.2.2 Algorithm Flowchart Viewer

This viewer (Figure 7.6) presents the algorithm flowchart to the learner. It provides the learner with many options to manipulate the viewed flowchart, such as copy, cut, paste, and print features.

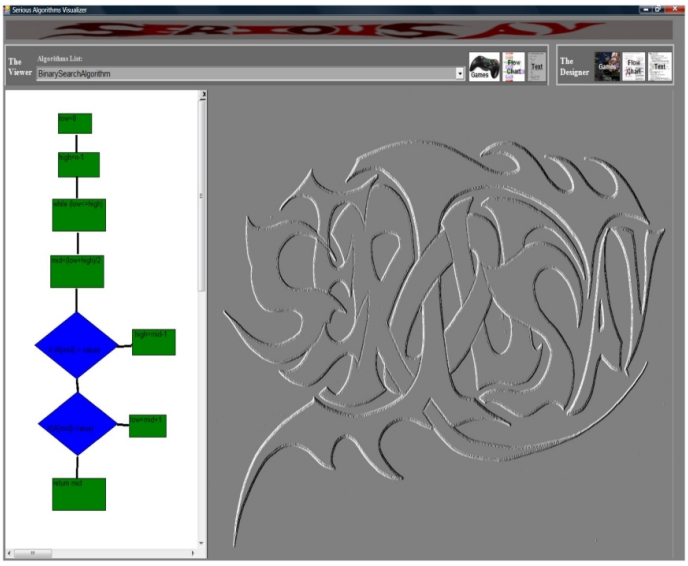

Figure 7.6: Algorithm Flowchart Viewer

7.2.3 Algorithm Game Viewer

This viewer (Figure 7.7) displays the algorithm game screen to the learner to play the game or watch its self-running demo.

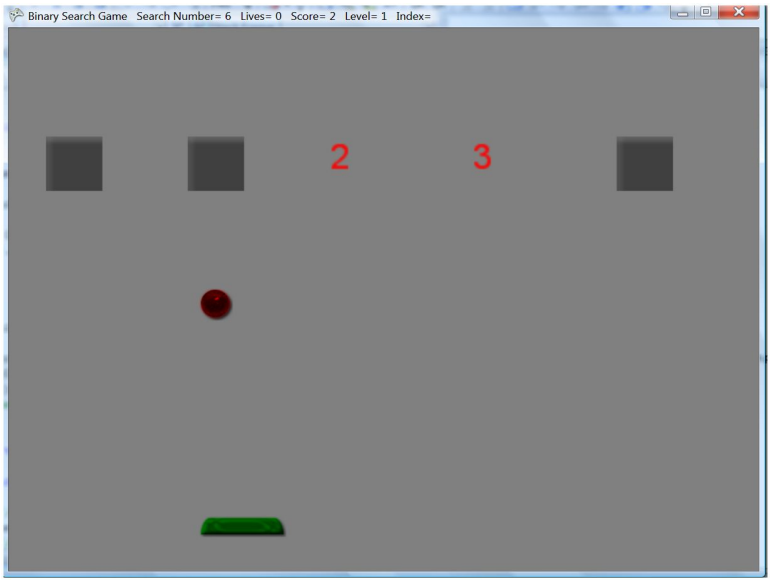

Figure 7.7: Algorithm Game Viewer

7.3 Serious-AV Designer

Serious-AV Designer has three designers to facilitate the creation of the algorithm text, flowchart, and game to the developer. These designers have a graphical user interface that simplifies the creation of the high-level visualizations using as little code as possible. As Figure 7.8 shows, there is a button to launch each of these designers by the developer.

Figure 7.8: Serious-AV Designer Buttons

Algorithm Text Designer

To simplify the creation of the Algorithm Text, this designer (Figure 7.9) contains the following components:

- Text Editor: enables the developer to write the algorithm text.
- File Menu: allows the developer to save and open the algorithm text to and from a local directory in addition to exporting it to the Algorithm Text Viewer.
- Edit Menu: enables the developer to cut, copy, paste, and delete any written text on the editor.
- Format Menu: allows the designer to format the font of the written text.

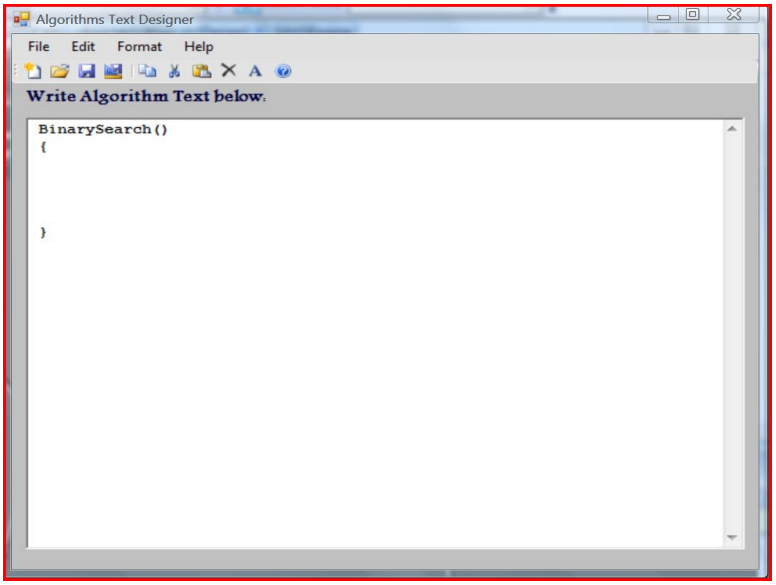

Figure 7.9: Algorithm Text Designer

Algorithm Flowchart Designer

To simplify the creation of the algorithm flowchart, this designer (Figure 7.10) contains the following components:

- Graphics Editor: enables the developer to create the algorithm flowchart by dragging items from the Flowchart Toolbox to it.

- Flowchart Toolbox: includes a set of standard flowchart shapes, such as arrow, process, data, decision, preparation, manual operation, and termination. The developer selects and drags the required shape to the editor, then adds the required text to it to create the flowchart.
- File Menu: allows the designer to save and open the completed flowchart to/from a local directory in addition to exporting it to the Algorithm Flowchart Viewer.
- Edit Menu: allows the designer to copy, paste, move, resize, and delete any selected flowchart shape on the editor.

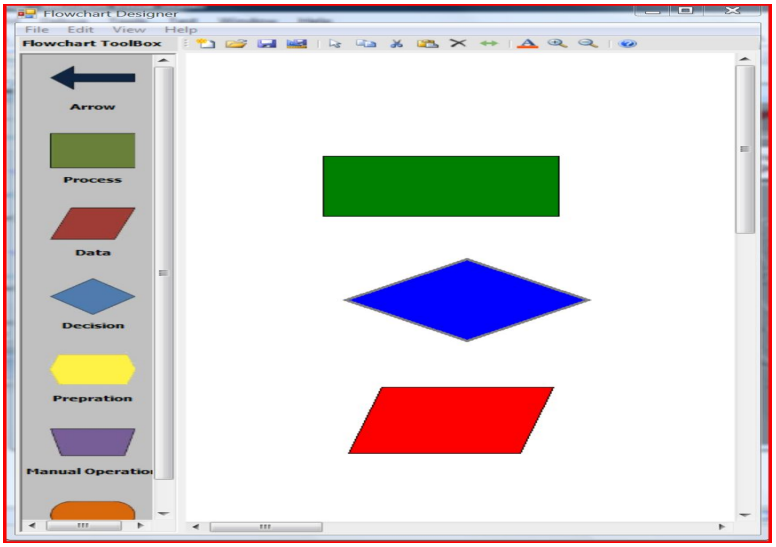

Figure 7.10: Algorithm Flowchart Designer

Algorithm Game Designer

This designer (detailed in Chapter 8) is used to build the algorithm game. It has a user interface (Figure 7.11) that has been built specifically to simplify the development of the game by providing the following components:

- File Menu to create a new or open an existing algorithm game project.
- Code Editor to write the algorithm game code.
- Edit Menu to edit the algorithm code.

- Build Menu to build the algorithm game.
- Debug Menu to debug and execute the algorithm game.
- Game Editors Menu to invoke one of the five Graphics Editors (Classes, Properties, Assets, Graphic Items, and Game Screens) that have been included in the designer to allow for the visual game design.
- Solution Explorer to show the algorithm menu project files.

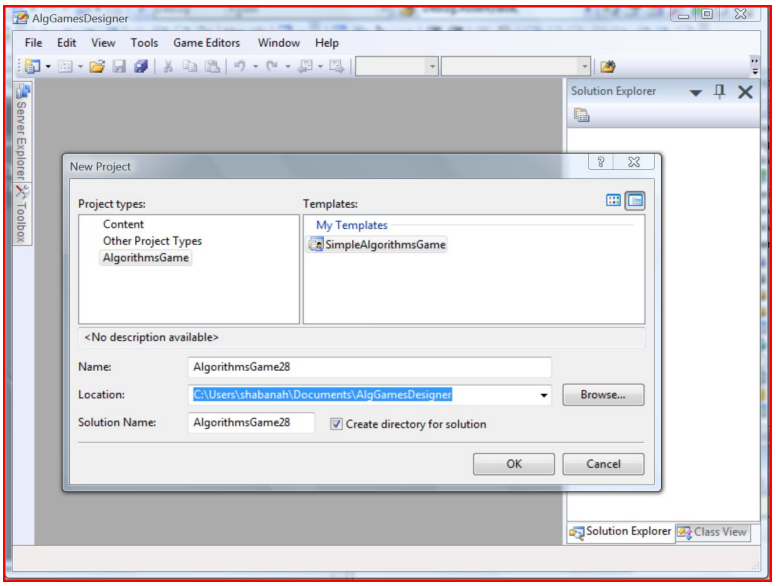

Figure 7.11: Algorithm Game Designer

7.4 Implementation Issues

The main window of Serious-AV has been implemented as a Windows Forms Application using C# and Microsoft .Net Framework in addition to the Algorithm Text Designer and Viewer, and the Algorithm Flowchart Designer and Viewer. Algorithm Games have been implemented as Microsoft XNA Games using XNA Framework, so the Algorithm Game Viewer is implemented as an XNA game window. Moreover, the Algorithm Game Designer has been implemented as a Visual Studio-Isolated Shell using C#, Visual Studio SDK, and has been customized using VSpackages. The

Visual Studio-Isolated Shell is a stand-alone application that provides an integrated development environment similar to VS 2008. However, the Isolated Shell does not provide support for any programming language, so we have extended it using Visual Studio SDK to add support for C# and XNA projects building operation depending on MSN build. Moreover, we have added Debug and Run support for the created projects and added menus to access these operations. Moreover, custom and nested project templates have been added to create projects of type XNA game and XNA content in addition to several item templates for each project type. SAVGEngine has been implemented as an XNA library. Algorithm Game Template has been implemented as Microsoft XNA Game template. Graphics Editors have been implemented as Windows Forms Applications using C#, XNA, and Microsoft .Net Framework. (All those systems are described in detail in Appendix A).

Chapter 8: Algorithm Game Designer

This chapter presents the design and implementation of one of the most valuable parts of Serious-AV: Algorithm Game Designer, along with its visual editors (graphic Editors) and the game engine (SAVGEngine) consumed by the generated code. Algorithm Game Designer is a visual game development environment through which the game designer can specify the main game configuration (resolution, screen mode, etc.), game states (screens), game assets, graphic items, classes, and properties. Specific instances of other parts related to the game design, such as Algorithm Game Architecture, Algorithm Game Template, and SAVGEngine Repository are also presented.

8.1 Algorithm Game Designer Components

As shown in Figure 8.1, the Algorithm Game Designer has five major components that simplify the creation of new algorithm games as follows:

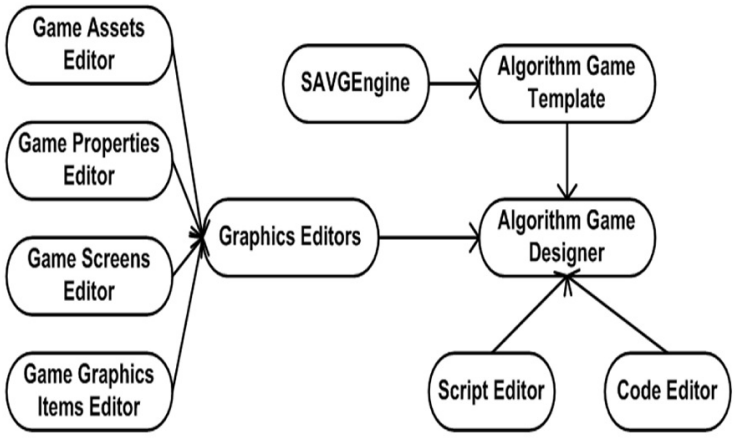

Figure 8.1: Algorithm Game Designer Architecture

1. Algorithm Game Engine (SAVGEngine) to help in the re-use of different game components.

2. Algorithm Game Template, to be used in the creation of new algorithm games.

3. Algorithm Game Components Repository to provide ready-to-use algorithm game components.

4. Code Editor to help in the algorithm game implementation.

5. Developer Graphics Editors (Properties, Assets, Graphic Items, Classes, and Game Screens) to help in the visual creation of the algorithm game content.

8.2 Algorithm Game Architecture

The architecture of a typical algorithm game is described in Figure 8.2. It consists of many modules including game content, game graphic items, game screens, input, and game-play.

- Game Content: includes several types of files that can be grouped into two main categories, including:

    1. Game Assets: textures, fonts, models, effects, and sound files.
    2. Script and Data: xml files.

- Game Graphic Items: define the game entities or objects that make the game world. Each graphic item is either a 2D Sprite or a 3D Model with attributes, such as size, position, color, texture, and name in addition to behaviors like render, move, and update. Examples of graphic items of a typical algorithm game are as follows:

    1. Node: animated item used to visualize one element of the algorithm data structure, such as Card, Box, and Domino.
    2. Data Structure: a collection of similar nodes that have been organized according to specific rules to visualize an algorithm data structure; for example, a deck of cards visualizes an array and a map of cities visualizes a graph.
    3. Playing Items: animated items used to play the game, such as Ball, Paddle, and Shooter.

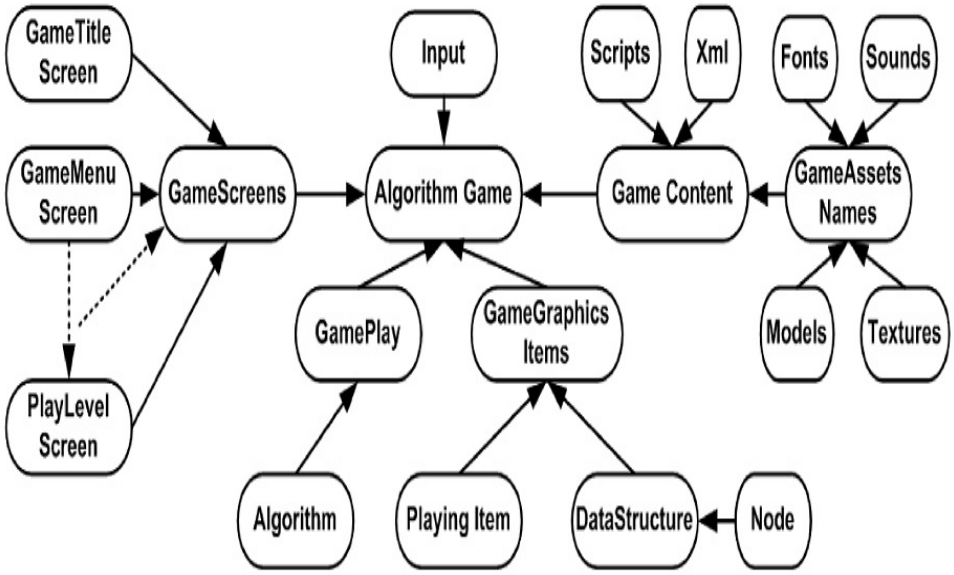

Figure 8.2: Algorithm Game Architecture

- Game Screens: each algorithm game has a set of game screens, each screen describes the state of the game at any one time during the game life cycle. Examples are Play and Won screens.
- Input: handles the game input from the mouse, keyboard, and game pad. The required feedback for each player input event must be implemented.
- Game-Play: is responsible for implementing the playing rules of the game according to the visualized algorithm behavior.

8.3 Algorithm Game Engine (SAVGEngine)

A game engine is the heart of all new video games and can be used many times to build different games [149]. Specifically, game engines used to automate game-related functions like loading and rendering animations, playing digital sounds, controlling input devices like joysticks, and simulating physics-based special effects [125]. Serious Algorithm Visualization Game Engine (SAVGEngine) is geared toward creating a two- and three-dimensional Algorithm Games. In particular, SAVGEngine has no specific game logic coded directly in it, it only provides a common set of base

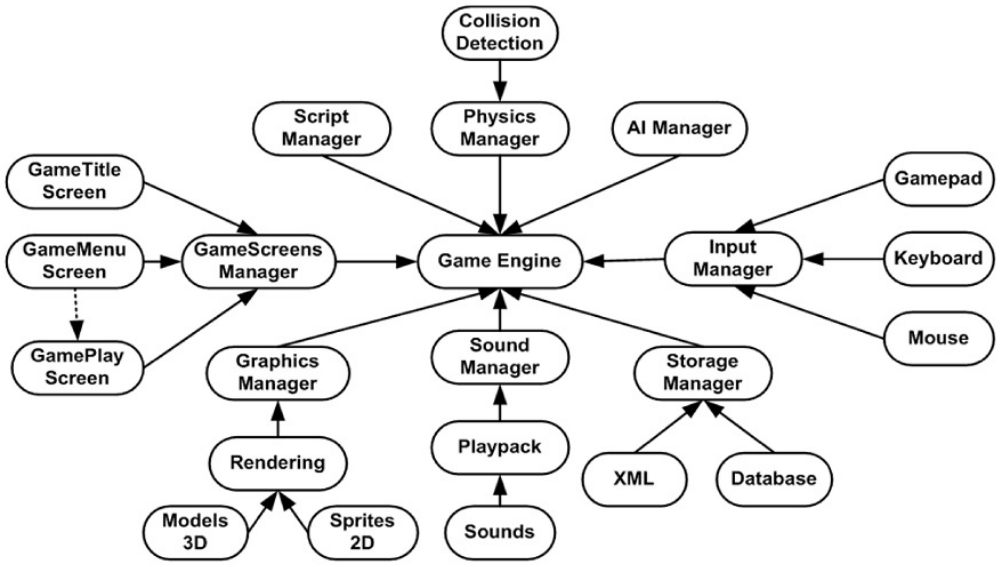

Figure 8.3: SAVGEngine Architecture

functionality that can be re-used for any algorithm game.

8.3.1 SAVGEngine Design

Hoetzlein and Schwartz [150] specify six requirements to be considered when designing an educational game engine, including the following:

1. The language used to implement the engine is the same language that is used to access the engine's functionality.

2. The choice of 2D or 3D games must depend on the educational level of the engine targeted audience. Despite that, 2D games are simpler to design and develop than 3D games; some may find 3D games development more satisfying because of the challenges these games design may produce.

3. Any game engine design must take in account the variety of audiences that may use the engine including professionals, hobbyists, students, and even non-computer scientists.

4. The game engine must be provided free of charge for students.

5. It is very important to provide the engine source code for reviewing by learners.

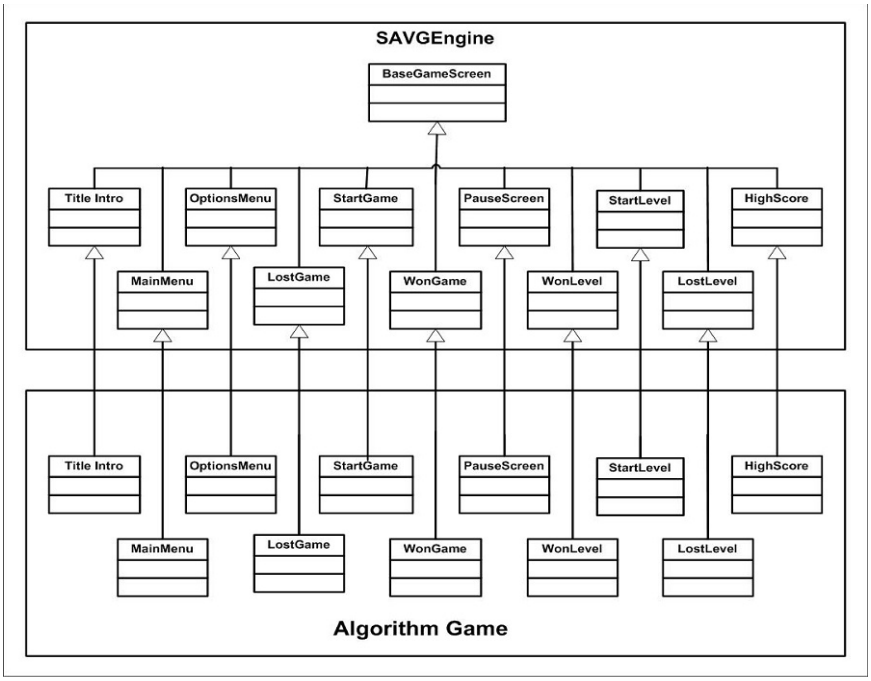

Figure 8.4: SAVGEngine–Game Screens Structure

6. It is very important for the educational game engine to provide high quality documentation and tutorials.

SAVGEngine has been designed as an educational game engine by applying the previous features of educational game engines. First, the same language (C#) has been used in both implementing SAVGEngine and accessing its functionality. Other SAVGEngine features include full support for 2D and 3D games, designed for various audiences, such as students and instructors, offered free of charge, its source code is included with it, and providing documentation.

8.3.2 SAVGEngine Architecture

In general, the implementation of a game engine must take many issues into account. The first issue is the adoption of a suitable programming language, which will be used not only to program the game engine but also by game programmers consuming the engine. Such a choice must consider easiness-to-use, efficiency, and portability. Other game engine requirements are modular and flexible architecture; good abstraction level; basic game elements specification, such as the game map

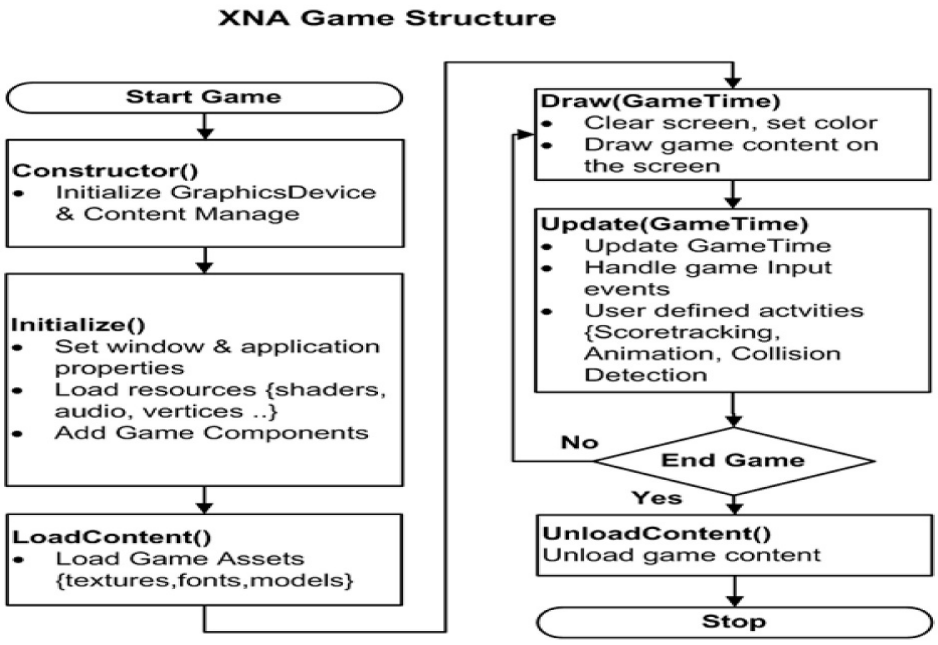

Figure 8.5: XNA Game Class Structure

(or background) and entities, world management, including physical modeling; input handling support (keyboard, mouse, joystick, etc.), game elements rendering (2d or 3d); sound and background music support, basic artificial intelligence algorithms, connectivity support (remote data exchange, session management, data synchronization, etc.), built-in support tools, a built-in script language for the definition of specific tasks, performance, and usability [151]. Specifically, a typical game engine must have several helper and manager modules to provide general support for the game engine, including an input module to handle the game input-related operations, graphics module to handle the game graphics-related operations, sound module to handle the game sound-related operations, physics module to handle the game physics-related operations, and math module to handle the game math-related operations. Therefore, SAVGEngine is designed to be responsible for the following major modules (Figure 8.3):

- Graphics manager: encapsulates all functionality of rendering and presenting 2D and 3D graphics, including sprites, models, effects, and layers.
- Sound manager: allows the playback of audio clips through a basic two-speaker setup.

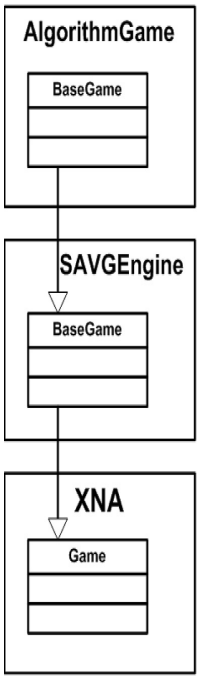

Figure 8.6: BaseGame

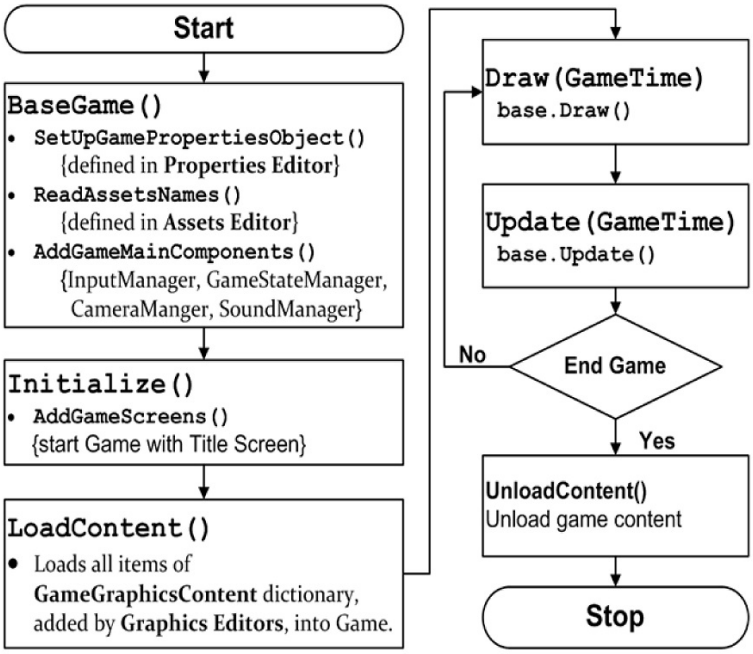

Figure 8.7: SAVGEngine–BaseGame Class

- Input manager: recognizes user input originated from the keyboard, mouse, and an Xbox game pad.
- Physics manager: allows for basic collision detection between game entities.
- Artificial Intelligence manager: allows for the creation of intelligent game characters or objects.
- Storage manager: allows for the storage of the player information and results in addition to game data.
- Game Screens manager: handles the transition between different game screens by storing them in a stack data structure called GameScreens stack (Figure 8.4). This stack must be updated with required game screens and its status must be maintained during the game-play.
- BaseGame class: provides the new algorithm game with timing and rendering loops. The BaseGame class of every created algorithm game is derived from this BaseGame class, which in turn is derived from the Game class of XNA Game (Figure 8.5) as Figure 8.6 shows. In

addition, this BaseGame class exposes key methods, properties, and events for the game life-cycle by implementing the following methods (Figure 8.7):

1. `Initialize`(); this method initializes the game window default attributes, such as width and height of the game rendering viewport, camera, and game font.
2. `ReadAssetsNames`(); this method loads the game assets into the GameGraphicsContent dictionary.
3. `AddGameScreens`(); this method loads the game screen into the GameScreens stack.
4. `AddGameMainComponents`(); this method initializes and adds basic game components, such as the input manager, the camera manager, and the sound manager into the game.
5. `SetUpGamePropertiesObject`(); this method loads the game properties into the GameProperties dictionary.
6. `SetGameGraphicItems`(); this method loads the game graphic items into the GameGraphicItems dictionary.

- GamePlay class: handles how the game is played by exposing general game-play structure that can be used with any algorithm game as shown in Figure 8.8. This structure implemented all basic playing rules of the game, so when designing a new game, only a few methods that define the specific playing rules need to be implemented. Algorithm 8.1 describes the mechanism of the GamePlay class.

8.4 Algorithm Game Template

The Algorithm Game Template is a blueprint that is used in the creation of a new algorithm game. Each created new game has a copy of the template content, which provides it with general game-related operations.

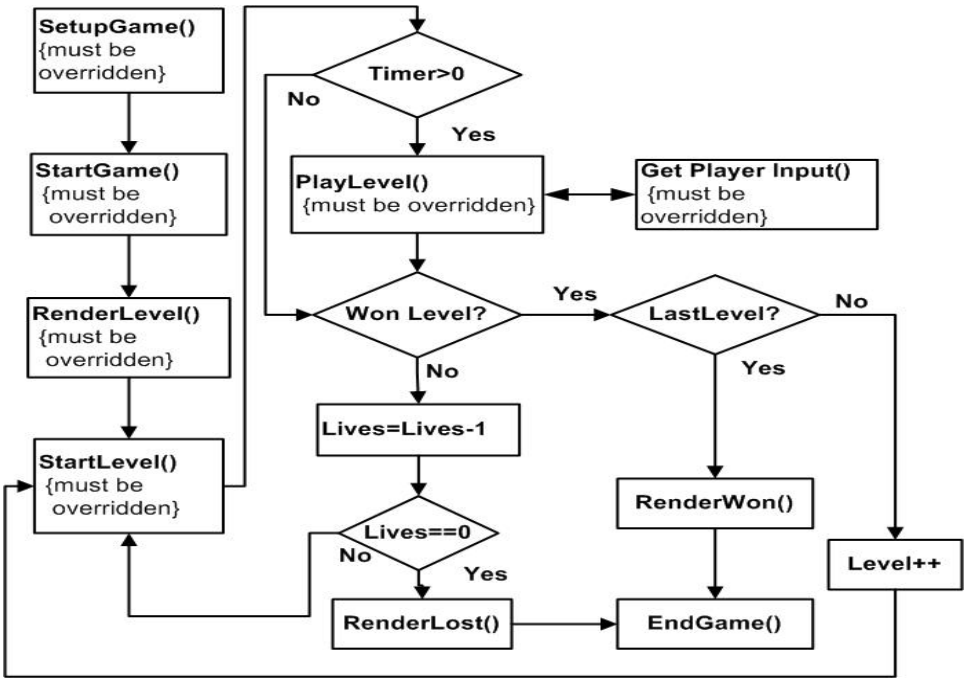

Figure 8.8: SAVGEngine–GamePlay Class

8.4.1 Algorithm Game Template Components

The Algorithm Game Template includes a module for each algorithm game component as shown in Figure 8.9:

- Program: the main class in the game project, it starts the execution of the game.
- Content Project: handles the loading and building of the game content, such as texture, sound, models, effects, and font files. A sample of the game content files, such as basic textures and fonts are included in the template.
- Game Screens: fourteen default game screens are enclosed in the template, including Title, Main Menu, Play, High Scores, Start Level, Win Level, Lost Level, Player Name, Player Report, Won Game, Lost Game, Exit, Credits, and Pause Screens.
- SAVGEngine: a copy of SAVGEngine is included in the template to enable the developer to access the engine various modules, such as input, sound, and storage managers.

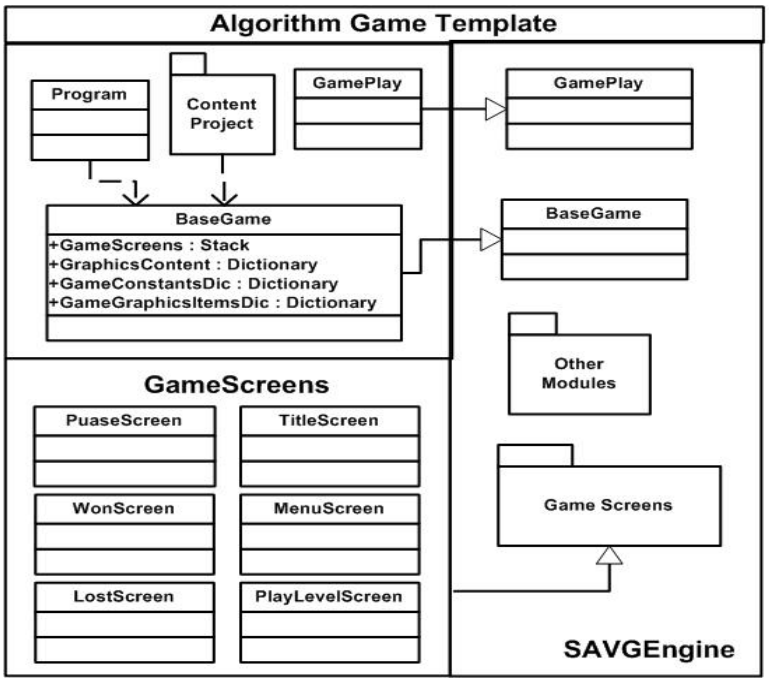

Figure 8.9: Algorithm Game Template Content

- BaseGame class: the main game class that is responsible for basic game operations and components. It has been derived from the BaseGame class of the SAVGEngine, which provides the game with many useful modules and the general game loop. To initialize the game, only three Set-Up methods (Figure 8.10) must be overridden in the BaseGame class as follows:

  1. `SetUpGameGraphicItems`(); in this method, the game designer needs to set up the game graphic items that have been defined previously in the Game Graphic Items Editor.
  2. `SetUpGameProperties`(); in this method, the game designer needs to set up the game properties that have been defined previously in the Properties Editor.
  3. `SetUpGameScreens`(); in this method, the game designer needs to set up the game screens that have been defined previously in the Game Screens Editor.

- GamePlay class: defines playing rules of the algorithm game. It inherits the general gameplay structure, the rendering and timing game loops from the GamePlay class of the SAVGEnigne. In addition, it provides a set of methods (Figure 8.11), which must be implemented

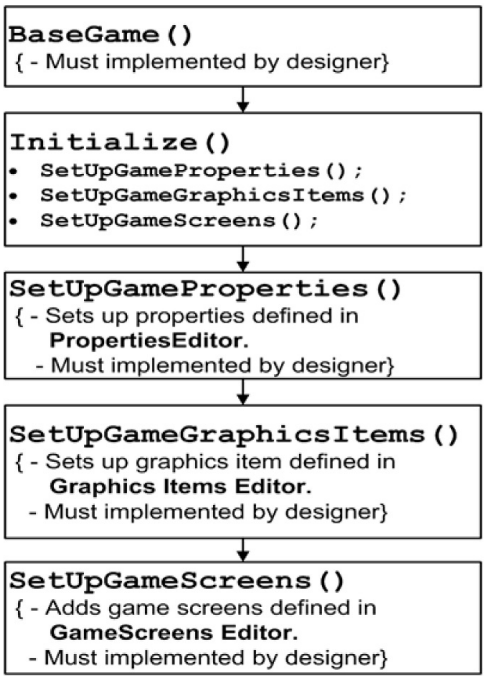

Figure 8.10: Algorithm Game Template–BaseGame Class

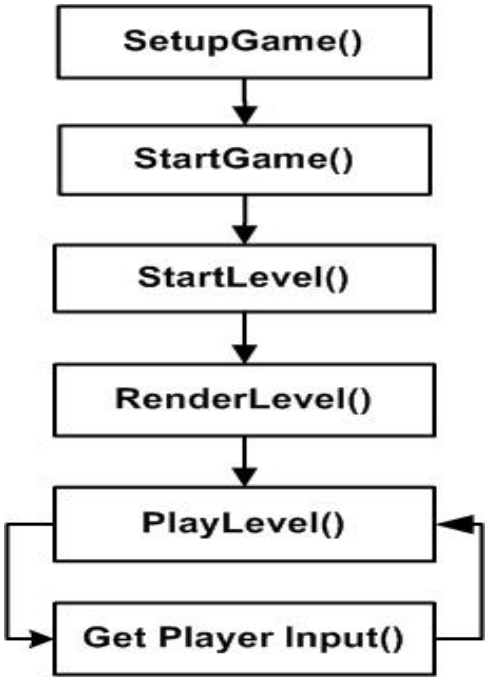

Figure 8.11: Algorithm Game Template–GamePlay Class

for each algorithm game designed. These methods create a general framework for the algorithm game playing rules as explained in the following:

1. `SetupGame`(); this method is called once at the beginning of the game execution to initialize the game.
2. `StartGame`(); this method is called each time the game starts or restarts, to define the game start parameters and functions.
3. `StartLevel`(); this method defines the level start parameters and functions and is called at the start of each level.
4. `RenderLevel`(); this method is called at the start of each level to render it on the screen.
5. `PlayLevel`(); this method is called at each frame during the game execution to apply the game-play rules.
6. `GetPlayerInput`(); this method is called at each frame during the game execution

to respond to the player input.

8.5 Algorithm Game Components Repository

This repository includes ready-to-use algorithm game components that can be altered and plugged in to the new game. Some of these items are:

- Data Structures: known data structures, each data structure having attributes, such as name, size, elements, etc. In addition, all queries and modifying operations related to this data structure are implemented, such as delete, insert, and predecessor. Examples are deck of cards to represent an array.

- Game Assets: some font, sound, effect, model, and texture files are included in this repository to be used in new algorithm games.

- Game Classes: some classes that are related to the game items, such as Score, Ball, Paddle, Button, Card, Domino, Deck, Pack, and Cube.

- Graphic Items: some graphic items are implemented, such as Nodes (block, cube, sphere, tree leaf, card, domino, etc.), Playing Tools (Paddle, Ball, and Shooter), and Buttons (different shapes and sizes).

- Game Screens: sixteen default game screens that can be used in the algorithm game have been implemented and included in the repository. These game screens are explained in the following:

    1. Title Screen: displays the game title, logo, and name.
    2. Player Name Screen: displays on screen keyboard at the game start to input the player name.
    3. Main Menu Screen: displays the game main menu options and handles the player choices.

4. Start Level Screen: displays the start of one game level for the player including the level number.
5. Play Screen: displays the game for the player to play, including a Head-Up-Display (HUD), showing several game attributes, such as Level Number, Remaining Lives, Game Name, Player Score, and Level Timer.
6. Help Screen: displays instructions of how the game is played for the player.
7. Pause Screen: allows the player to stop and resume the game at any time.
8. Exit Screen: allows the player to stop and end the game at any time.
9. Won Level Screen: displays "Level Completed" message when the player wins one level and asks the player to continue the game, then displays the next level for the player.
10. Lost Level Screen: displays the Remaining Player Lives when the player loses one level and asks the player to continue the game, then repeats the same level for the player.
11. Won Game Screen: displays a won message when the player wins the game.
12. Lost Game Screen: displays a lost message when the player loses the game.
13. Options Screen: displays the game options for the player.
14. Game Demo Screen: displays the self-running demo of the game.
15. High Score Screen: displays the five highest scores that have been achieved by all players, during the game, including: the Player Name, Score, and Level Number.
16. Player Report Screen: displays a progress report for the player for the last eight times s/he played the game, including Game Result (Lost, Won), Score, Level Number, Play Date, and Play Time.
17. Credits Screen: shows the game developers information.

8.6 Algorithm Game Designer Editors

This section describes two types of editors (code and graphics) that have been included in the Algorithm Game Designer.

8.6.1 Code Editors

Basic Editor

The Basic Editor is the main editor (Figure 8.12) to develop the algorithm game. It supports the creation of a new algorithm game project and debugging, compiling, and execution operations by providing:

- Main Menu that includes several submenus as follows:
  - File submenu to create a new or open and save an existing algorithm game project.
  - Edit submenu to edit the algorithm game code.
  - Build submenu to build the algorithm game.
  - Debug submenu to debug and execute the algorithm game.
  - Game Editors submenu to invoke one of the developer editors.
  - Other development submenus, such as Tools, Window, Help, View, and Project.
- Solution Explorer to show the algorithm game project files.
- Code Editor that supports the algorithm game coding in C# and XNA.
- Output and File Properties windows.

8.6.2 Developer Graphics Editors

The Developer Graphics Editors support the creation of different components of an algorithm game using a flexible, user-friendly graphical user interface (GUI). They produce all required game components files that compose the entire game. These editors are reliant on the game engine and can dynamically load game code modules to provide design support for game-specific code. However, it is not required to know the internal working of the game engine to work with the editors. Five developer editors have been designed so far, including Properties, Assets, Graphic Items, Classes, and Game Screens Editors.

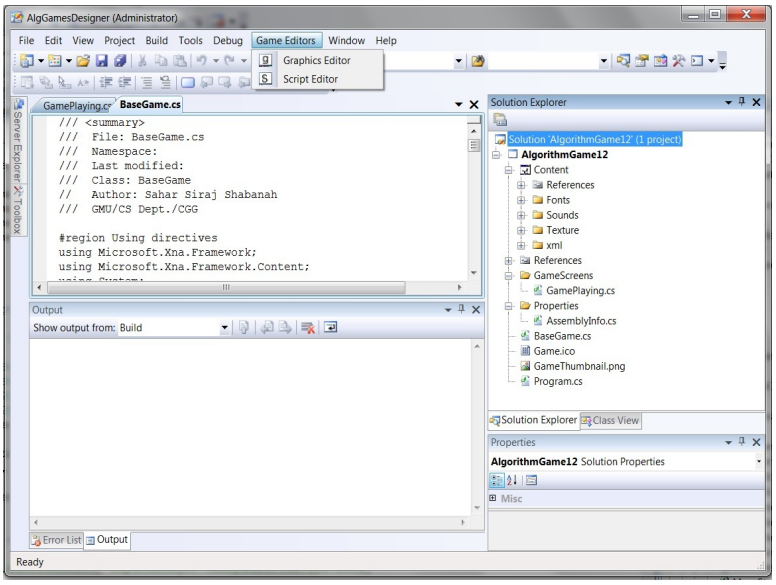

Figure 8.12: Game Basic Editor

Game Properties Editor

This editor enables the designer to enter the game properties. It has some basic game properties to be initialized in addition to the ability to add new ones. Its user interface (Figure 8.13) has three input fields:

- Content Directory that is used to enter the game content directory.
- Game Name that is used to enter the game name.
- Game Constants that is used to enter the names and values of the game attributes, such as number of lives and number of levels.

When the designer clicks on the Save button, the entered game properties are saved in an XML file (GameProperties.xml) to be imported into the game. In the game, SAVGEngine loads the Game Name to use it to set up the name of the game attribute. It also loads all saved Game Constants into the GameConstantsDic dictionary using their names as keys to simplify referencing them in the game. For example, if the item name is MaxLives, it can be called as follows:

```
LivesLimit = (int)GameConstantsDic["MaxLives"];
```

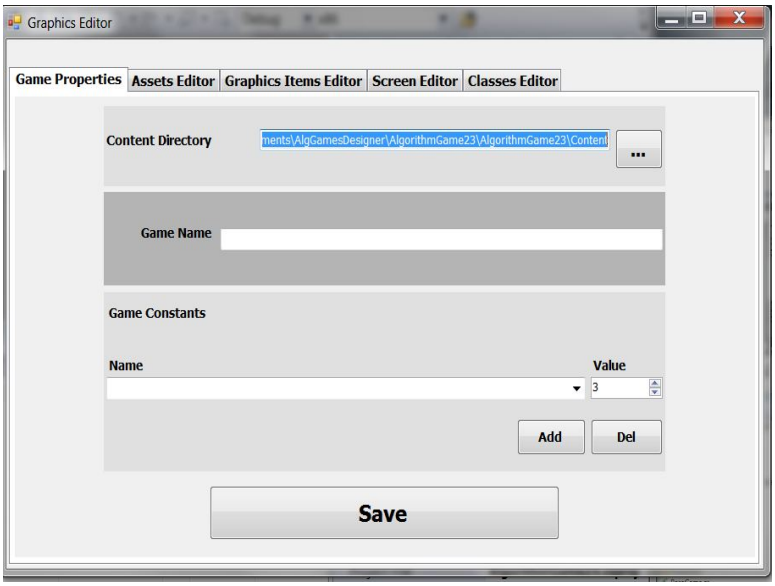

Figure 8.13: Game Properties Editor

Game Assets Editor

This editor enables the designer to load the game assets (texture, font, sound, effects, and models files) into the game. It provides various default assets files for the developer to choose from in addition to the ability to add new ones. Its user interface (Figure 8.14) consists of the following items:

- An input field for each type of game assets, including Game Fonts, Game Sounds, Game Effects, Game Models, and Game Textures. To add a new asset, the user clicks the Add button; then, chooses its file from the open file directory. To delete an item, the developer needs to select it; then, click on the Del(ete) button.
- Default assets pane that provides the developer with default game assets, which can be added to the game using the Add button.
- Preview pane that enables the game designer to view the entered texture and model before loading them into the game.

When the developer clicks on the Save Button, the entered game assets are saved into an XML file (GameAssets.xml) to be imported into the game. In the game, SAVGEngine loads all saved game

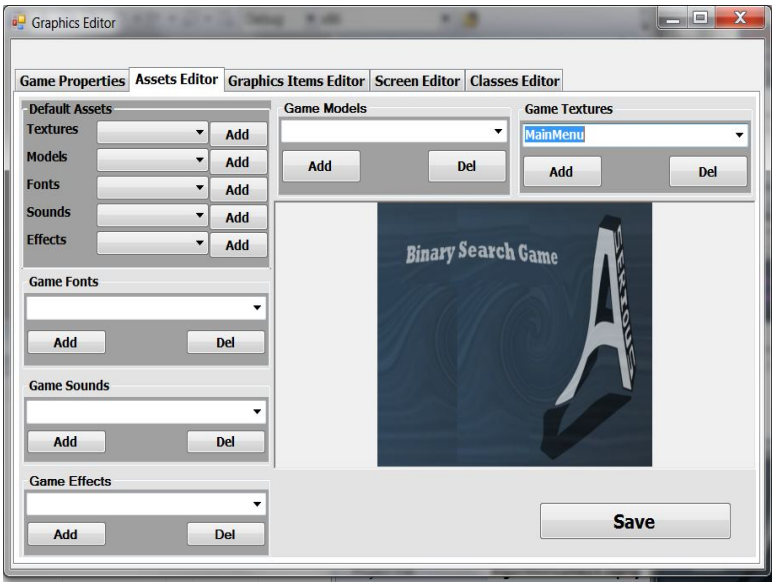

Figure 8.14: Game Assets Editor

assets into the GraphicsContentDic using their names as keys to simplify referencing them. For example, if the item name is Ball, then it may be called in the game as follows:

```
BallTexture = (tTexture)graphicsContentDic["Ball"];
```

However, most of the time, the developer does not have to write code to use the game assets, since this is done visually using the Game Graphic Items editor and Game Screens editor. Moreover, SAVGEngine sets the game font for the player using the entered font file. It also sets the Sound manager using the entered sound file to allow the developer to use it directly as follows:

```
Sound.Play("BallHit");
```

Game Classes Editor

This editor (Figure 8.15) enables the designer to enter the game classes names. It offers a list of default classes to choose from in addition to the ability to add new classes. The entered game classes are saved in an XML file (GameClasses.xml) to be imported into the game. SAVGEngine instantiates an object for each saved class and adds it to the game. To use a class in the game, the developer only needs to call it using its name without instantiation. For example, a class called

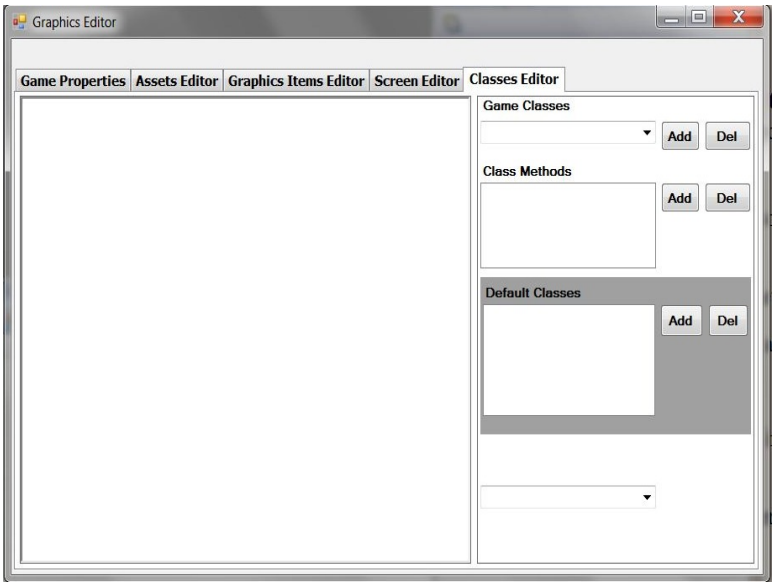

Figure 8.15: Game Classes Editor

Paddle can be used as follows:

```
pPaddle = (Paddle)base.GameGraphicsItemsDic["Paddle"];
```

Game Graphic Items Editor

This editor enables the designer to set up the game graphic items or entities. It provides many default graphic items for the developer to choose from in addition to the ability to add new items. Its user interface (Figure 8.16) consists of the following items:

- Game Graphic Items List: contains the game graphic items, which the developer can update, delete from, or add to, an item.
- Default Graphic Items: contains built-in, default items, which the developer can use directly by adding them to the game.
- Classes List: shows all classes that have been added previously by the developer using the Classes editor.
- Texture List: shows all textures that have been added previously by the developer using the Assets editor.

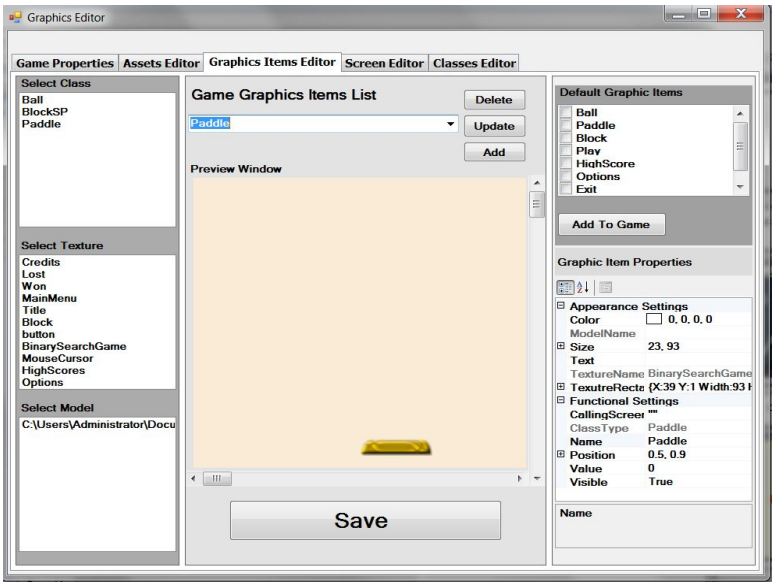

Figure 8.16: Game Graphic Items Editor

- Model List: shows all models that have been added previously by the developer using the Assets editor
- Preview Window: draws the selected graphic item.
- Graphic Items Properties: shows all properties of the item (Table 8.1) and enables the designer to set or change them.

When an item is selected from the Game Graphic Items List, it is drawn on the preview window and its properties appear.

- To update an item of the game, the developer can change its properties as required, then click on the Update button.
- To add a new item to the game, the developer must choose the class that defines its behavior, the texture or model that determines its appearance, set up its properties, give it a name, and then Finally add it to the game using the Add button.
- To delete an item in the game, the developer chooses it, then clicks on the Del(ete) button.

When the designer clicks on the Save button, all game graphic items in the list are saved into an XML file (GameGraphicsItems.xml) to be imported into the the game. In the game, SAVGEngine

Table 8.1: Game Graphic Item Properties

Property Name	Description	Type
Name	Name of the item to reference it.	String
Text	Text to be displayed on the item.	String
Calling Screen	Name of the screen the item is calling.	String
Class Type	Class that implements the item behavior.	String
Value	Numerical or alphabetical value of the item.	Alpanum
Visible	When true, item is rendered.	Boolean
Color	Color of the item.	Color
Model Name	Model defines the 3D item shape.	String
Texture Name	Texture specifies the item appearance.	String
Size	Size of the item on the screen.	Number
Position	Position of the item on the screen.	Vector
Location	Location of the item on the screen.	Point
Rotation	Rotation of the item about an axis.	Degree
Repetitions	Number of repetitions.	Number
Mass	Mass of the item.	Number
Speed	Speed of the item when moving.	Number
Direction	Direction the item moves.	Vector

loads all saved game graphic items into the GameGraphicsItemsDic dictionary using their names as keys to simplify referencing them. For example, if the item name is Ball, then it is called as follows:

```
bBall = (Ball)base.GameGraphicsItemsDic["Ball"];
```

Game Screens Editor

This editor enables the designer to load various game screens either by choosing from a list of default game screens or by adding new ones. Its user interface (Figure 8.17) consists of the following items:

- Game Screens: a list contains the game screens, which the developer can update, delete, or add. Each screen has four attributes:

    1. Screen Texture: the background texture of the screen.
    2. Screen Class: the class that implements the behavior of the screen.
    3. Graphic Items: a list of game graphic items that are associated with the screen to be displayed with it.

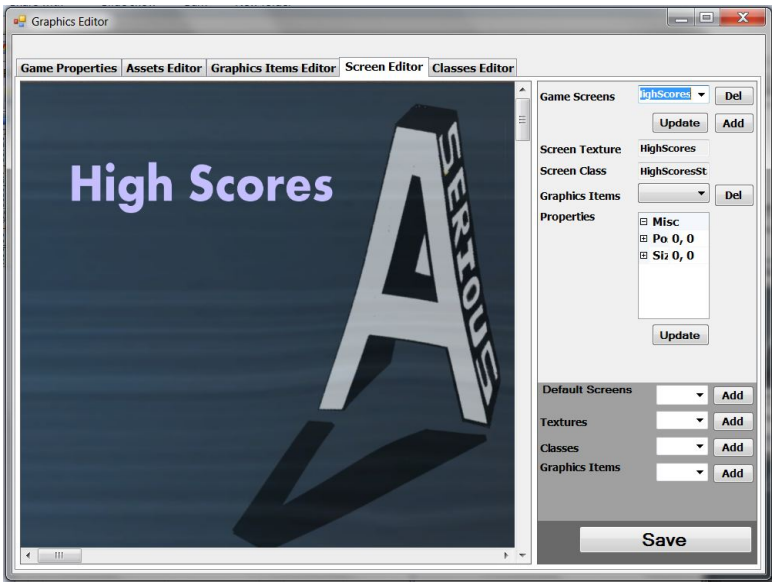

Figure 8.17: Game Screens Editor

4. Graphic Item Properties: shows size and position properties of the selected graphic item to enable the developer to properly place this item on the screen.

- Default Screens: contains built in screens the developer can use directly by adding them to the game.
- Classes: a list of all classes that have been previously added by the developer using the Classes editor.
- Textures: a list of all textures that have been previously added by the developer using the Assets editor.
- Graphic Items: a list of all graphic items that have been previously added by the developer using the Graphic Items Editor.
- Preview Pane: shows the selected screen and all its graphic items.

When a screen is selected from the Game Screens, it is drawn on the preview pane and its attributes, including texture, class, and graphic items, appear.

- To update a screen of the game, the developer can change its attributes as required, and then click on the Update button.
- To add a new screen to the game, the developer must choose the class that describes its behavior, its background texture, the graphic items associated with it by placing them on the screen, give it a name, and then finally add it to the game using the Add button.
- To delete a screen in the game, the developer chooses it, then clicks on the Del(ete) button.

When the designer clicks on the Save button, all game screens in the list are saved into an XML file (GameScreens.xml) to be imported into the game. In the game, the game engine loads the saved game screens into the GameScreensDic dictionary using their names as keys to simplify referencing them. For example, if the screen name is Title, then it is called as follows:

```
screenManager.PushState((BaseGameState)GameScreensDic["Title"].Value);
```

Algorithm 8.1 SAVGEngine–Play Game

```
1: SetupGame();{ Sets up the game for the first time}
2: StartGame();{ Starts the game }
3: GetPlayerInput();{ Implemented by the designer}
4: StartLevel();{ Push Start Level screen to the stack}
5: RenderLevel();{ Render Level on the screen}
6: PlayLevel();{ Implemented by the designer }
7: RenderWin();{ Push Won Game screen to the stack }
8: RenderLose();{ Push Lost Game screen to the stack}
9: EndGame();{ Ends the game}
10: PlayAlgGame()
  if PlayLevel() = "TimeUp" then
    GameOperation ⇐ TIMEUP
  else if PlayLevel() = "LevelCompleted" then
    if currentLevel < maxLevels then
      GameOperation ⇐ LEVELCOMPLETED
    else
      GameOperation ⇐ WON
    end if
  else if PlayLevel() = "Lost" then
    GameOperation ⇐ LOST;
  end if
12: PlayGame()
  if GameOperation =MENU then
    Menu();
  else if GameOperation =PLAY then
    PlayAlgGame();
  else if GameOperation =END then
    EndGame();
  else if GameOperation =LOST then
    RenderLose();
  else if GameOperation =WON then
    RenderWin();
  else if GameOperation =TIMEUP then
    TimeUp();
  else if GameOperation =LEVELCOMPLETED then
    LevelCompleted();
  end if
```

Chapter 9: Case Studies–Creating New Algorithm Games

This chapter presents a walkthrough for the creation of a new algorithm game, showing all required steps, and how it is simplified by using the Algorithm Game Designer different components.

9.1 Game Creation

After the game designer requests the IDE to create a new Algorithm Game (Figure 9.1), the Solution Explorer window shows the initial Algorithm Game Project that have been created using the Algorithm Game Template. The Algorithm Game Project components are the Content project, Program class, BaseGame class, and Game Screens folders (Figure 9.2).

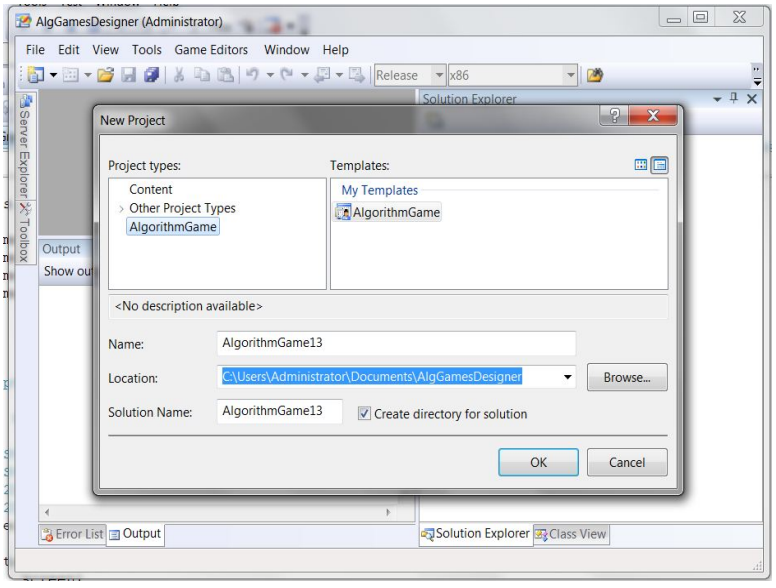

Figure 9.1: Creating New Algorithm Game Solution

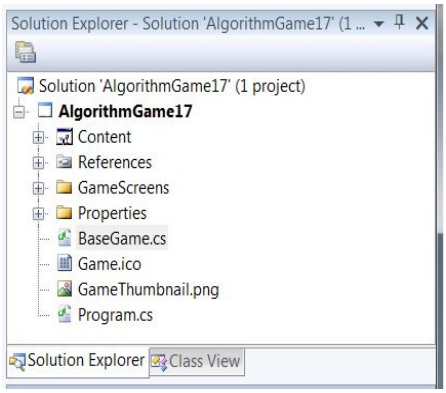

Figure 9.2: Initial Solution Items

If the game developer expands all items in the Algorithm Game Project, the initial contents of each item are as follows:

- Content project: has six folders including Effects, Models, Fonts, Sounds, Texture, and Xml (Figure 9.3). The first four folders contain the actual content files while the last folder contains the xml files that describe these content files and how they will be used in the game.
- References folder: shows the game default references, including a reference to SAVGEngine (Figure 9.4).
- GameScreens folder: includes the GamePlay class (Figure 9.4).

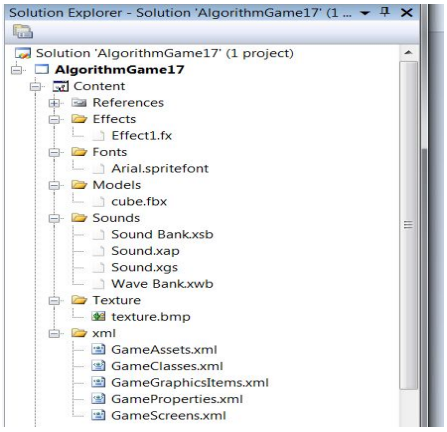

Figure 9.3: Initial Content Items

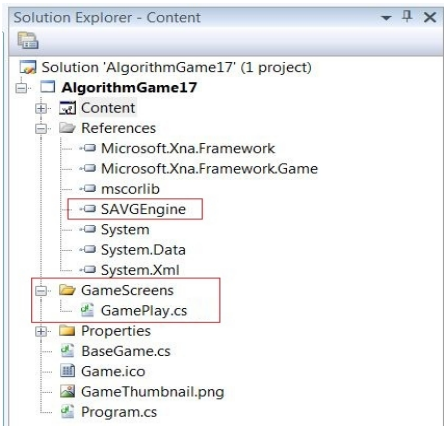

Figure 9.4: Initial Algorithm Game Items 1

9.2 Game Build and Debug

If the game developer double-clicks either the GamePlay, Program, or BaseGame classes, the code editor shows this class content. These three classes implement game general code, which the developer may modify according to the game specific requirements. When the developer builds the game, the build operation succeeds as shown in Figure 9.5.

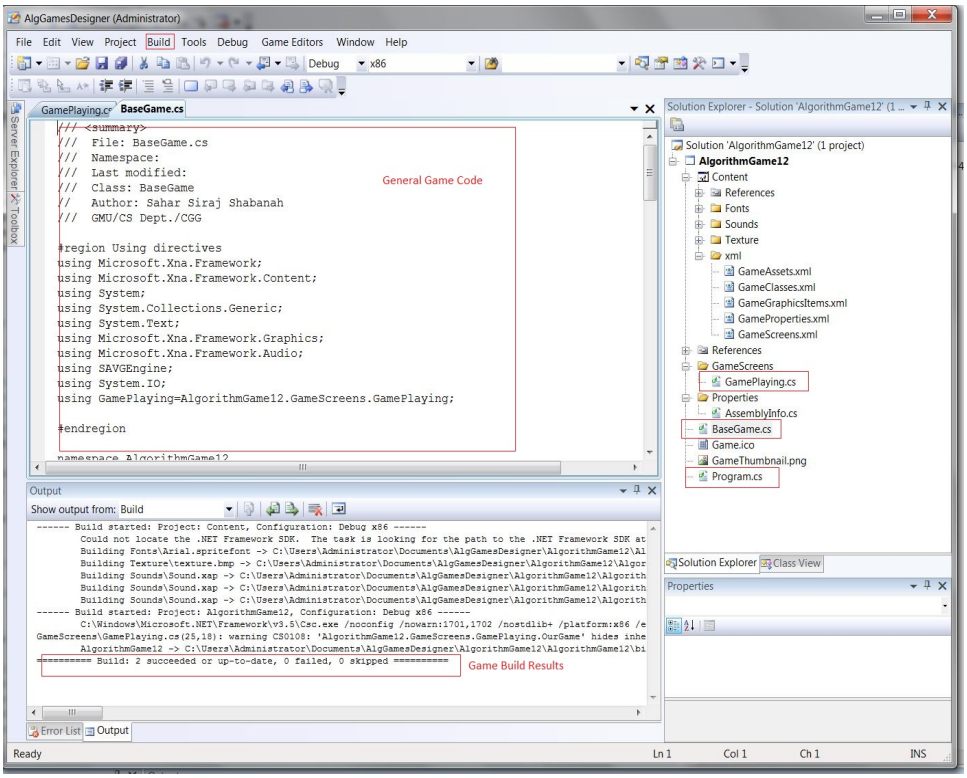

Figure 9.5: Initial Algorithm Game Items 2

Besides the GamePlay, BaseGame, and Program classes, the created game has fourteen default game screens, each screen showing the game state at a specific time as explained in the following:

1. Title Screen is displayed when the game is launched for first time. To leave this screen and start the game, the player must click the Enter Key (Figure 9.6).

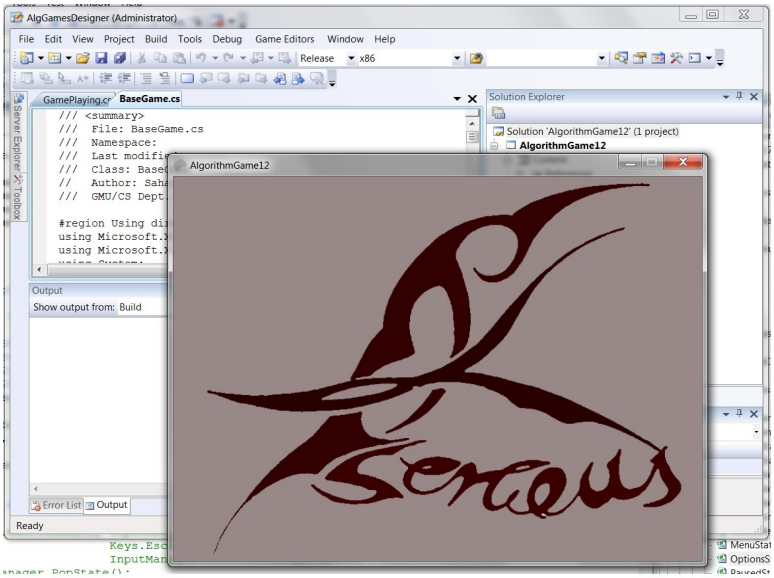

Figure 9.6: Default Algorithm Game Screens–Title screen

2. Player Name Screen appears either after the player clicks the Player Name button, or after the Title screen at the start of the game. To enter the Player Name after typing it, the Enter Key must be clicked (Figure 9.7).

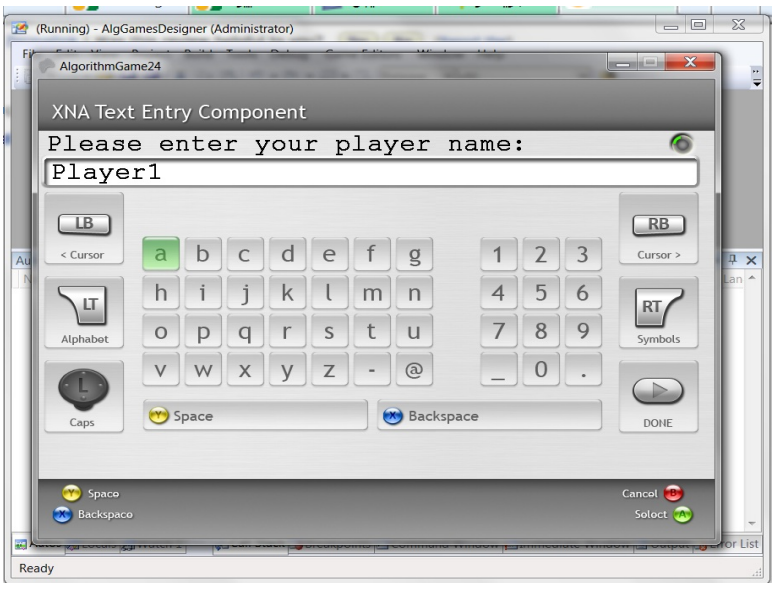

Figure 9.7: Default Algorithm Game Screens–Player Name Screen

3. Main Menu Screen appears when the player exits from several game screens, such as Title, Play, Player Name, High Scores, Player Report, Won, and Lost. The Main Menu screen displays five default buttons: Play Game, High Scores, Player Report, Game Demo, and Exit (Figure 9.8).

Figure 9.8: Default Algorithm Game Screens–Main Menu Screen

4. Play Screen is displayed if the player clicks Play Game button on Main Menu screen. It displays an empty screen with some HUD initial properties (Figure 9.9). It has six screens with which it is associated: Start Level, Lost Level, Won Level, Pause, Help, and Exit.

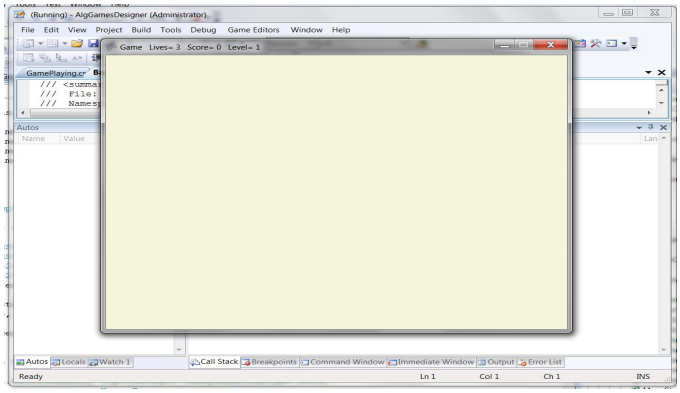

Figure 9.9: Default Algorithm Game Screens–Play Screen

5. Start Level Screen is displayed at the start of each level to show its number (Figure 9.10).

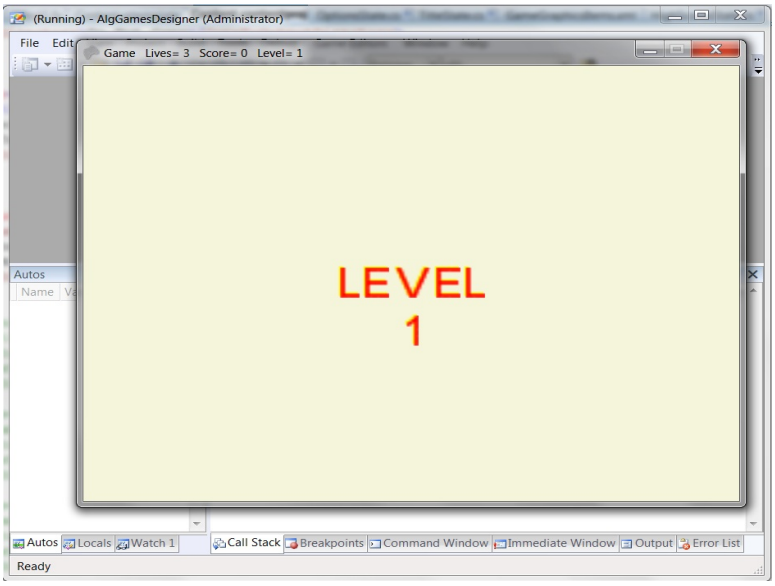

Figure 9.10: Default Algorithm Game Screens–Start Level Screen

6. Lost Level Screen is displayed after the player loses one game level (Figure 9.11).

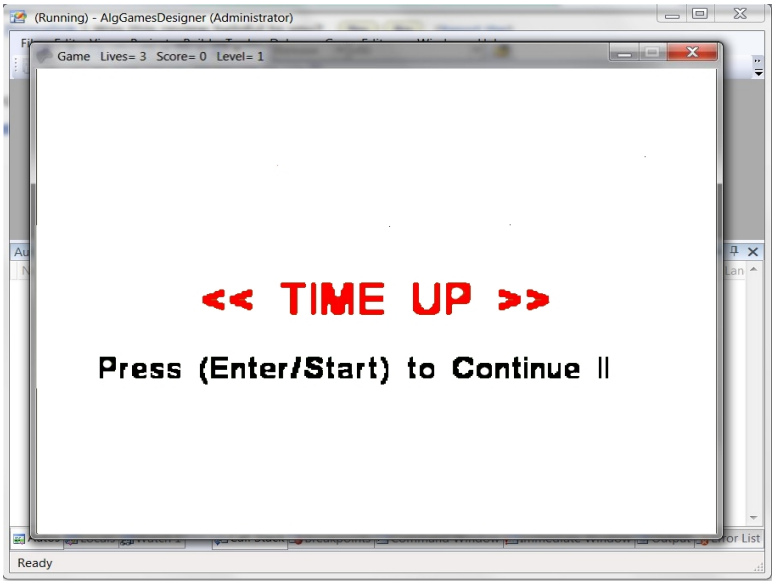

Figure 9.11: Default Algorithm Game Screens–Lost Level Screen

7. Won Level Screen is displayed after the player wins one game level (Figure 9.12).

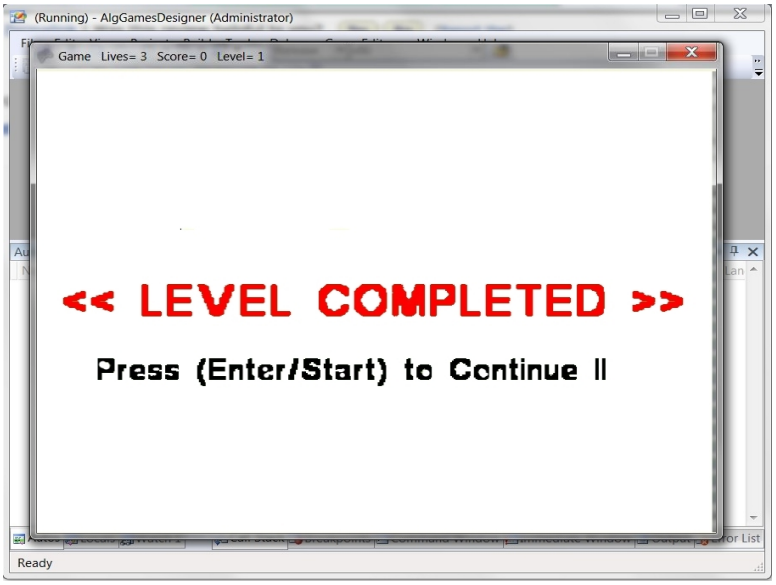

Figure 9.12: Default Algorithm Game Screens–Won Level Screen

8. Pause Screen appears after Pause key is clicked (Figure 9.13), the game is halted until Pause key is pressed again.

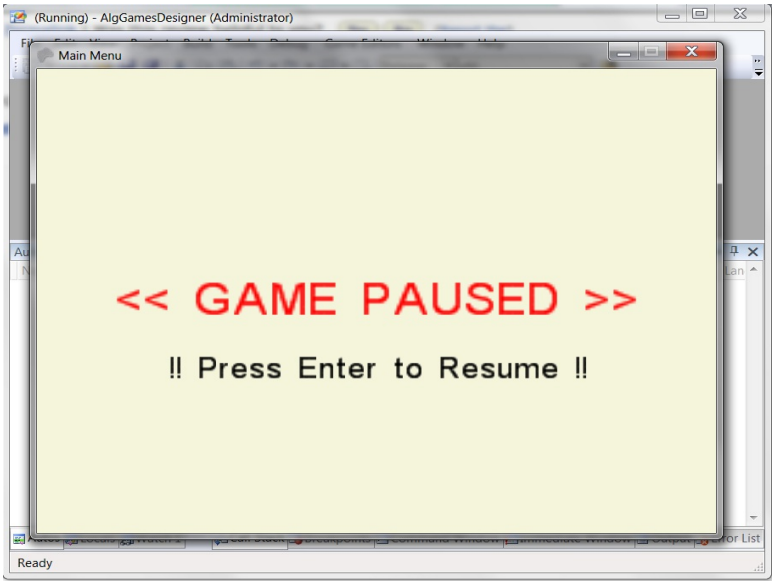

Figure 9.13: Default Algorithm Game Screens–Pause Screen

9. Exit Screen is displayed after the player clicks either the Escape key or the gamepad Return button. A confirmation question appears (Figure 9.14); if the player confirms the request, the game is exited and the Main Menu screen appears; otherwise, the game resumes.

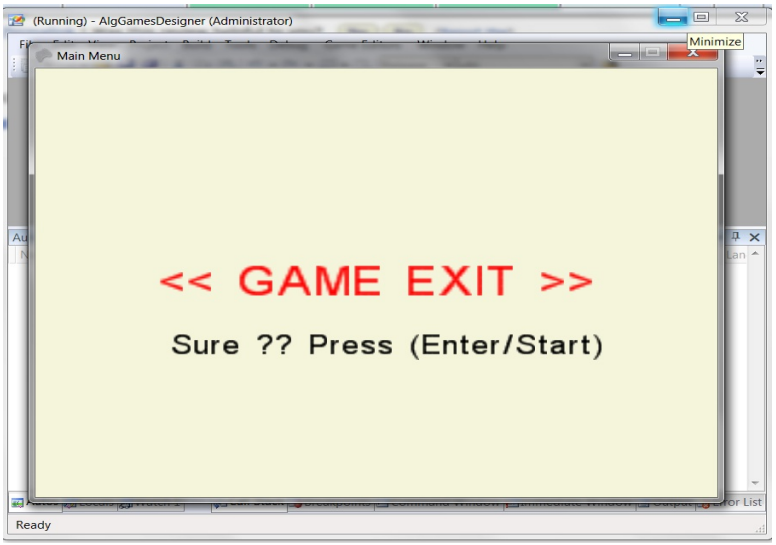

Figure 9.14: Default Algorithm Game Screens–Exit Screen

10. Help Screen is displayed when the player press F1 (Figure 9.15).

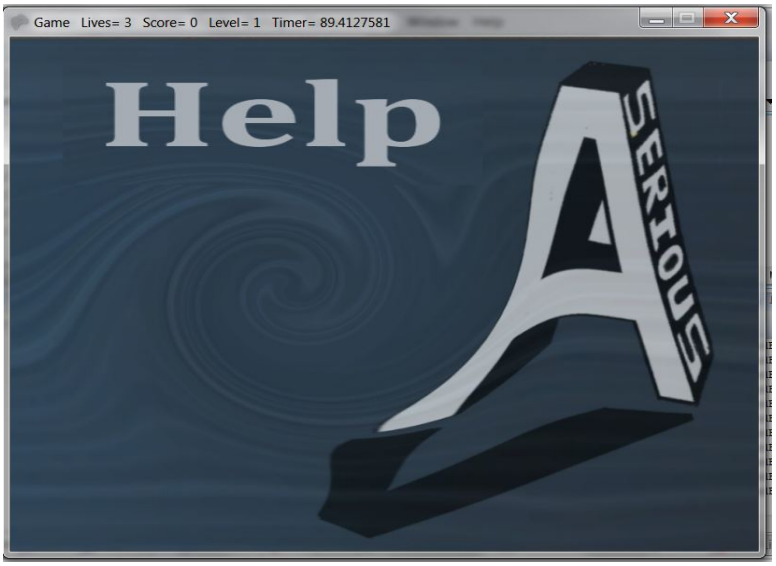

Figure 9.15: Default Algorithm Game Screens–Help Screen

11. Won Screen appears after the player wins the game (Figure 9.16).

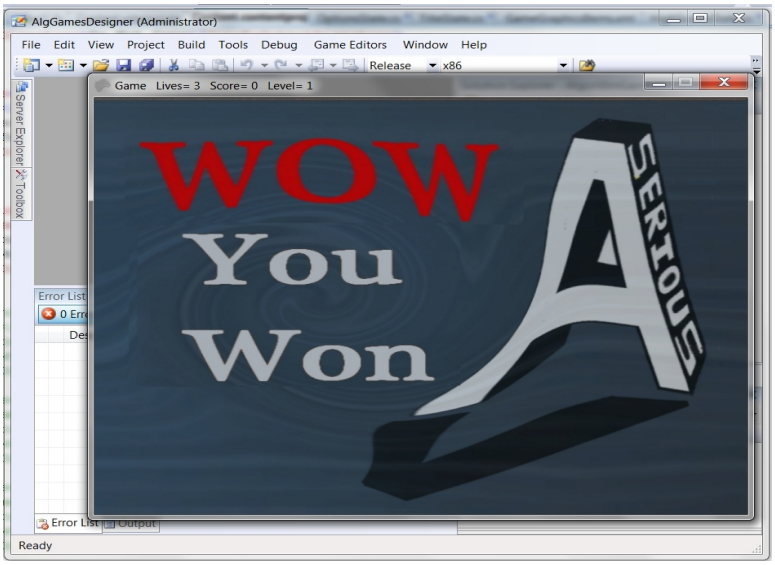

Figure 9.16: Default Algorithm Game Screens–Won Game Screen

12. High Scores Screen appears if the designer clicks the High Scores button on the Main Menu screen (Figure 9.17). To leave this screen and go back to Main Menu screen, the player must click the Enter Key.

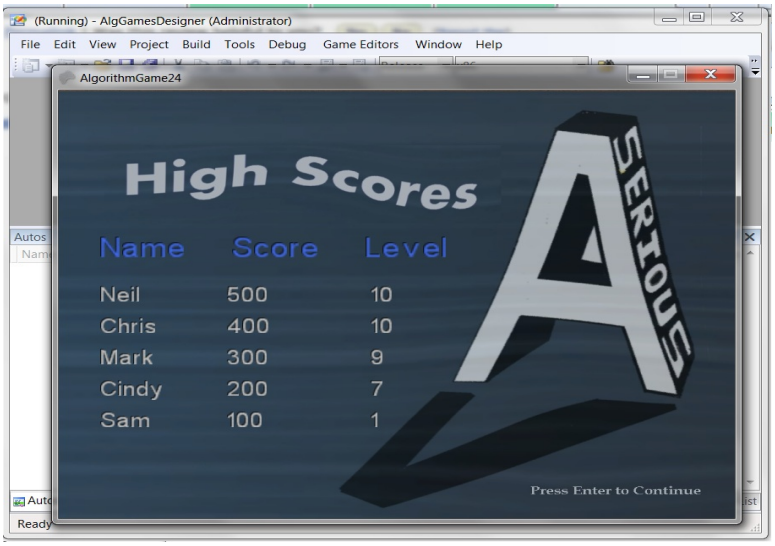

Figure 9.17: Default Algorithm Game Screens–High Scores Screen

13. Lost Screen appears after the player loses the game (Figure 9.18).

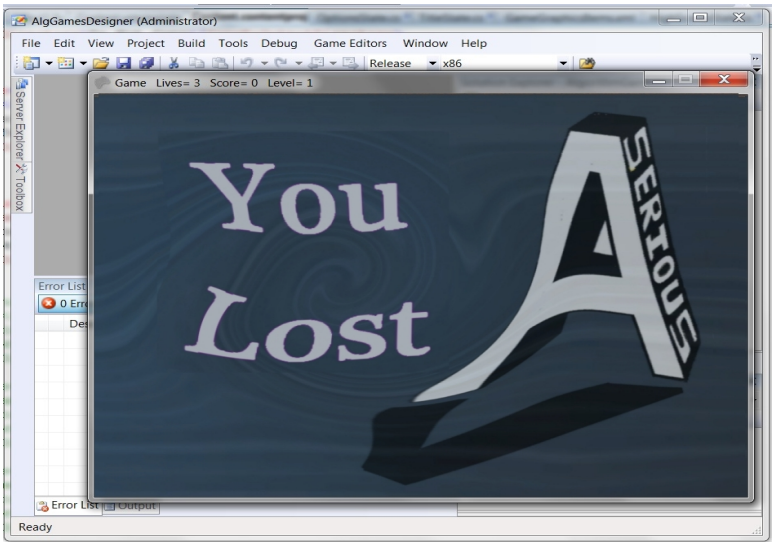

Figure 9.18: Default Algorithm Game Screens–Lost Game Screen

14. Game Demo Screen is displayed when the player clicks the Game Demo button on the Main Menu screen. It displays a demonstration of how the algorithm should be played (Figure 9.19).

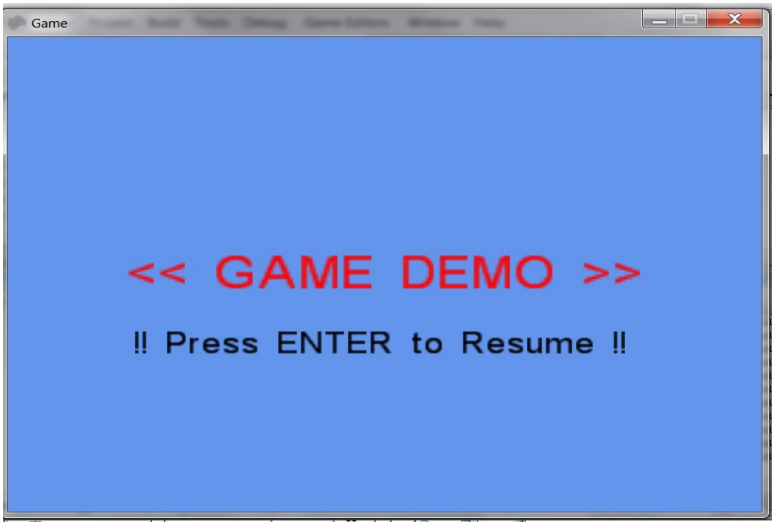

Figure 9.19: Default Algorithm Game Screens–Game Demo Screen

15. Player Report Screen appears if the designer clicks the Player Report button on the Main Menu screen (Figure 9.20). To leave this screen and go back to Main Menu screen, the player must click the Enter Key.

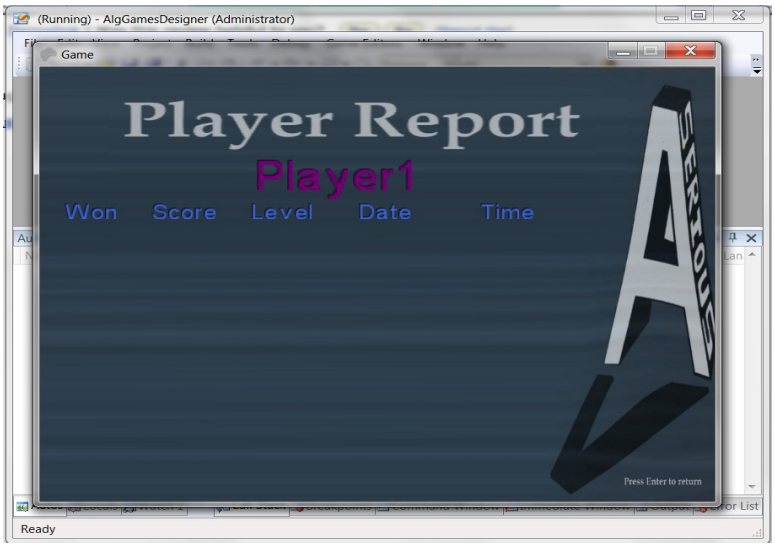

Figure 9.20: Default Algorithm Game Screens–Player Report Screen

16. Credits Screen is displayed after the player clicks the Exit button (Figure 9.21).

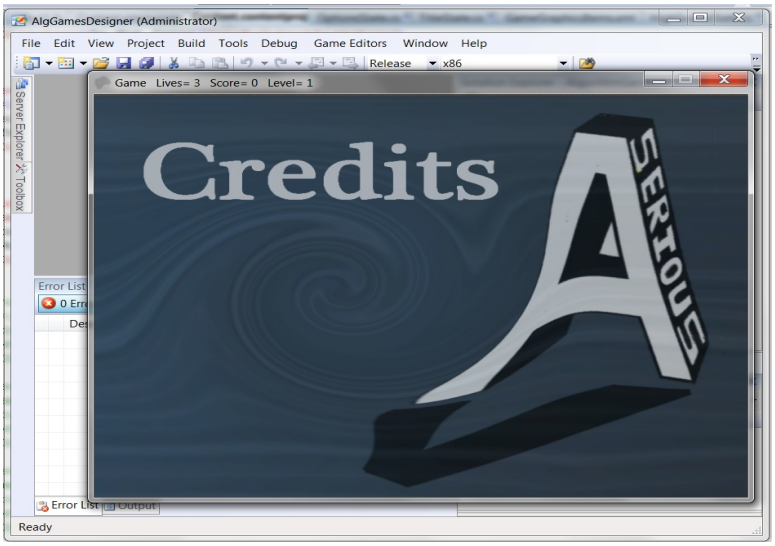

Figure 9.21: Default Algorithm Game Screens–Credits Screen

9.3 Game Configuration and Content

To modify the game initial configurations and build the game content, the developer must follow sequential steps:

First: the developer needs to set up the Game Properties, such as the name, number of levels, and number of lives using the Game Properties Editor (Figure 9.22).

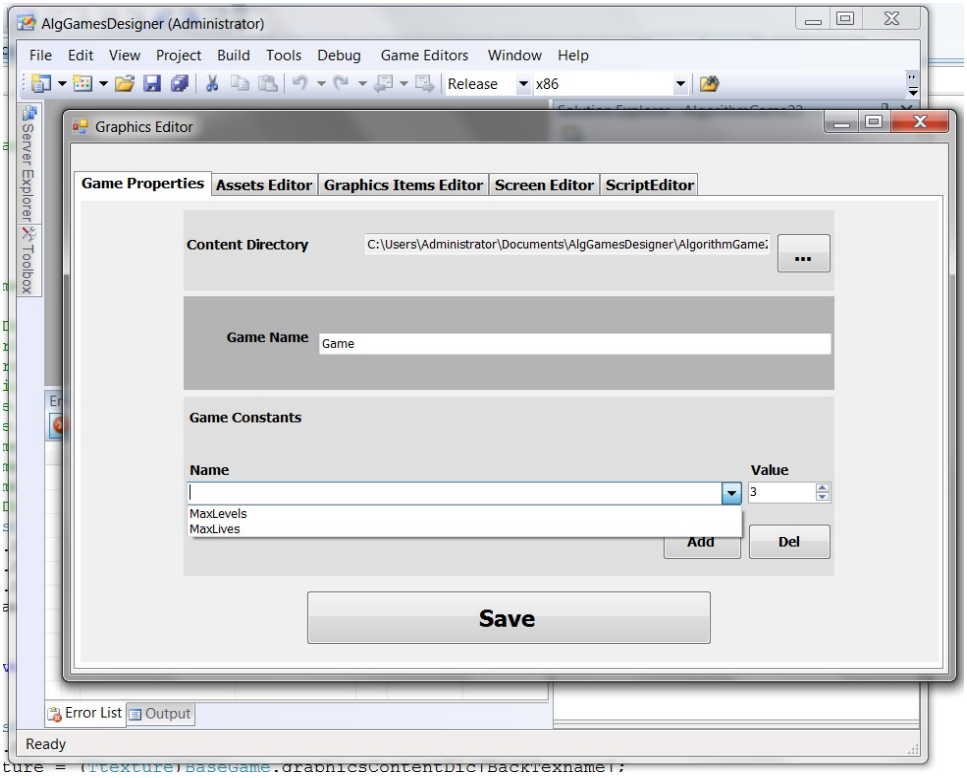

Figure 9.22: Properties Editor–Default Items

Second: the developer must add all Game Assets (font, texture, model, sound, and effect files) using the Game Assets Editor. The developer can choose an asset file from the included default assets files list (Figure 9.23), or add a new one.

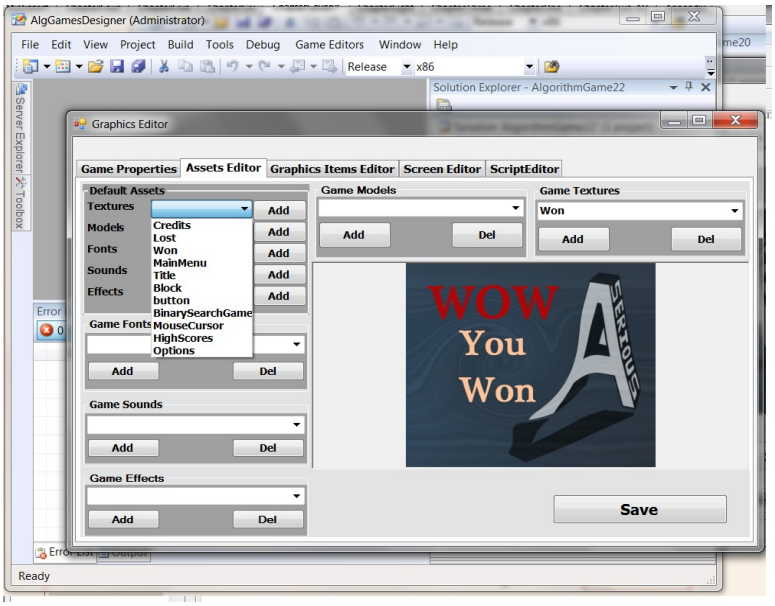

Figure 9.23: Assets Editor–Adding Default Texture

However, the new asset file must be added to the content project first (Figure 9.24), then loaded into the game using the Assets Editor (Figure 9.25).

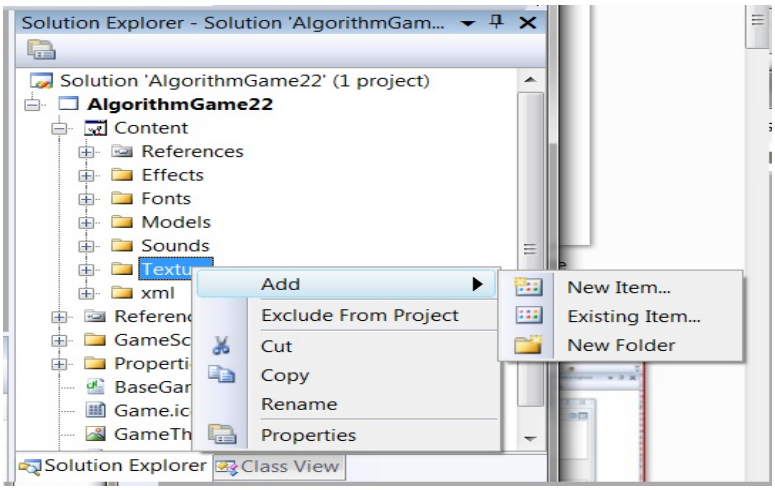

Figure 9.24: Content Project–Add New Texture

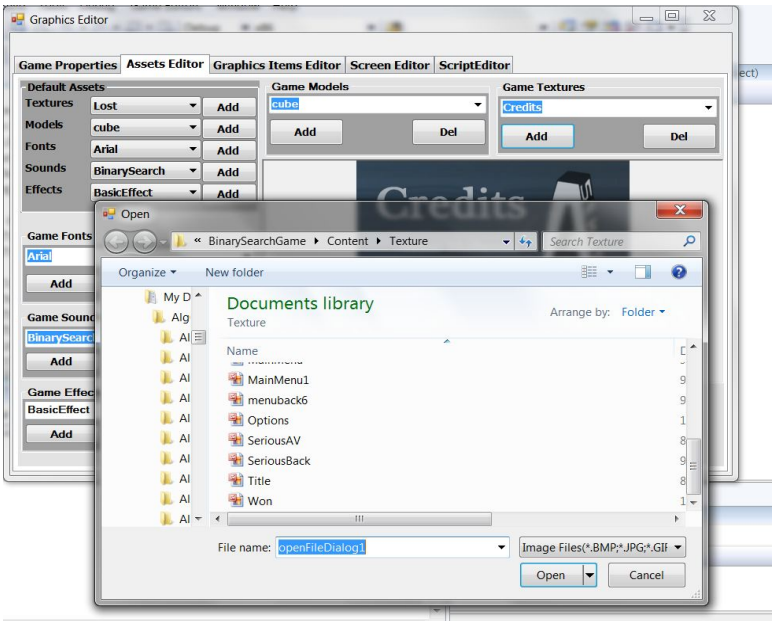

Figure 9.25: Assets Editor–Add New Texture

Third: the developer must use the Classes Editor to add the names of default, or newly created Game Classes into the game (Figure 9.26).

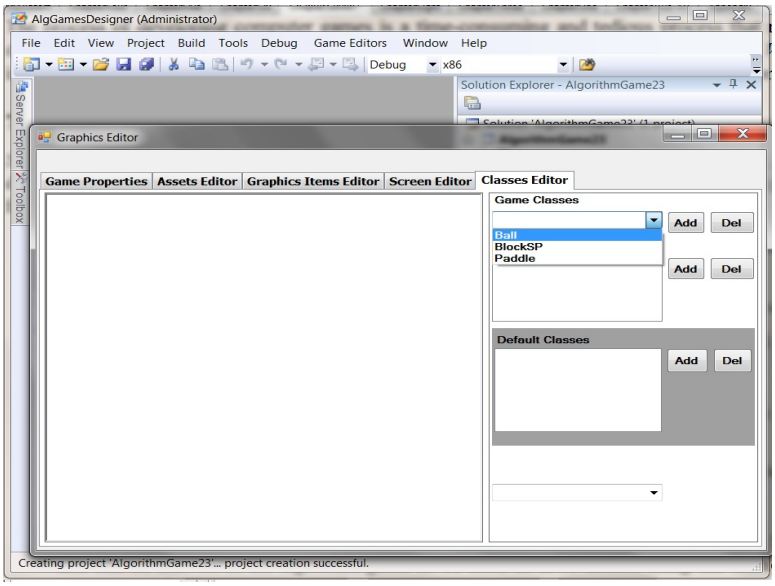

Figure 9.26: Classes Editor–Default Classes

Fourth: the developer must add all Game Graphic Items into the game using the Game Graphic Items Editor. To add a default item, the designer selects it from the default graphic items list (Figure 9.27).

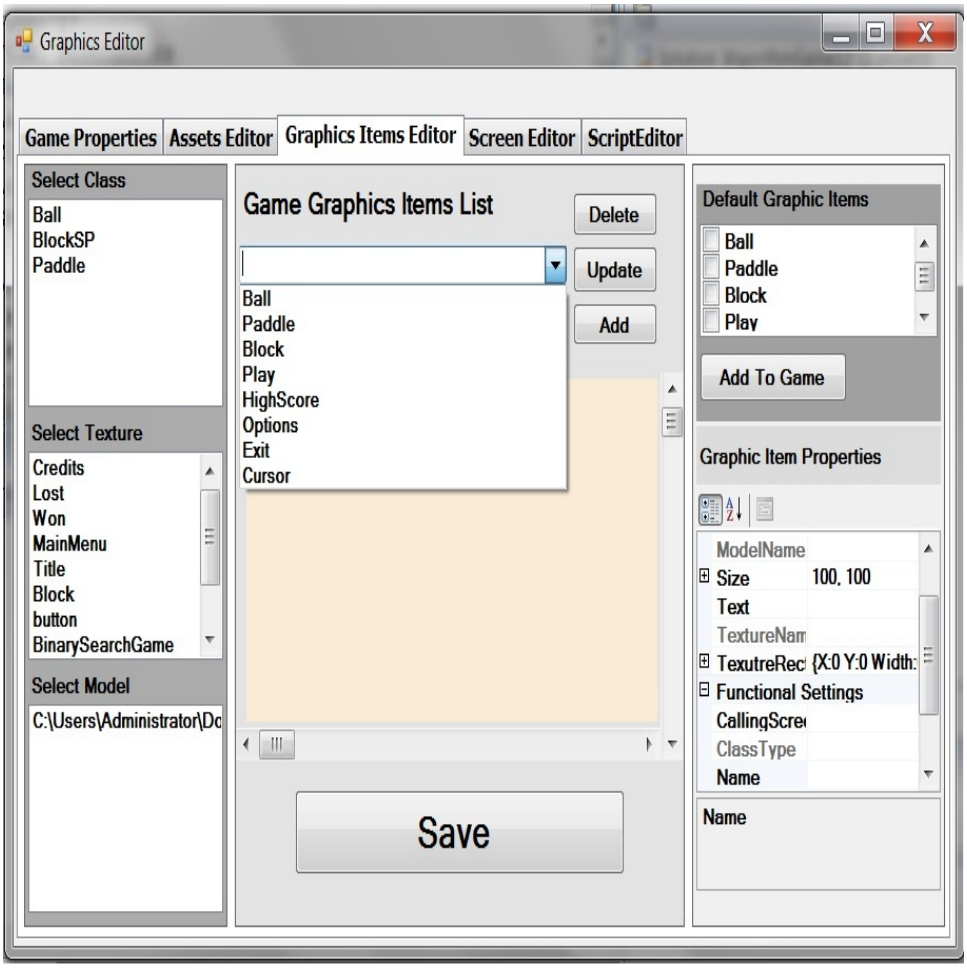

Figure 9.27: Graphic Items Editor–Add Default Item

For adding a new graphic item the designer selects the class that explains the item behavior and the model and/or texture that determines its appearance. The properties of every graphic item can be set or changed as required by using its properties pane (Figure 9.28).

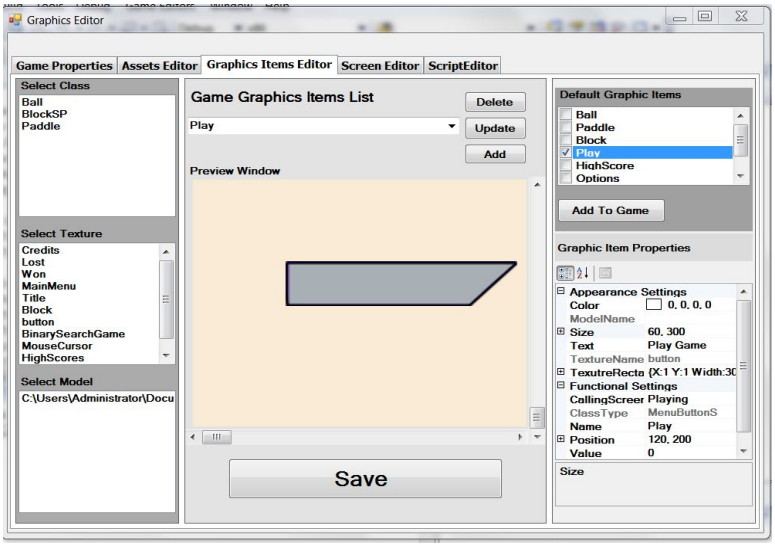

Figure 9.28: Graphic Items Editor–Add New Item

Fifth: the developer can now design the Game Screens using the Game Screens Editor by either choosing from the default screens list (Figure 9.29) or adding a new screen.

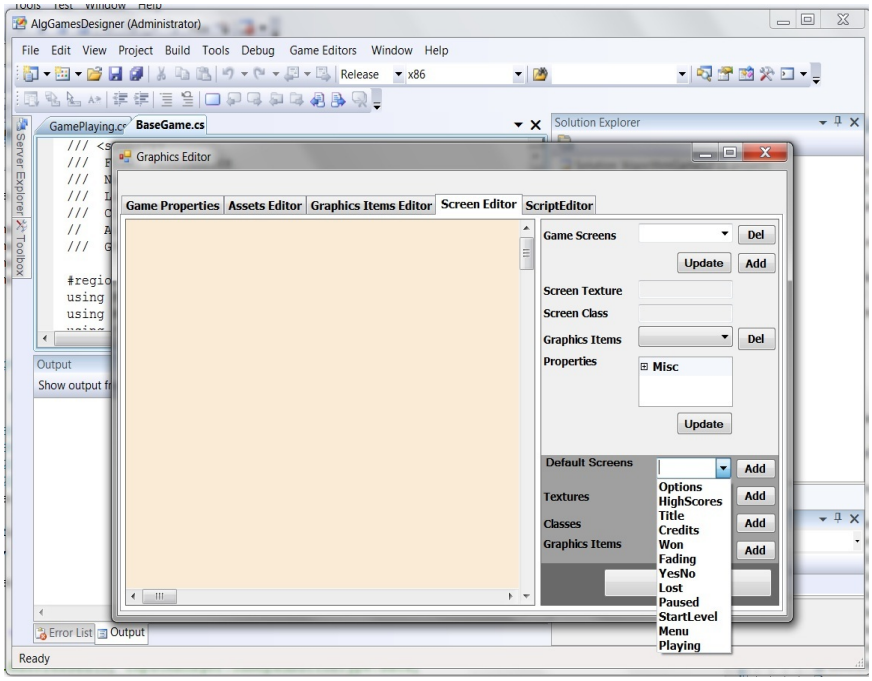

Figure 9.29: Screens Editor–Default Screens

To add a new screen, the developer must define its class name, background texture, and all graphic items (Figure 9.30).

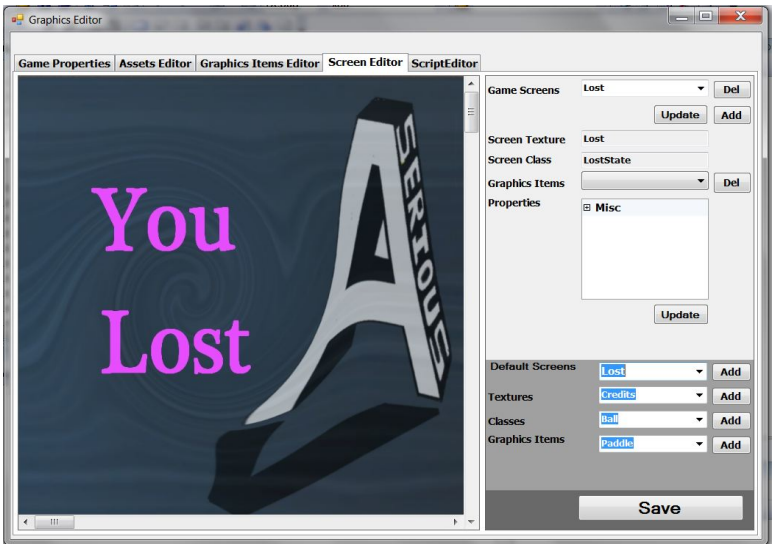

Figure 9.30: Screens Editor–Add New Screen

The developer can customize a graphic item (Figure 9.31) by changing its position on the screen and/or its size using either the mouse or its properties pane (Figure 9.32).

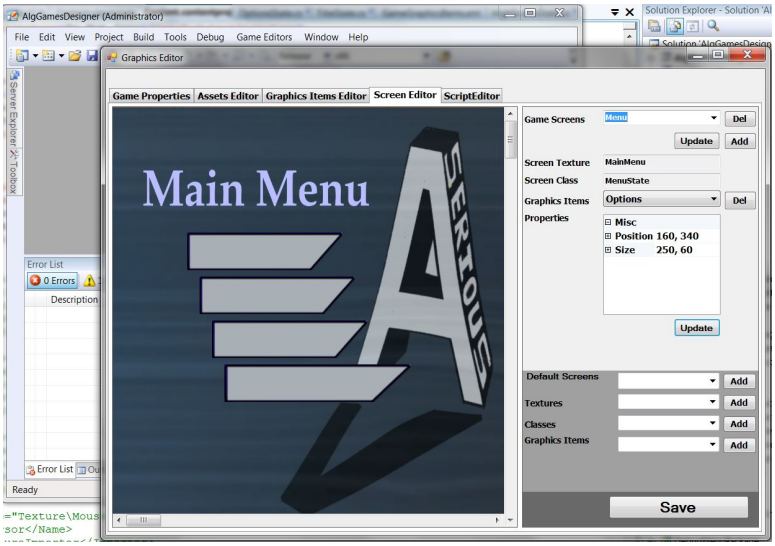

Figure 9.31: Screens Editor–Initial Buttons Position and Size

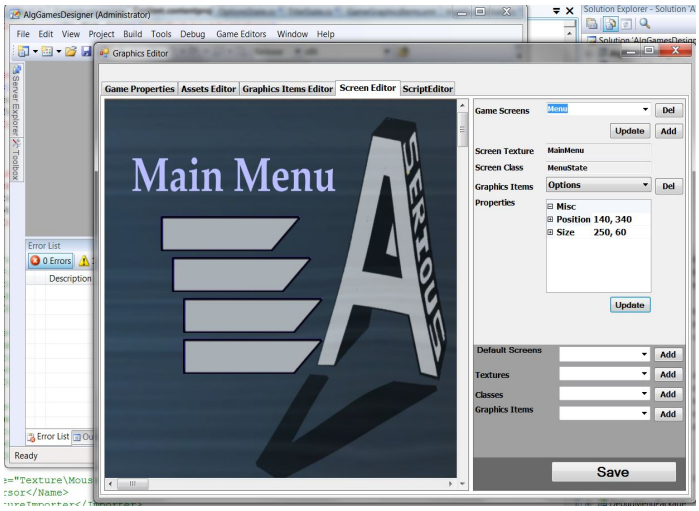

Figure 9.32: Screens Editor–Changing Button Position and Size

9.4 Game-Play

After designing all visual aspects of the game, the developer needs to implement the required game behavior. To simplify the implementation process, the GamePlay and BaseGame classes have definitions (headers) with empty bodies for most of the required methods that need to be completed by the designer to simulate the required game behavior.

```
public class BaseGame : SAVGEngine.BaseGame
 {
    private void SetUpGameProperties(){      }
    private static void SetUpGameGraphicsItems(){      }
    private void SetUpGameScreens() {      }
 }
```

```
public class GamePlay : SAVGEngine.PlayingState
{
  protected override void SetupGame(){      }
  protected override void StartGame(){      }
  protected override void GetPlayerInput(GameTime gameTime){      }
  protected override void StartLevel(){      }
  protected override void PlayLevel(){      }
  protected override void RenderLevel(){      }
}
```

9.5 Binary Search Game Creation

For illustration, this section shows the creation of the Binary Search game that has been created using the Algorithm Game Designer.

9.5.1 Game Configuration and Content

The Binary Search game prototype, which has been described in Section 6.1, is summarized in the following.

Game Properties

The Binary Search game properties are as follows:

- Maximum Level Number=3.
- Maximum Lives Number=3.
- Game Name=Binary Search.

These game properties entered into the game using the Game Properties Editor as shown in Figure 9.33.

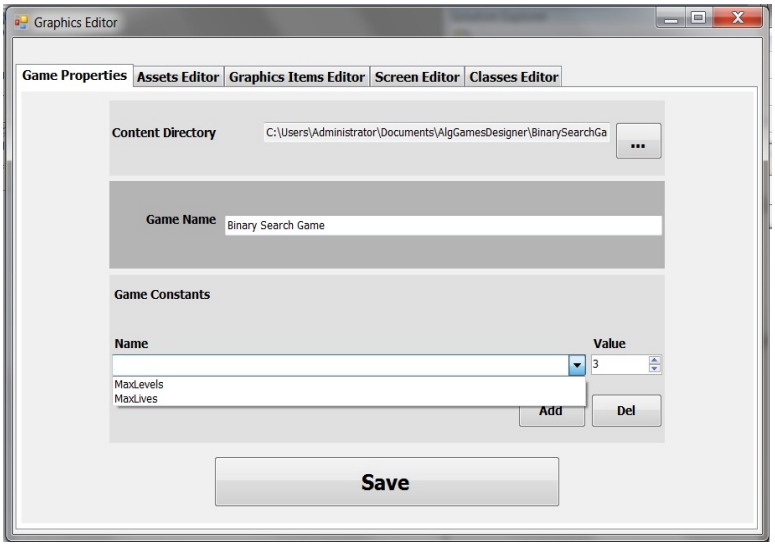

Figure 9.33: Binary Search Game–Add Properties

Game Assets

The Binary Search game assets are as follows:

- Textures: Ball, Paddle, and Block.
- Sounds: LostBall, HitBall, LostLive, LostGame, and WonGame, which are defined in the BinarySearch.xap sound file.
- Fonts: Ariel.spritefont.

These game assets were entered into the game using the Game Assets Editor as shown in Figure 9.34.

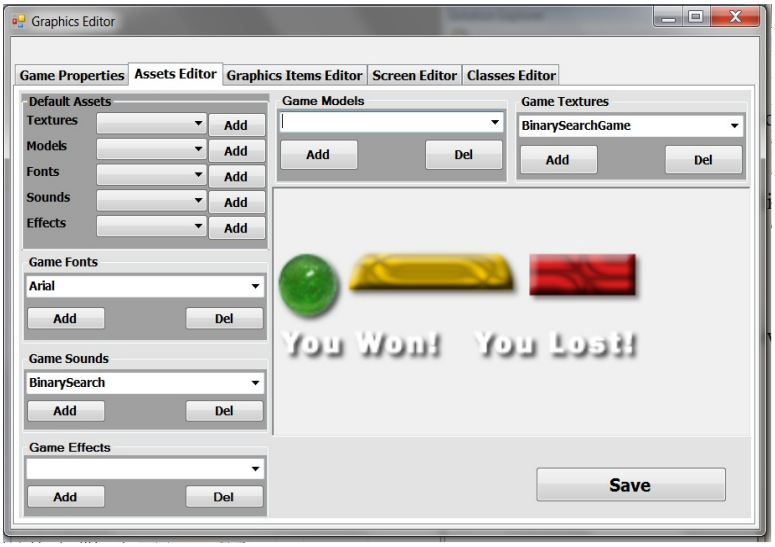

Figure 9.34: Binary Search Game–Add Assets Files

Game Classes

The Binary Search game classes are as follows:

- Ball class is a default class.
- Paddle class is a default class.
- BlockSp class is a default class.

These game classes were entered into the game using the Game Classes Editor as shown in Figure 9.35.

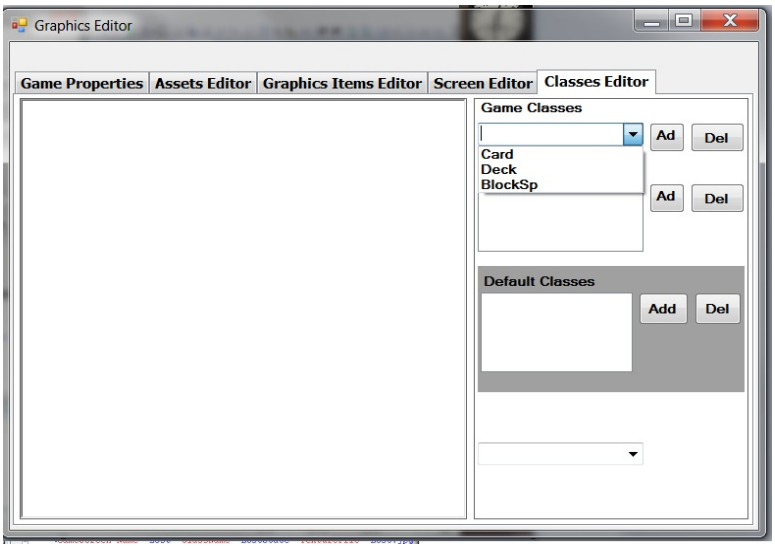

Figure 9.35: Binary Search Game–Enter Classes

Game Graphic Items

The Binary Search game graphics items are as follows:

- Ball has a texture (Ball) that determines its appearance, class (Ball) that defines its behavior, and attributes: position, speed, repetition, and size.
- Paddle has a texture (Paddle) that determines its appearance, class (Paddle) that defines its behavior, and attributes: position, speed, repetition, and size.
- Block has a texture (Block) that determines its appearance, class (BlockSp) that defines its behavior, and attributes: position, repetition, and size.

These items were created and added to the game using the Game Graphic Items Editor as shown in Figures 9.36, 9.37, and 9.38.

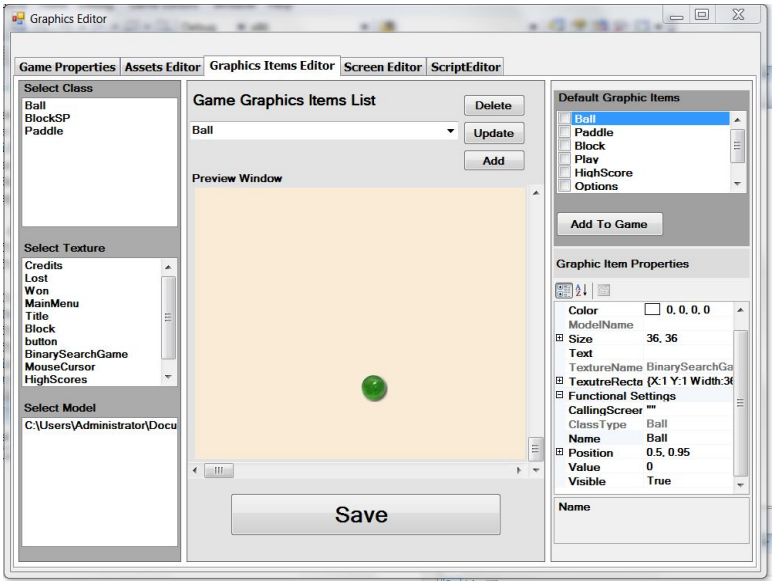

Figure 9.36: Binary Search Game–Graphic Item (Ball) Design

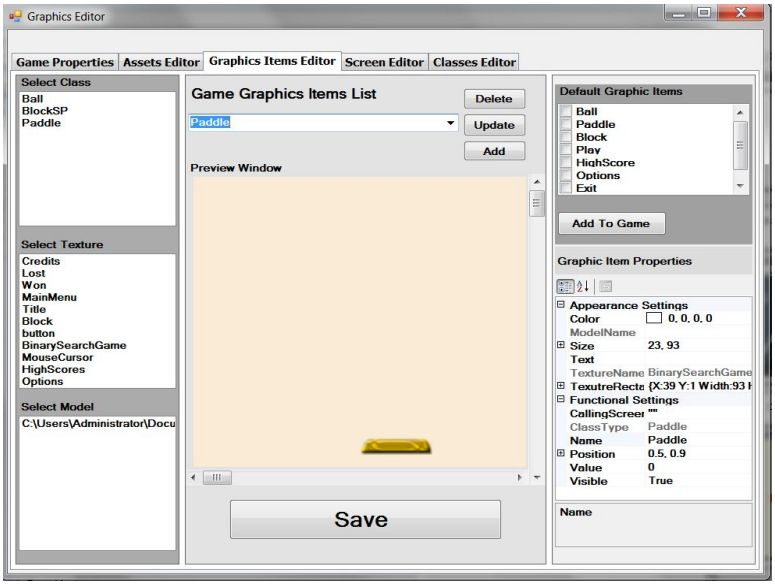

Figure 9.37: Binary Search Game–Graphic Item (Paddle) Design

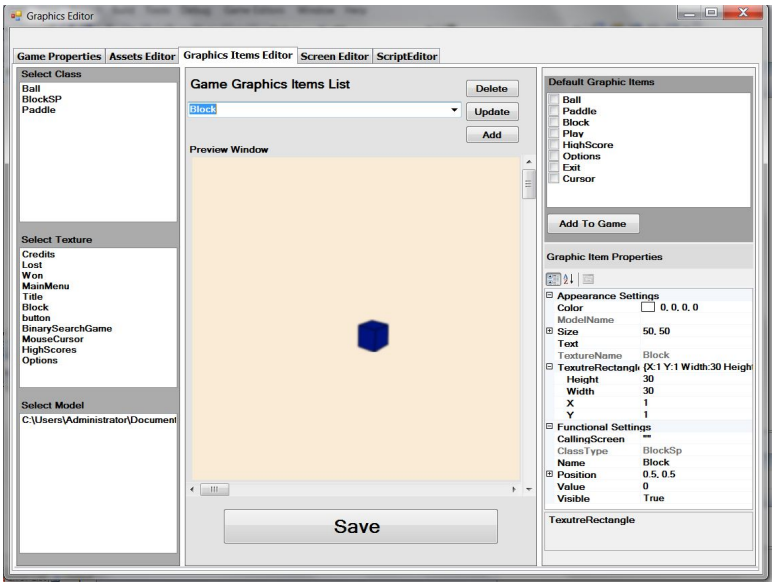

Figure 9.38: Binary Search Game–Graphic Item (Block) Design

Game Screens

The Binary Search game screens are as follows:

- • All default Algorithm Game Screens, which have been used in the game without modification except for the Play screen. • The play screen that has been customized by adding three graphic items: Ball, Paddle, and Block.

The Game Screens Editor has been used to add all game screens as shown in Figure 9.39 and to customize the Play screen as shown in Figure 9.40.

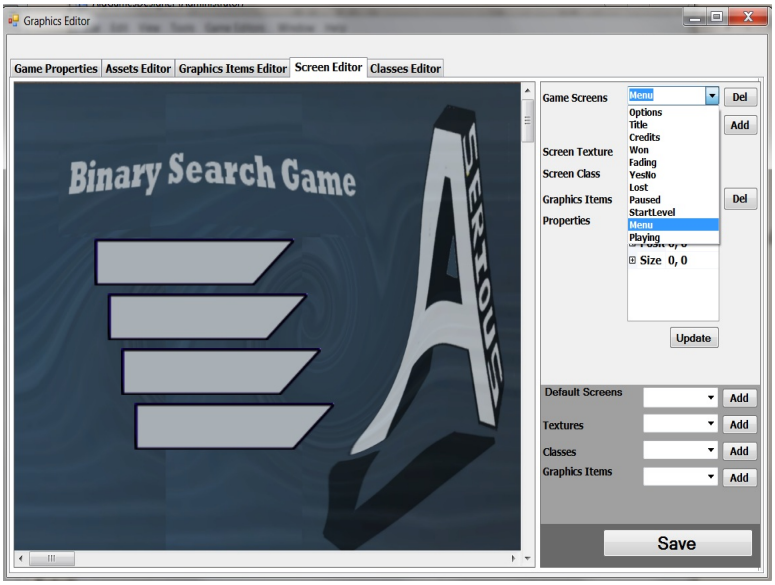

Figure 9.39: Binary Search Game–All Game Screens

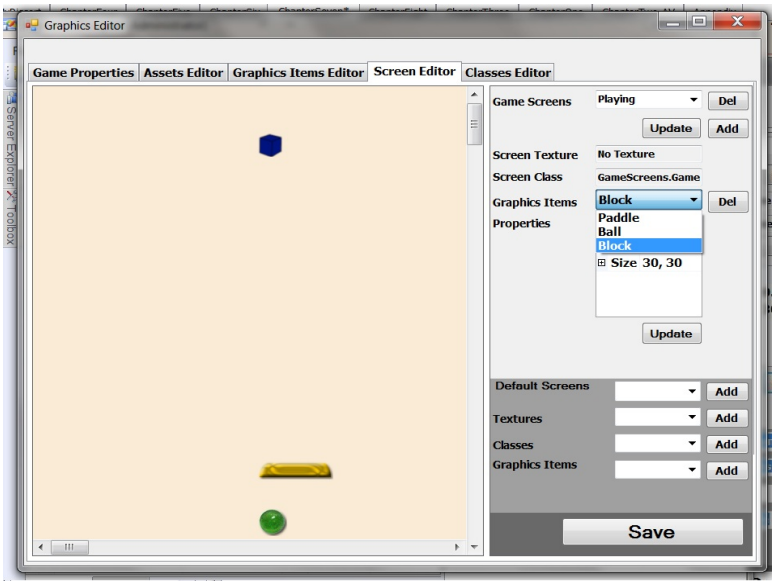

Figure 9.40: Binary Search Game–Play Screen Design

9.5.2 Game-Play

The BaseGame and GamePlay classes, which were provided with the Algorithm Game Template, must be implemented to create the game-play.

BaseGame Implementation

In the BaseGame class, three methods are needed to be implemented:

1. `SetUpGameProperties`(); the game uses all default game properties, so this method is not implemented.

2. `SetUpGameScreens`(); the game uses all default game screens, so this method is not implemented.

3. `SetGameGraphicsItems`(); this method initializes all game graphic items to be used later in the GamePlay class. As explained in Section 8.6.2, all graphic items have been stored in the GameGraphicsItemsDic, so they can be accessed using their names as shown in the following code snippet:

```
private void SetGameGraphicsItems()
{
   //intilaze the Ball, Paddle, and Blocks
   bBall = (Ball)base.GameGraphicsItemsDic["Ball"];
   pPaddle = (Paddle)base.GameGraphicsItemsDic["Paddle"];
   Block = (BlockSp)base.GameGraphicsItemsDic["Block"];
}
```

GamePlay Implementation

In the GamePlay class six methods need implementation, as explained in the following:

1. `SetupGame`(); the game uses the default settings, so there is no need to implement this method.

2. `StartGame`(); the game uses the default method, so there is no need to implement this method.

3. `StartLevel`(); this method starts one game level by calculating the number of the blocks in the array, search number, Mid, Low, High values, Ball, and Paddle start positions as the next code snippet shows:

```
protected override void StartLevel()
{
  //intialize blocks number
  NumOfColumns = Math.Min(20,(CurrentLevel+1)*rRandom.Next(3,7));
  //intialize Search number as random
  SearchNum = rRandom.Next(NumOfColumns + 2);
  Mid = Low = 0;//intialize Mid and Low
  High = NumOfColumns - 1;//intialize High
  Index = -1;  // intialize  search Index
  BlockInit();  // intialize Block array
  bBall.StartNew(new Vector2(0.525f, 0.9f));// intialize Ball
  pPaddle.StartNew(new Vector2(0.5f, 0.95f)); // intialize Paddle
}
```

4. `RenderLevel`(); this method draws a new game level on the screen by rendering Ball, Paddle, and Blocks array as the next code snippet shows:

```
protected override  void RenderLevel()
{
  pPaddle.Render(OurGame.SpriteBatch);//draw Paddle
  bBall.Render(OurGame.SpriteBatch);//draw Ball
  RenderBlocks(); //draw Blocks array
}
```

5. `GetPlayerInput`(); this method registers the player input and responds accordingly, as shown in the following code snippet:

```
protected override void GetPlayerInput(GameTime gameTime)
{
  bBall.LeftRightHit(gameTime,Input);\\check the Ball movement
  pPaddle.LeftRightPush(gameTime, Input); \\check the Paddle movement
}
```

6. `PlayLevel`(); this method implements the game-play by following the binary search algorithm steps as shown in next code snippet:

```
protected override void PlayLevel()
{
  for (int x=0; x<NumOfColumns; x++)
  {
   if (Low>High) //end loop and not found search
   {
     Index = -1;
     PlayLevelState= "LevelCompleted";
   } //not found
   Mid = (Low+High)/2; //calculate mid;
   if(Blocks[x].Visible) //if hitted block is visible
   {
    if(OurGame.bBall.Intersects(Blocks[x])) //Ball Hits Block?
{
      Sound.Play("BallHit"); //yes
      bBall.StartNew(newVector2(pPaddle.Position.X, 0.95f));
      if ((x)!= Mid)//doesnot hit Mid
         if (RemainingLives <= 0) PlayLevelState = "Lost";
         else{
               RemainingLives--;
               Sound.Play("BallLost");
               }
      else {
             Sound.Play("BlockKill");//hit Mid
             TotalScore++;//add score
             Blocks[x].Visible = false;//show pivot value
             //check for Search Number
             if (Blocks[x].Value > SearchNum) //search> mid
                  High = Mid - 1; //array[0-->mid]
             else if (Blocks[x].Value < SearchNum)//serach<mid
                  Low = Mid + 1; //array[mid--> max]
              else{
                    Index = Mid; //found search
                    PlayLevelState = "LevelCompleted";
                   }
           }// hit Mid
        } //Ball Hits Block
    } //if hitted block is visible
  }//for
}// method
```

9.5.3 Binary Search Game Screen Shots

1- Title Screen

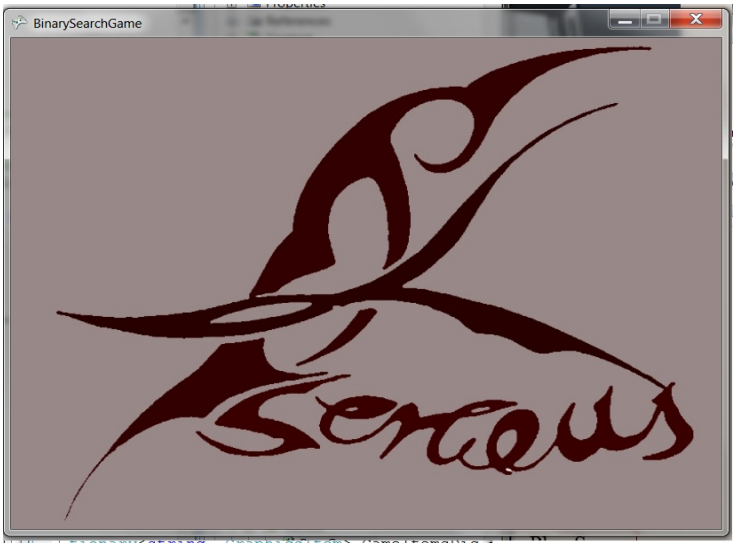

Figure 9.41: Binary Search Game–Title Screen

2- Player Name Screen

Figure 9.42: Binary Search Game–Enter Player Name

3- Main Menu Screen

Figure 9.43: Binary Search Game–Main Menu Screen

4- Start Level Screen

Figure 9.44 shows the first level of the game, including six blocks; the HUD items are Search Number=4, Lives=3, Level=1, and Score=0.

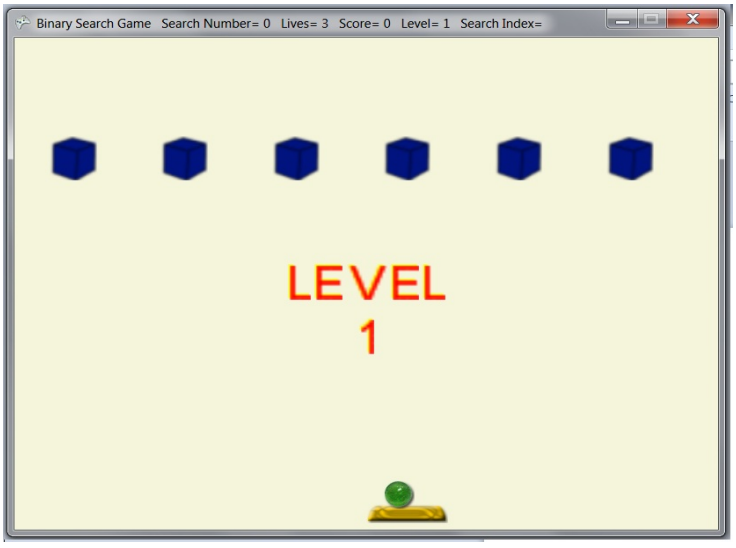

Figure 9.44: Binary Search Game–Play Screen (Level 1)

Figure 9.45 shows the second level of the game; as it appears, the number of blocks was increased to 12. The HUD items are Search Number=1, Lives=2, Level=2, and Score=3.

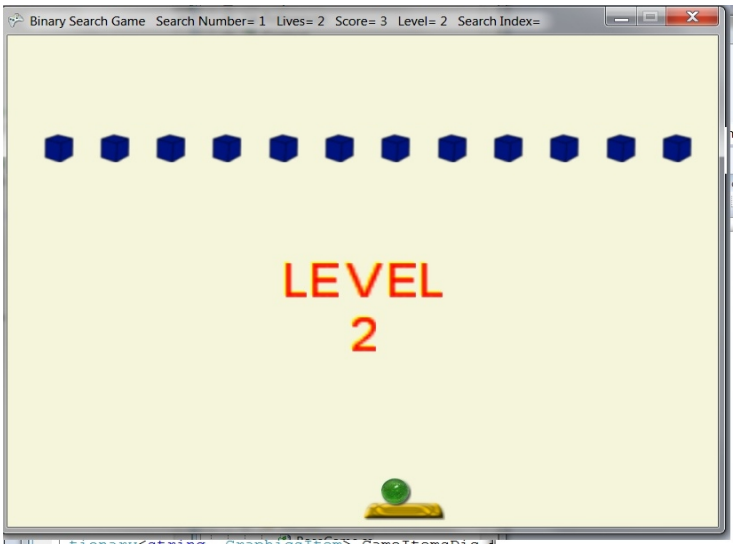

Figure 9.45: Binary Search Game–Play Screen (Level 2)

Figure 9.46 shows the third level of the game with more blocks 15; the HUD items are Search Number=4, Lives=1, Level=3, and Score=6.

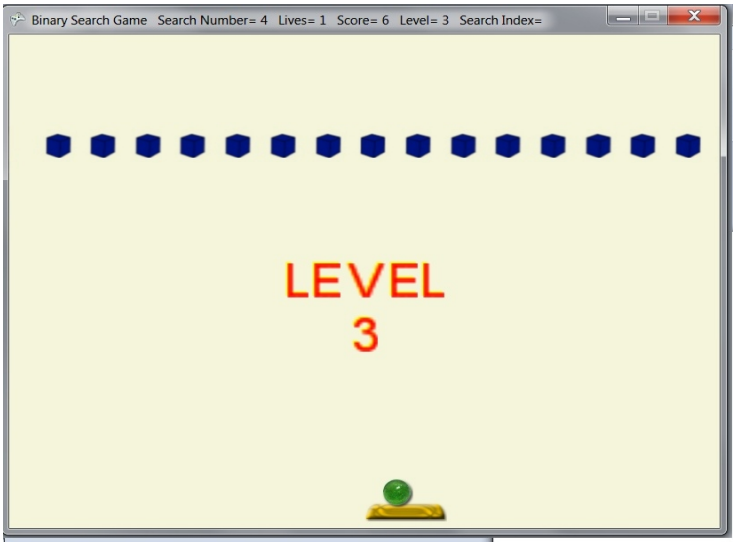

Figure 9.46: Binary Search Game–Play Screen (Level 3)

5- Play Screen

- Figure 9.47 shows the Play screen with the middle block value shown after being hit by the player using the Ball and the Paddle.

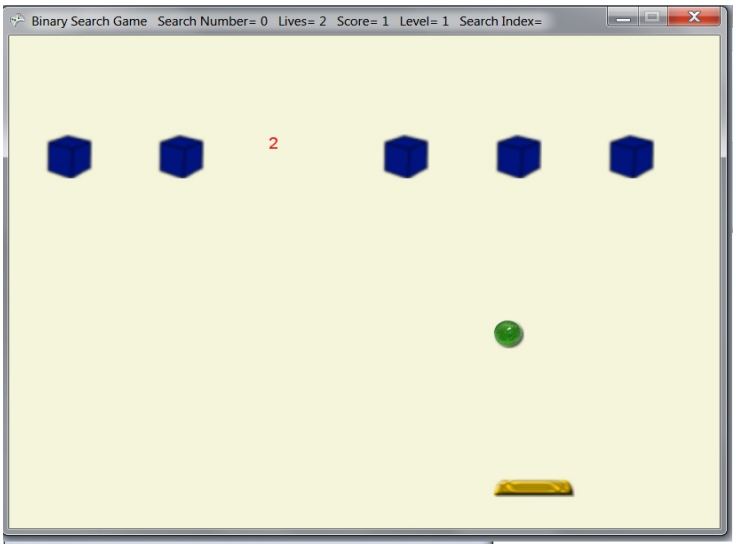

Figure 9.47: Binary Search Game–Play Screen 1

- Figure 9.48 shows another screen shot of the game, the HUD items are Search Number=16, Score=7, and Level=2. The progress of the game is as follows:
  - When the player hits the middle block (value=7) < 16 (Search Number), s/he goes left,
  - then when the player hits middle block (value=11) < 16 (Search Number), s/he goes left,
  - then when the player hits middle block (value=13) < 16 (Search Number), s/he goes left.

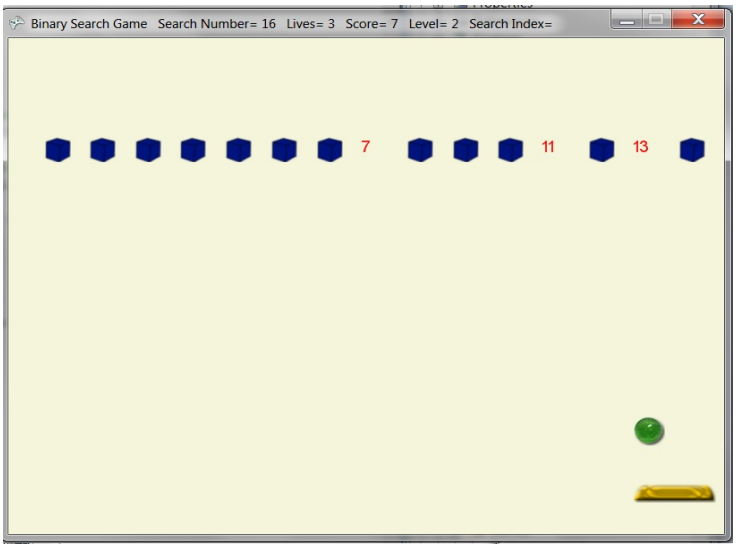

Figure 9.48: Binary Search Game–Play Screen 2

6- Pause Screen

Figure 9.49: Binary Search Game–Pause Screen

7- Exit Screen

Figure 9.50: Binary Search Game–Exit Screen

8- Lost Level Screen

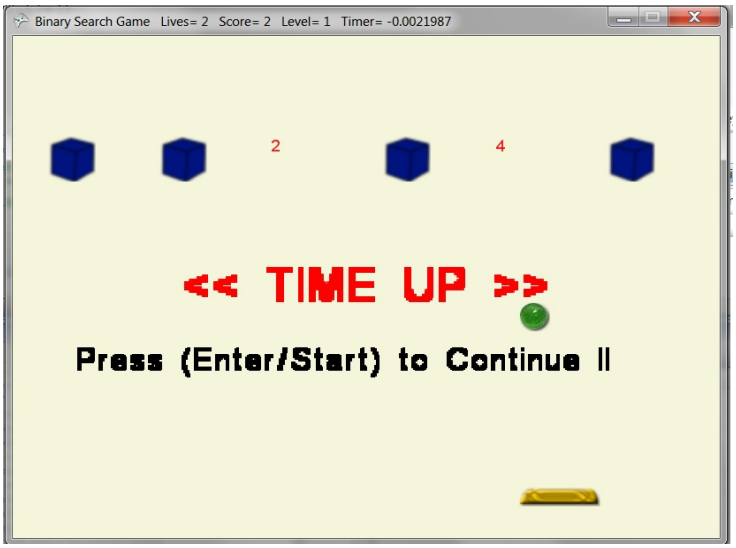

Figure 9.51: Binary Search Game–Lost Level Screen

9- Won Level Screen

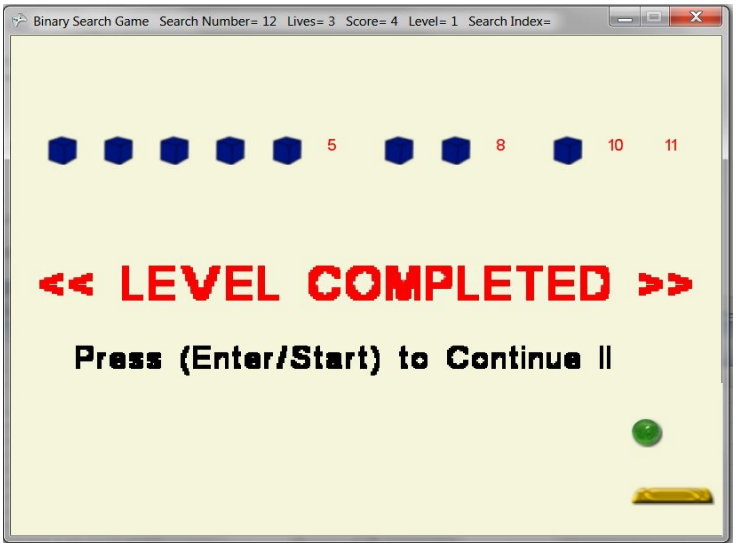

Figure 9.52: Binary Search Game–Won Level Screen

10- Won Game Screen

Figure 9.53 shows the HUD items, which are Score=12, Level=3, and Lives=1.

Figure 9.53: Binary Search Game–Won Game Screen

11- Lost Game Screen

Figure 9.54 shows the HUD items, which are Search Number =0, Lives=0, and Score=1.

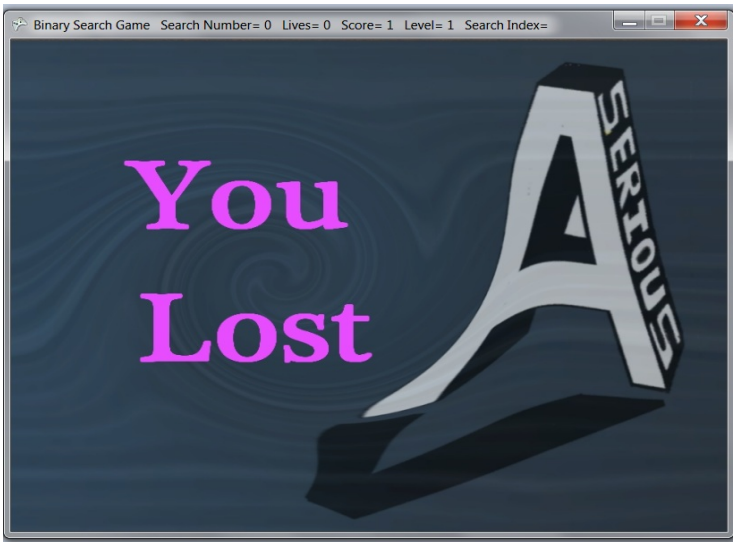

Figure 9.54: Binary Search Game–Lost Game Screen

12- High Scores Screen

Figure 9.55: Binary Search Game–High Scores Screen

13- Player Report Screen

This screen (Figure 9.56) displays the report for a player named Sahar.

Figure 9.56: Binary Search Game–Player Report Screen

14- Credits Screen

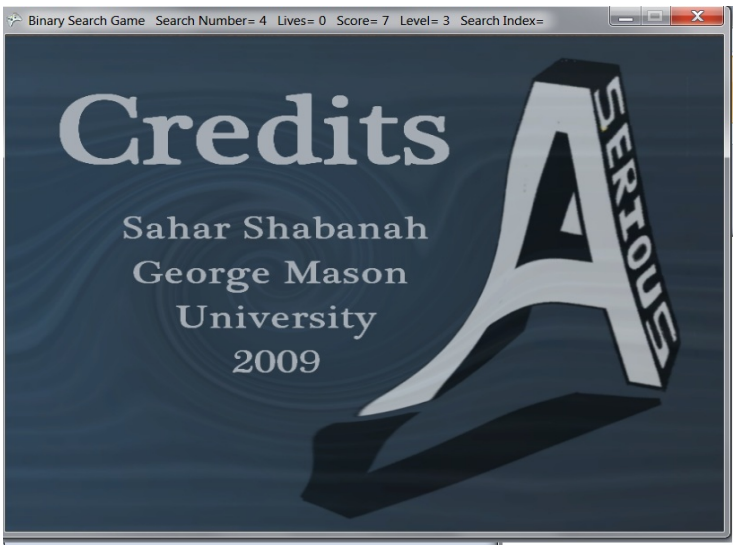

Figure 9.57: Binary Search Game–Credits Screen

9.6 Bubble Sort Game Implementation

9.6.1 Game Configuration and Content

The Bubble Sort game prototype, which has been described in Section 6.2, is summarized in the following.

Game Properties

The Bubble Sort game properties are as follows:

- Maximum level number=3.
- Maximum lives number=3.
- Timer Limit=1 Second.
- Game Name=Bubble Sort.

These game properties were entered into the game using the Game Properties Editor as shown in Figure 9.58.

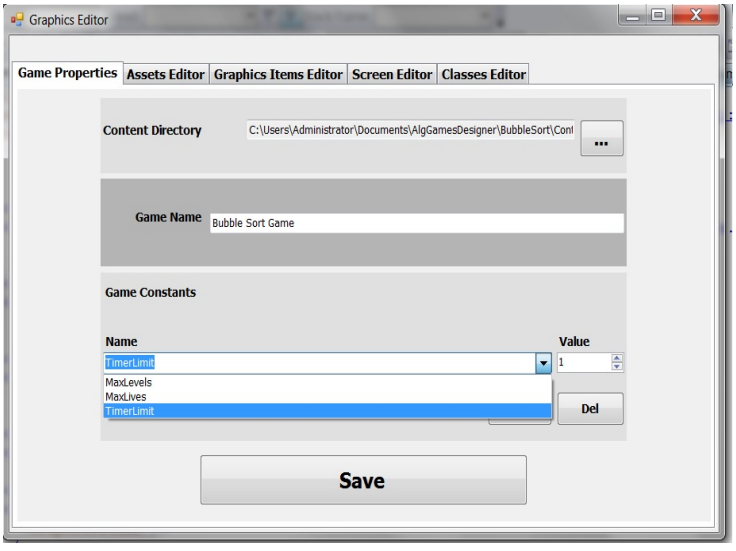

Figure 9.58: Bubble Sort Game–Enter Game Properties

Game Assets

The Bubble Sort game assets are as follows:

- Textures: 52 Cards, 13 for each card suit, Swap button, and Card Cover.
- Sounds: LostLive, Won, and Lost.
- Fonts: Ariel.

These game assets were entered into the game using the Game Assets Editor as shown in Figure 9.59.

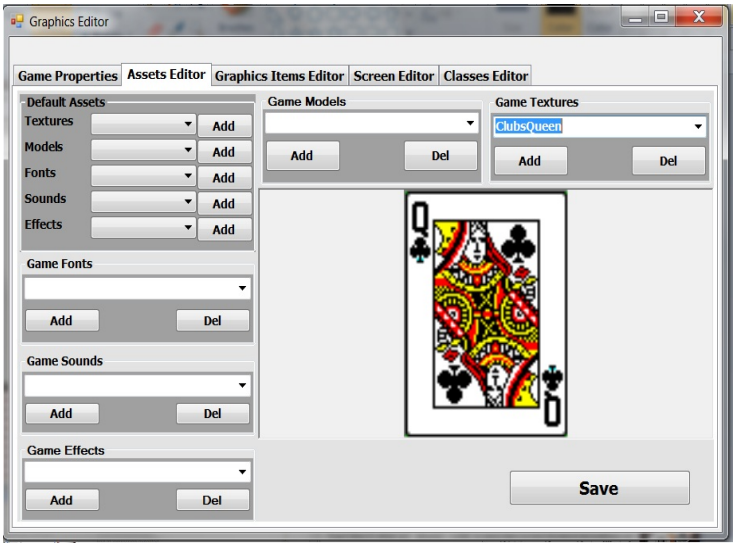

Figure 9.59: Bubble Sort Game–Enter Assets files

Game Classes

The Bubble Sort game classes are as follows:

- Card class is a default class.
- Deck class is a default class.
- BlockSp class is a default class.

- Button class is a default class.

These game classes were entered into the game using the Game Classes Editor.

Game Graphic Items

The Bubble Sort game graphics Items are as follows:

- Card has a texture (Card) that determines its appearance, class (BlockSp) that defines its behavior, and attributes: position, repetition, and size.
- Deck, which consists of a set of cards, has a class (Deck) that defines its behavior and attributes: position, repetition, and size.
- Swap has a texture (Swap) that determines its appearance, class (Button) that defines its behavior, and attributes: position, repetition, and size.

These items were created and added to the game using the Game Graphic Items Editor as shown in Figures 9.60, 9.61, and 9.62.

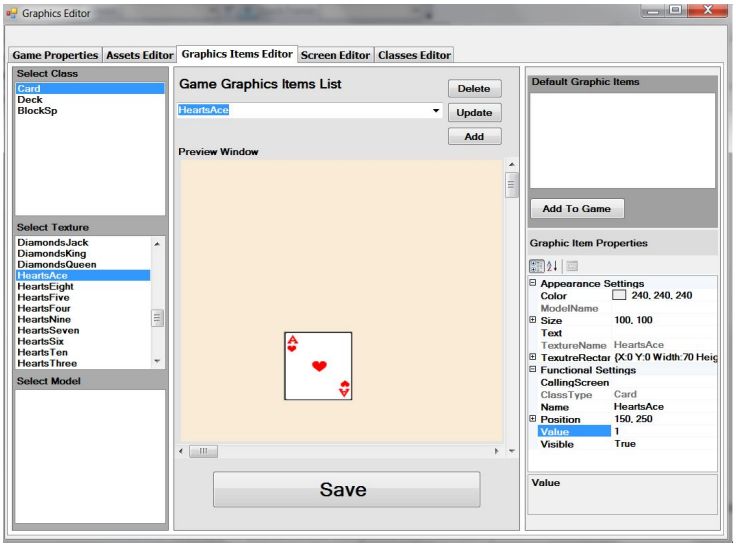

Figure 9.60: Bubble Sort Game–Graphic Item [Ace] Design

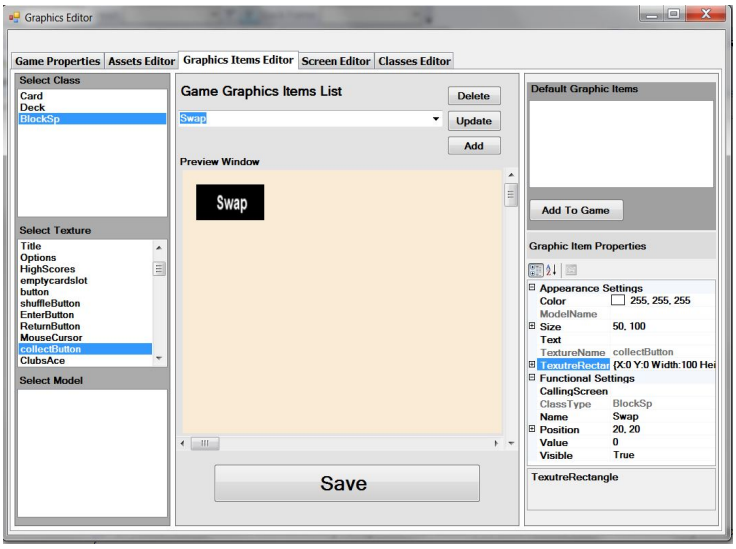

Figure 9.61: Bubble Sort Game–Graphic Item [Swap] Design

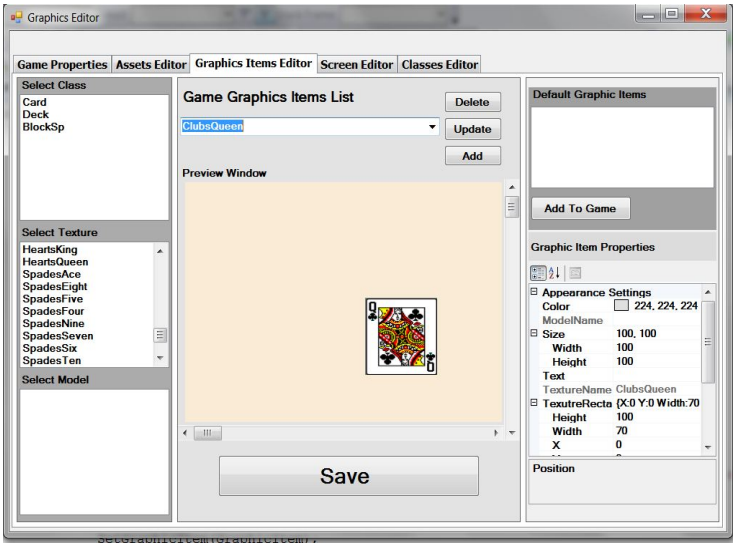

Figure 9.62: Bubble Sort Game–Graphic Item [Queen] Design

Game Screens

The Bubble Sort game screens are as follows:

- All default Algorithm Game Screens that have been used in the game without modification except for the Play screen.

- The play screen that has been customized by adding two graphic items: Card and Swap.

The Game Screens Editor was used to add all game screens as shown in Figure 9.63 and to customize the Play screen as shown in Figures 9.64, 9.65, and 9.66.

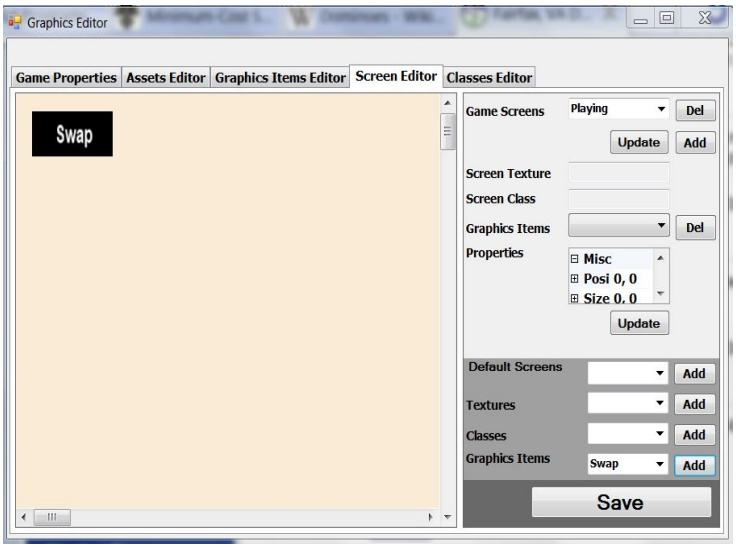

Figure 9.63: Bubble Sort Game–Adding [Swap] to Play Screen

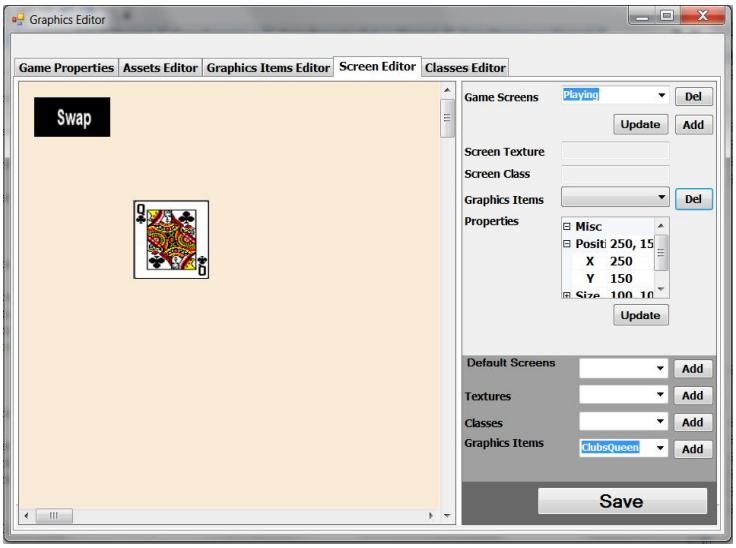

Figure 9.64: Bubble Sort Game–Adding [Queen] to Play Screen

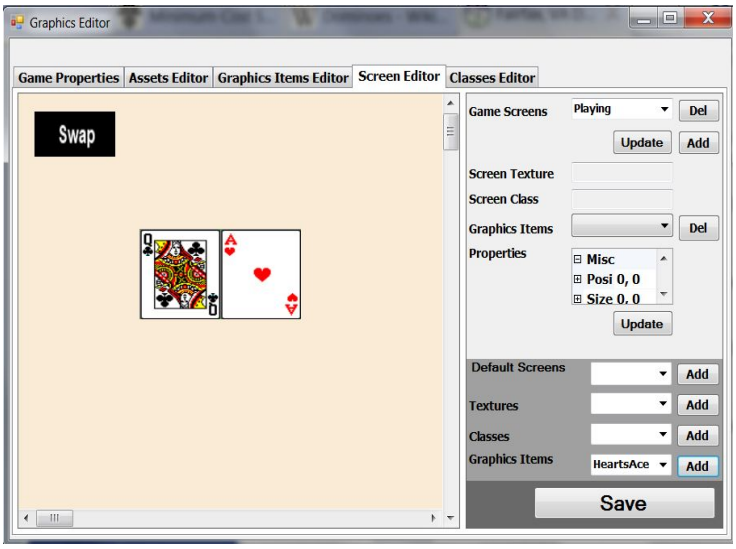

Figure 9.65: Bubble Sort Game–Adding [Ace] to Play Screen

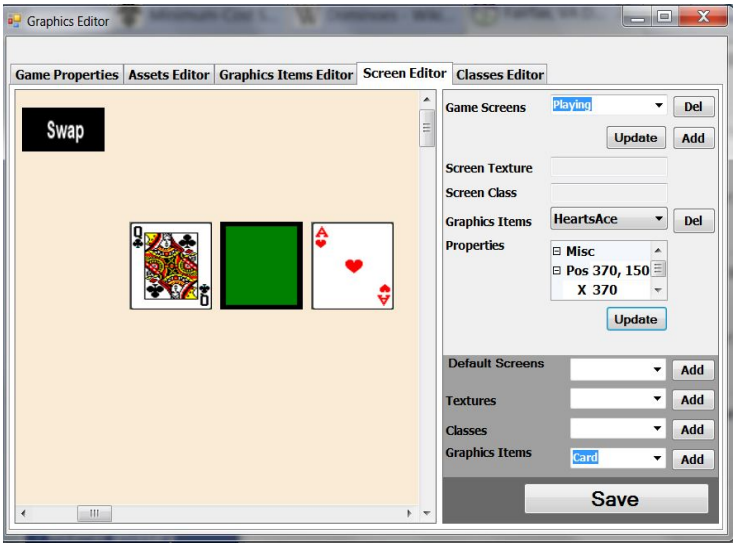

Figure 9.66: Bubble Sort Game–Adding [Covered Card] to Play Screen

9.6.2 Game-Play

The BaseGame and GamePlay classes, which have been included in the template when the game was created, were implemented as follows:

BaseGame Implementation

In the BaseGame class, three methods need implementation:

1. `SetUpGameProperties`(); the game uses all default game properties, so there is no need to implement this method.

2. `SetUpGameScreens`(); the game uses all default game screens, so there is no need to implement this method.

3. `SetGameGraphicsItems`(); this method initializes all game graphic items to be used later in the GamePlay class. As explained in Section 8.6.2, all graphic items have been stored in the GameGraphicsItemsDic, so they can be accessed using their names as shown in the following code snippet:

```
private void SetGameGraphicsItems()
{
      cCard = (Card)base.GameItemsDic["Card"];
      cardDeck = new Deck(true, cCard);
      cardDeck.Load();
      Swap = (BlockSp)base.GameItemsDic["Swap"];
}
```

GamePlay Implementation

In the GamePlay class, six methods need implementation:

1. `SetupGame`(); the game uses the default settings, so there is no need to implement this method.

2. `StartGame`(); the game uses the default method, so there is no need to implement this method.

3. `GetPlayerInput`(); this method registers the player input and responds accordingly as shown in the following code snippet:

```
protected override void GetPlayerInput(GameTime gameTime)
{
   if (Input.WasPressed(0,InputManager.GamepadButtonType.Start,
       Keys.Enter, InputManager.MouseButtonType.Left))
     { //swap operation
      if (Input.MouseHandler.MouseInBox(OurGame.Swap.innerRect))
{
           Swap = true;
           ProcessClickedSwap();
         }
         for (int i = 0; i < NumOfCards; i++)
             {
               if (Input.MouseHandler.MouseInBox
                     (OurGame.cardDeck.CardsInDeck[i].innerRect))
                     ProcessClickedCard(i);
             }//selectedCardIndex
      }
}
```

4. `StartLevel`(); this method starts one game level by calculating the number of the cards in the deck related to the level number as shown in the following code snippet:

```
protected override void StartLevel()
{
   base.StartLevel();
   NumOfCards = (CurrentLevel) * NumOfColumns;
   clickedSlotIndex.Clear();
   OurGame.cardDeck.ShuffleDeck();
   if (IsSorted(OurGame.cardDeck.CardsInDeck))
          OurGame.cardDeck.ShuffleDeck();
   LastCardIndex = 0;
}
```

5. `RenderLevel`(); this method draws a new game level on the screen by rendering Deck of Cards and Swap button as the next code snippet shows:

```
protected override  void RenderLevel()
{
    OurGame.GraphicsDevice.Clear(Color.OliveDrab);
    OurGame.Swap.Render(OurGame.SpriteBatch);
    RenderCards();
    MenuCursor.Render(OurGame.SpriteBatch);
}
```

6. `PlayLevel`(); this method implements the game-play by following the bubble sort algorithm steps as shown in next code snippet:

```
protected override void PlayLevel()
{
     if (IsSorted(OurGame.cardDeck.CardsInDeck))
      UnCoverAll();
}// method
```

9.6.3 Bubble Sort Game Screen Shots

1- Title Screen

Figure 9.67: Bubble Sort Game–Title Screen

2- Player Name Screen

Figure 9.68: Bubble Sort Game–Enter Player Name

3- Main Menu Screen

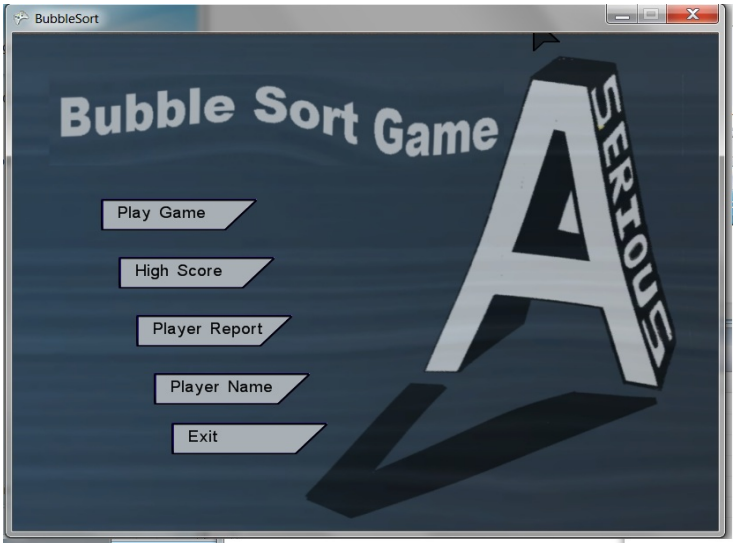

Figure 9.69: Bubble Sort Game–Main Menu Screen

4- Start Level Screen

Figure 9.70 shows the first level of the game with a deck of eight cards; the HUD items are Time=1.5 minutes, Lives=3, Level number=1, and Score=0.

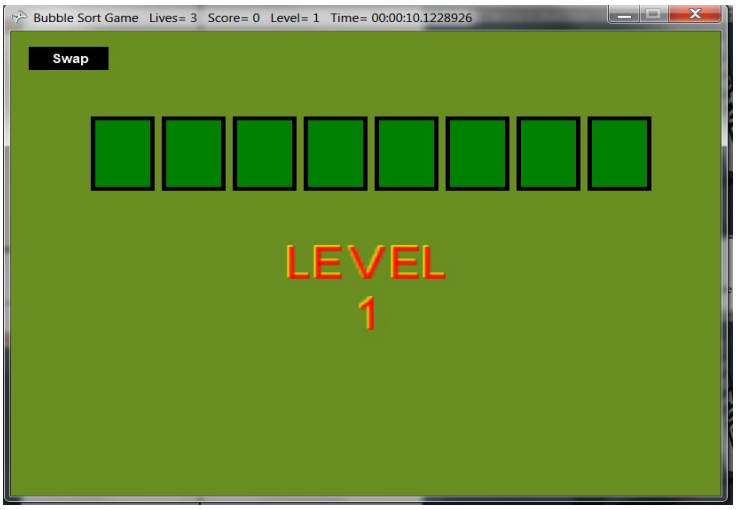

Figure 9.70: Bubble Sort Game–Play Screen (Level 1)

Figure 9.71 shows the second level of the game with a deck of 16 cards; the HUD items are Lives=3, Level=2, Score=80, and time=3 minutes.

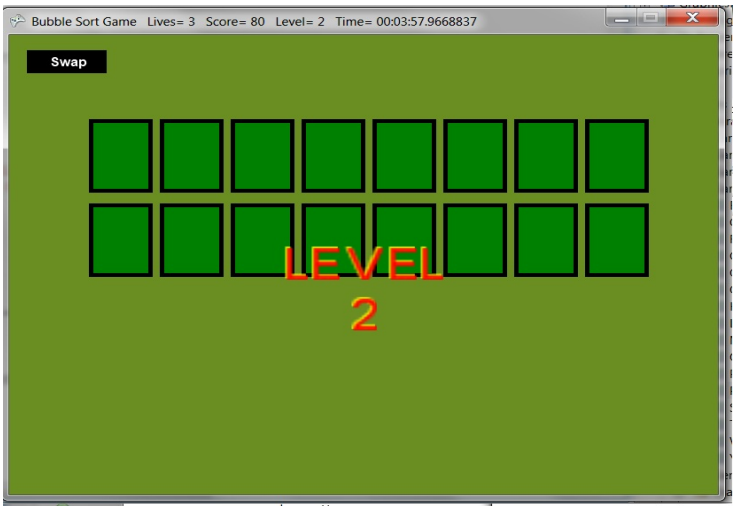

Figure 9.71: Bubble Sort Game–Play Screen (Level 2)

Figure 9.72 shows the third level of the game with a deck of 24 cards; the HUD items are Lives=3, Level=3, Time increased, and Score=0.

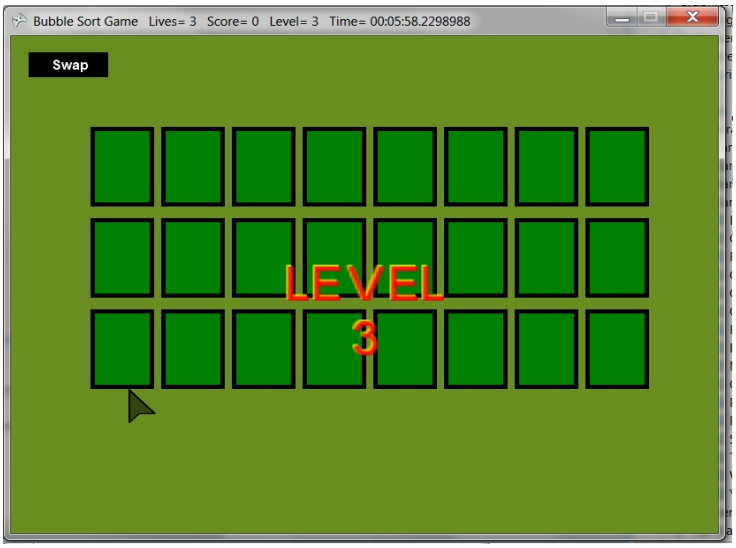

Figure 9.72: Bubble Sort Game–Play Screen (Level 3)

5- Play Screen

- Figure 9.73 shows the Play screen after the player uncovers the first two adjacent cards; the HUD items are Score=0, since the second card value < first card value, and the player must click on Swap button to swap the cards.
- Figure 9.74 shows the cards after having been swapped; the HUD items Score=5 and the decreased Time.

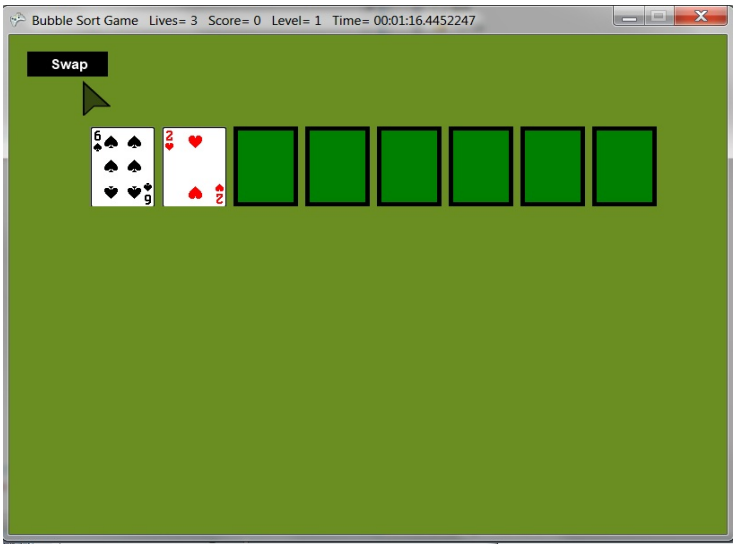

Figure 9.73: Bubble Sort Game–Play Screen (Card Swap Operation 1)

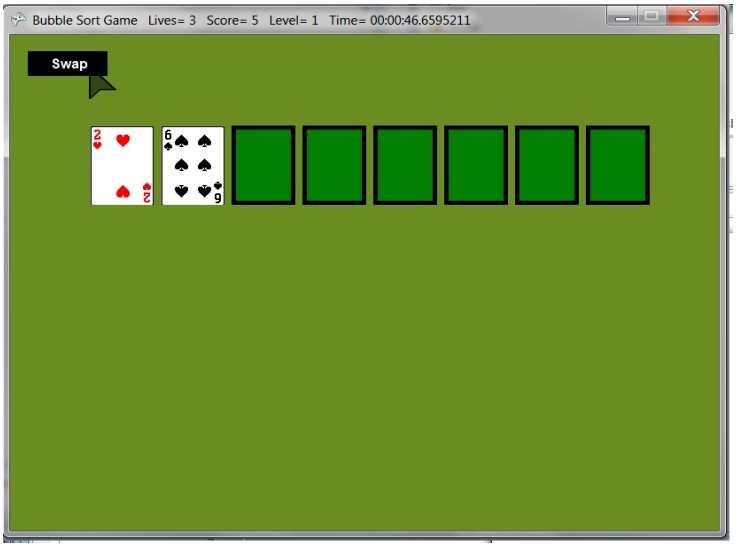

Figure 9.74: Bubble Sort Game–Play Screen (Card Swap Operation 2)

6- Pause Screen

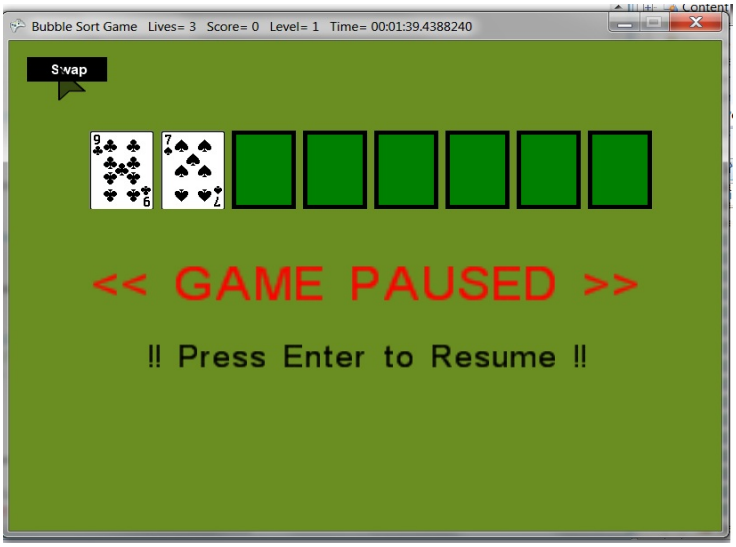

Figure 9.75: Bubble Sort Game–Pause Screen

7- Exit Screen

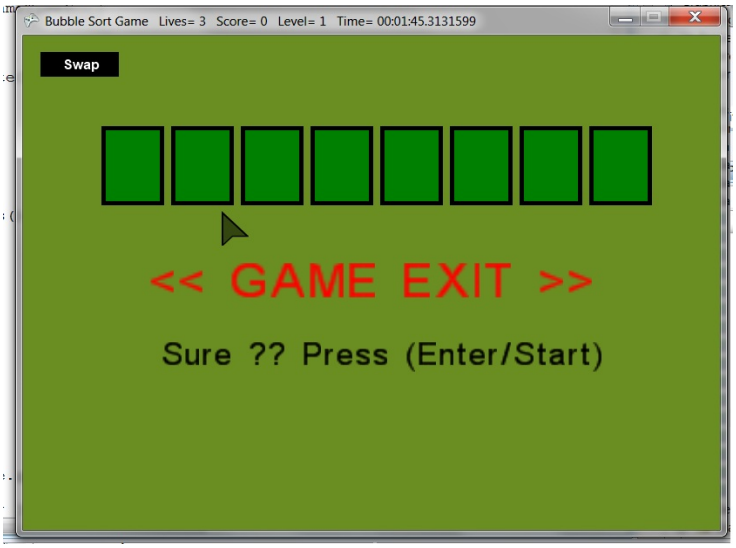

Figure 9.76: Bubble Sort Game–Exit Screen

8- Lost Level Screen

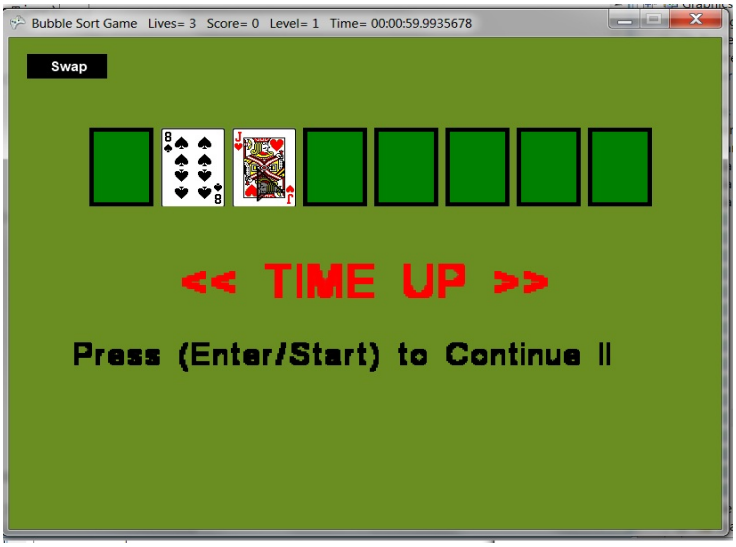

Figure 9.77: Bubble Sort Game–Lost Level Screen

9- Won Level Screen

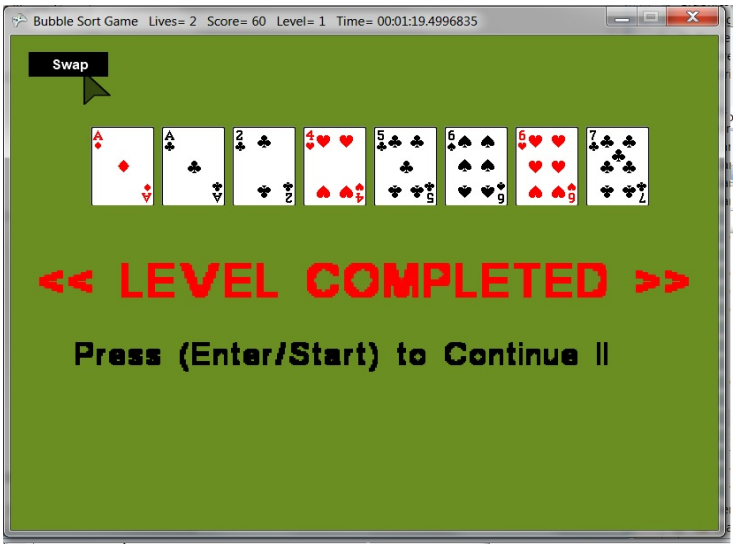

Figure 9.78: Bubble Sort Game–Won Level Screen

10- Won Game Screen

Figure 9.79 shows the HUD items, which are Score=12, Level =3, and Lives=1.

Figure 9.79: Bubble Sort Game–Won Game Screen

11- Lost Game Screen

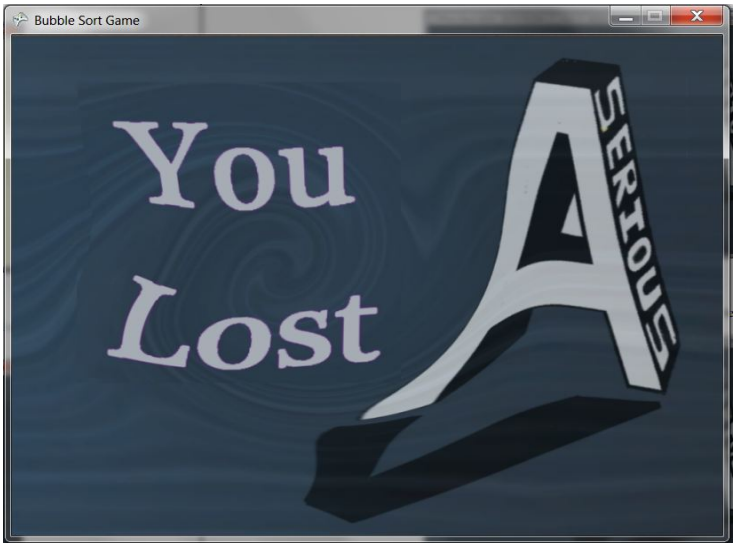

Figure 9.80: Bubble Sort Game–Lost Game Screen

12- High Scores Screen

Figure 9.81: Bubble Sort Game–High Scores Screen

13- Credits Screen

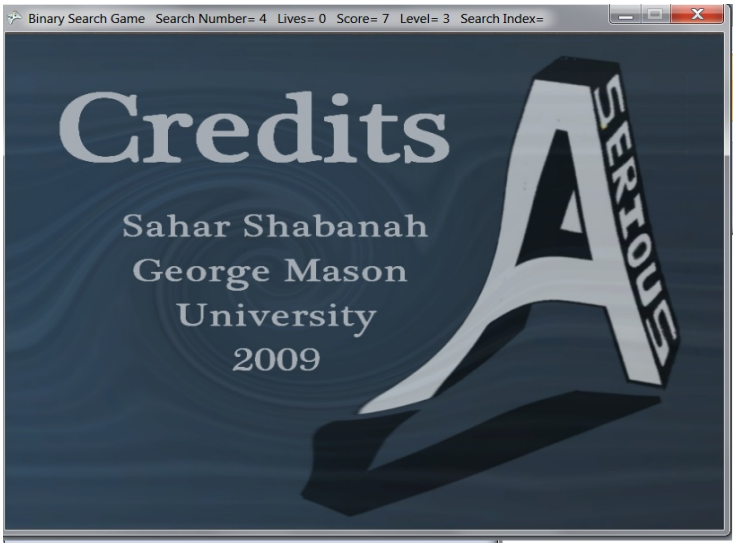

Figure 9.82: Bubble Sort Game–Credits Screen

14- Player Report Screen

Figure 9.83 displays the report for the player named Sahar.

Figure 9.83: Bubble Sort Game–Player Report Screen

Chapter 10: Evaluation and Future Work

10.1 Evaluation

Researchers in computer science education identify many pedagogical requirements for the success of an algorithm visualization system [20]; the most effective factor is the students' engagement, since there is an increasing correlation between the level of student learning and engagement in algorithm visualization systems [21]. Alternatively, computer games fully interact with players and encourage them to think and act. Furthermore, according to Rieber [22], "playing computer games involves active engagement" because computer games support all components of flow [23], as defined by Csikszentmihalyi [24]. In fact, recent research shows that computer games build content and activities into high levels of motivation and interest that are useful in reinforcing a wide variety of learning values [25]. Therefore, I can conclude that the use of computer games to visualize algorithms can help and motivate students in algorithm learning process. However, a study can be carried out to investigate the affects of using computer games in algorithm learning by comparing the performance of three treatment groups. This section describes a study about the affects of computer games on algorithm learning. In this study, three sorting algorithms–Bubble, Insertion, and Selection sorts– will be taught to three groups of students using three methods to compare their effects on the learning of the algorithms. With first group, no visual tool is used to teach algorithms under study, with second group, students watch a demonstration of algorithm games, and with third group, algorithm games are used to teach the algorithms.

10.1.1 Study Design

The three Treatments were identified based on the type of visualization students had for the three sorting algorithms as follows:

1. Students in Treatment Text, see the algorithm text and flowchart visualizations only.

2. Students in Treatment View, see the algorithm text, flowchart, and game demo visualizations in lecture controlled by the instructor.

3. Students in Treatment Play, see the algorithm text, flowchart, and game demo visualizations in lecture controlled by the instructor, then play the algorithm game during a closed lab.

The study must be applied on students in computer science introductory courses, where these type of algorithm are usually taught. For each course, three sections are needed; in each section, a group with one Treatment must be taught using the suggested type of visualization.

10.1.2 Study Materials

1. Students must complete a pre-test before reading any information about the sorting algorithms (Appendix B.1), which includes:

   (a) Questions to measure the student's learning preferences.

   (b) Questions to measure the student's current knowledge about the tested algorithms.

2. Identical lecture slides must be used to describe each algorithm, but without including any visual representations of the algorithm.

3. Serious-AV system must supplement the lecture for the three Treatments by allowing students in each Treatment to have direct access to its visualizations as follows:

   (a) Students in all Treatments can see the algorithm text and flowchart.

   (b) Students in Treatment View can also watch the game demo, which describes the algorithm behavior.

   (c) Only students in Treatment Played can play the algorithm game.

4. After learning the algorithms using Serious-AV, students must complete a post-test that include (Appendix B.2):

   (a) Questions about the played algorithms, which are identical to the questions on the pretest.

(b) A brief survey about the student experience with the played algorithm games.

10.1.3 Study Procedure

1. Students must complete the pretest before covering any of sorting algorithms material.
2. Same lecture slides must be used to describe the algorithms for all groups.
3. The post-test must administered after a reasonable time period (at least a week for each algorithm learned).
4. Exams must be graded consistently using the same grading metric.

10.1.4 Data Analysis Method

Student scores will be collected from three unmatched groups to be compared and analyzed. These scores are not derived (non-parametric) and not normally distributed. Therefore, a non-parametric test like Kruskal-Wallis can be used to compare treatments. Learning can be measured by subtracting pretest from post-test scores.

Kruskal-Wallis

The Kruskal-Wallis Test was developed by Kruskal and Wallis jointly and named after them. The Kruskal-Wallis test is a nonparametric (distribution free) test, which is used to compare three or more groups of sample data. Kruskal-Wallis Test is used when there the assumption that the distribution of each group should be normally distributed is not met. Therefore, Kruskal-Wallis Test is a distribution free test. The Kruskal-Wallis test is a nonparametric method of testing the hypothesis that several populations have the same continuous distribution versus the alternative that measurements tend to be higher in one or more of the populations.

Hypothesis in Kruskal-Wallis Test:

- Null hypothesis: In Kruskal-Wallis Test, null hypothesis assumes that the samples are from identical populations.

- Alternative hypothesis: In Kruskal-Wallis Test, alternative hypothesis assumes that the sample comes from different populations.

Assumptions of Kruskal-Wallis Test:

In Kruskal-Wallis Test, it is assumed that the samples drawn from the population are random, the cases of each group are independent, and the measurement scale for Kruskal-Wallis Test is at least ordinal.

Procedure for Kruskal-Wallis Test:

1. Arrange the data of all samples in a single series in ascending order.
2. Assign rank to them in ascending order. In the case of a repeated value, assign ranks to them by averaging their rank position.
3. Once this is complete, ranks or the different samples are separated and summed up as R1 R2 R3, etc.
4. To calculate the value of Kruskal-Wallis Test, apply the following formula:

$$H = \frac{12}{n(n+1)} \sum_{i=1}^{k} \frac{R_i^2}{n_i} - 3(n+1)$$

H = Kruskal-Wallis Test

n = total number of observations in all samples

R= Rank of the sample

Kruskal-Wallis Test statistics is approximately a chi-square distribution, with k-1 degree of freedom where ni should be greater than 5.0 If the calculated value of Kruskal-Wallis Test is less than the chi-square table value, then the null hypothesis will be accepted. If the calculated value of Kruskal-Wallis Test H is greater than the chi-square table value, then the null hypothesis will be rejected.

10.1.5 Possible Discussion Questions

Many questions need to be discussed after conducting the study, such as what are the important results? Does learning improve as the level of student engagement with AV increases from viewing to playing? Do treatment groups have similar academic levels? Is there a distinction in learning when using dissimilar kinds of visualizations? What influence does playing games have on student motivation? Do learning preferences influence the students choice of preferred visualization? Is it difficult to gather data in classrooms?

10.1.6 Study Results

Due to time constraints, the study could not be applied as designed here. These limitations will be addressed in future work by the researcher. Instead, a preliminary study that used small sample size led to some encouraging initial results about the effects of using games in algorithm learning. Participants liked the idea of playing games to learn the algorithms. Moreover, even participants with no computer science background were able to learn how an algorithm works by playing its game.

10.2 Future Research

The possibilities for future research for using computer games in teaching algorithms are opened up by the work described here. In this research only sorting and searching algorithms have been considered in developing algorithm games prototypes, including Binary Search, Bubble Sort, Insertion Sort, Linked List, and Binary Tree operations. Therefore, future research should examine other types of algorithms to develop game prototypes that visualize them. In addition, the SAVGEngine repository must be updated with more components, such as implementations of more known data structures, algorithms, graphic items, game screens, and game assets that support the design of algorithm games for the newel investigated algorithm types.

Another research to investigate the development of a DSL for Algorithm Game creation can also be carried out in the future. DSL (Domain-Specific Language) is a programming language geared

towards solving problems in a specific domain. The use of DSL is a new trend

in game development research that have not been fully investigated yet.

Alternatively, the support for using a script language, such as Iron-Python or Lua in implementing algorithm games code can be researched by designing a script manager and editor. The Script Editor must feature IntelliSense-like programmer assistance, along with prototyped definitions for scripting event functions. It should export the script files into a format supported by the game engine. Specifically, code written in script files should have access to components of the game to further abstract specific behavior.

The Algorithm Game Designer has been developed to create algorithm games with minimal training and effort. However, a future research can address modifying it to be used to create computer games other than algorithm games.

Finally, limitations from algorithm games effects preliminary study, which has been conducted in this research, such as small sample size and short duration, can be addressed with future research. Larger sample sizes, from the university students, would provide a broader understanding of the effectiveness of algorithm games. Our short term results indicate that algorithm games are educational, enjoyable, and engaging. Longitudinal studies will be required to determine if these motivational benefits are retained in the long run.

In fact, more than one study can be conducted to study the outcomes of this research. Beside the previously described study, a second study can be carried out to compare the performance of three treatment groups using the three different learning models of AVuSG approach, which are Gagne based, Bloom based, and Constructivist models.

Chapter 11: Conclusion

Algorithms play a major role in Computer Science Education, but they are complex and hard to learn. Therefore, educational tools, such as Algorithm Visualization Systems are needed to help students better learn and understand algorithms [3]. However, the focus on graphics and sound instead of teaching aspects in the design of current algorithm visualization systems in addition to their lack of features that encourage student engagement are responsible for their failure to be effective in teaching algorithms [14] [20]. This research has addressed some required issues in creating algorithm visualization techniques by integrating learning theories and models in algorithm teaching and by visualizing algorithms, using computer games to maximally engage students in the algorithms learning process.

Several algorithm visualization systems have been compared according to their level of engagement as defined by Active Engagement Taxonomy [35]. Computer games fully interact with their players and encourage them to think, talk, and act. Therefore, by using computer games to visualize algorithms, "playing" computer games has been introduced as an alternative form of engagement that actively engage students through repetition, challenge, and enjoyment in addition to combining all five forms of active engagement–viewing, responding, changing, constructing, and presenting–described by the Active Engagement Taxonomy.

The history of computer games usage in education has been reviewed and several educational games have been surveyed, including edutainment games, research educational games, and serious games. Despite the existence of many educational computer games that teach a variety of subjects, there is still need for more games to teach many other subjects, such as algorithms and data structures. Consequently, this research has engendered a new research area for the usage of computer games in algorithm learning. Algorithm Visualization using Serious Games (AVuSG) has been defined as a novel approach for visualizing algorithms using serious or computer games to overcome the shortness in current algorithm visualization systems. Specifically, several questions have been

raised and answered during this research as explained next.

How the pedagogical requirements can be satisfied when teaching algorithms using computer games? AVuSG produces three forms of visualization: text, flowchart, and game, for each algorithm under consideration. It defines three learning processes: viewing, playing, and designing that learners can use to engage with each one of these three forms of visualization. It integrates learning theories with game design to introduce three educational models that instructors can deploy in their classes to teach students algorithms:

1. Bloom Based Model.
2. Gagne Based Model.
3. Constructivist Based Model.

Serious Algorithm Games Visualizer (Serious-AV) is an algorithm visualization system that has implemented AVuSG framework by including several viewing and development tools that support the creation and viewing of the three algorithm visualization forms (text, flowchart, and game), which the AVuSG approach must produce. These tools can be grouped into two main subsystems as follows:

1. Serious-AV Viewers that support the viewing and the playing processes by providing three viewers:
   (a) The Algorithm Text Viewer that shows the algorithm text to the algorithm learners.
   (b) The Algorithm Flowchart Viewer that presents the algorithm flowchart to the algorithm learners.
   (c) The Algorithm Game Viewer that displays the algorithm game to the algorithm learners.
2. Serious-AV Designers that support the designing process by providing three designers:
   (a) The Algorithm Text Designer that simplifies the creation of the algorithm text.
   (b) The Algorithm Flowchart Designer that simplifies the creation of the algorithm flowchart.

(c) The Algorithm Game Designer that simplifies the creation of the algorithm game.

Serious-AV can be used by both instructors and students. The instructor uses Serious-AV designers to design the text, flowchart, and game for the algorithm under study. The students use Serious-AV viewers to view their instructor's designs. Depending on the deployed AVuSG learning model, students may also create their own algorithm text, flowchart, and game designs.

What are the specifications of the educational computer games that efficiently visualize and teach algorithms to new generations? In this research, the computer games that can be used to visualize algorithms have been specified. First, these games have been named as "Algorithm Games" and have been defined using instructional and simulation games definitions. Second, Algorithm games attributes have been given and their genres have been determined.

How computer games can be designed and developed to efficiently visualize algorithms? This research detailed the design and development process of algorithm games by reviewing the theory of computer games design in general and then deriving an Algorithm Game prototype that can be used to simplify the design of any algorithm game by providing general design elements and gameplay. At the same time, several development designers including an Algorithm Game Designer have been created in this research to simplify the development and creations of algorithm games visualizations with minimal training and effort. However, game design and development are not easy tasks, so more attention has been given to the development of the Algorithm Game Designer, which has several components and editors to automate the game development as follows:

1. Algorithm Game Template: is used as a blueprint for creating new algorithm games. It provides each newly created game with its architecture by defining its basic components, such as the BaseGame class, PlayGame class, Content project, and 14 default Game Screens.

2. Serious Algorithms Visualization Game Engine (SAVGEngine): has various modules that encapsulate all the functionality for creating an algorithm game, such as graphics, sound, input, game screens, physics, and storage managers. It also includes BaseGame and PlayGame

classes that provide the new algorithm game with game timing and rendering loop in addition to the implementation of all basic game-play rules of the game.

3. Algorithm Game Components Repository: has built-in, ready-to-use algorithm game components that can be used to develop various algorithm games, such as game assets, screens, graphic items, and data structures algorithms.

4. Basic Code Editor: helps in game coding by providing project management and programming language support (C# and XNA), such as debugging, compiling, and executing operations.

5. Developer Graphics Editors: support the visual creation of different game content. There are five types of such editors: Properties, Assets, Graphic Items, Classes, and Game Screens.

What methodologies and technologies can be used to produce prototypes for these games? Various game development technologies, such as application program interfaces, visual creation tools, game engines, and domain-specific languages have been surveyed and compared to take advantage of their benefits and avoid their mistakes when designing the Algorithm Game Designer. Moreover, several new technologies have been deployed in the implementation of the Algorithm Game Designer and its components, such as Microsoft Visual C#, .NET Framework, XNA Game Studio, and Visual Studio Shell-Isolated Mode.

In conclusion, AVuSG approach provides rich visualizations that take advantage of computer games, modern graphics, and audio technology. These visualizations have been developed to improve the student learning experience by creating a more engaging and immersive environment. Moreover, the approach benefits from the player's desire to win, love of competition, and entertainment accompanies playing games to motivate students learning algorithms. Furthermore, it facilitates students assessment by using the winning-losing paradigm of computer games without the need for testing.

Appendix A: Development Method and Implementation Technology

A.1 Agile Methodology

In this research, Agile method is used in developing all software systems. Agile is a conceptual framework for software development using short iterations. Each iteration is like a short project in itself. It uses (inspect and adapt) practices to adjust the project plan. Using Agile the algorithm game can be built with minimum features to be tested then more features added later in next iterations to speed up the game development process.

A.2 Technologies Deployed

The following technologies have been deployed during the development of the systems.

A.2.1 Microsoft Visual C# (3.0)

C# (pronounced C Sharp) is a multi-paradigm programming language that encompasses functional, imperative, generic, and object-oriented (class-based) programming disciplines. It is developed by Microsoft but made available to the general public through international standards. The current release is termed as 3.0. It is supported by the .NET Framework's Common Language Runtime.

A.2.2 Microsoft .NET Framework (3.5)

The Microsoft .NET Framework is a managed code programming model for building applications on Windows clients, servers, and mobile or embedded devices. Developers use .NET to build applications of many types: web applications, server applications, smart client applications, console applications, database applications, and more.

A.2.3 Microsoft XNA Game Studio Express (3.0)

Microsoft XNA GSE is a set of tools based on Microsoft Visual C# that allow students and hobbyists to build games for both Microsoft Windows and Xbox 360 (Figure A.1). XNA games can be

compiled and run from the moment they are created because of the following useful components:

- XNA Framework, which is a set of managed libraries based on the Microsoft .NET Framework that are designed for simplifying graphics, audio, input, and storage used in all games.
- XNA Framework Content Pipeline, which is integrated into the build environment and allows game developers to add their art content, such as 3D models, graphical textures, shaders, and audio into their solution in the IDE. Then the content will be built into runtime objects that can be accessed in code. This reduces the overhead associated with art content.
- Game Class, which constructs the timing and render loops automatically to handle its own timing and rendering. It exposes an Update method where objects can be added to update each frame and a Draw method where objects can be added to render to the screen. It also sets up the appropriate graphics device without complicated device enumeration code.

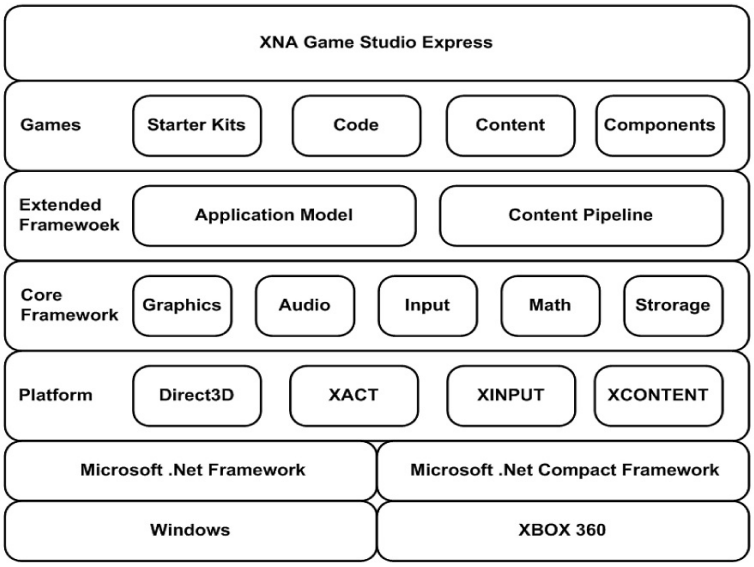

Figure A.1: XNA Game Studio Architecture

A.2.4 Microsoft Visual Studio

It is a complete set of development tools for building ASP.NET, web applications, XML Web Services, desktop applications, and mobile applications. Visual Basic, Visual C++, Visual C#, and Visual J# all use the same IDE, which enables them to share tools and makes creating mixed-language

solutions easier.

A.2.5 Microsoft Visual Studio Shell (2008)

It provides a redistributable version of the Visual Studio Integrated Development Environment (IDE) from which the programming languages and the features that support their respective project systems have been removed. This IDE-only version of Visual Studio serves as a development platform to which developers can add their own development tools and programming languages, such as COBOL and Perl.

A.2.6 Visual Studio Shell: Isolated Mode (2008)

The Visual Studio Isolated Shell is a unique stand-alone application that runs side-by-side with other versions of Visual Studio that are installed on the computer. It is optimized for the deployment of specialized tools that have full access to Visual Studio services but also have extensive, custom appearance, and branding flexibility. Visual Studio features and menu command groups can be turned on and off. Application titles, application icons, and splash screens are fully customizable.

A.2.7 Microsoft Visual Studio SDK (2008)

The Visual Studio SDK provides a framework that can be used to extend the functionality of Microsoft Visual Studio (IDE). Visual Studio SDK provides three ways to extend the Visual Studio IDE: macros to automate repetitive tasks, a flexible add-in framework, and VSPackages, which are the same kinds of extensions that were used to build major Visual Studio components, such as the C# language integration.

Appendix B: Sorting Algorithms Study Tests

B.1 Pre-test

B.1.1 Preferred Learning Approach

This part of the test identifies your preferred learning approach. Choose your preferred answer for each question:

1. After reading a novel, what you will do

   (a) watching a movie about it.

   (b) playing a computer game that simulate its events.

2. To relax, I would like to

   (a) play a computer game.

   (b) read a book.

3. I prefer

   (a) drawing, playing, or working with my hands.

   (b) speaking, writing, and listening.

4. To learn in class, I would prefer to

   (a) play games related to the topic.

   (b) participate in a class discussion about the topic.

5. To learn an algorithm, I would prefer to

   (a) watch a drawing that describes its behavior.

   (b) apply its steps on sample data.

6. To understand a process, I prefer to

   (a) see a diagram describing its steps.

   (b) play a game simulating it.

B.1.2 Sorting Algorithms

This part of the test measures your knowledge about sorting algorithms. If you do not know the right answers, do your best.

1. Apply the bubble sort algorithm to the following array. Use the empty table cells to show the contents after each pass. The array may or may not be sorted after the third pass.

Array Index	0	1	2	3	4
Initial State of Array	48	87	35	12	37
show the array contents after the first pass					
show the array contents after the second pass					
show the array contents after the third pass					

2. Given an initial array with N items that is already sorted, how many comparisons (not passes) would bubble sort make? Number of Comparisons _____

3. Apply the insertion sort algorithm to the following array. Use the empty table cells to show the contents after each pass. The array may or may not be sorted after the third pass.

Array Index	0	1	2	3	4
Initial State of Array	48	87	35	12	37
show the array contents after the first pass					
show the array contents after the second pass					
show the array contents after the third pass					

4. Given an initial array with N items that is already sorted, how many comparisons (not passes) would insertion sort make? Number of Comparisons _____

5. Apply the selection sort algorithm to the following array. Use the empty table cells to show the contents after each pass. The array may or may not be sorted after the third pass.

Array Index	0	1	2	3	4
Initial State of Array	48	87	35	12	37
show the array contents after the first pass					
show the array contents after the second pass					
show the array contents after the third pass					

6. Given an initial array with N items that is already sorted, how many comparisons (not passes) would selection sort make? Number of Comparisons _____

7. Consider Selection Sort, Insertion Sort, and Bubble Sort. Which of the three sorting algorithms (if any) will always perform the same number of passes and comparisons regardless of the initial array? The correct answer might include more than one algorithm.

8. Sort the Selection Sort, Insertion Sort, and Bubble Sort algorithms according to their speed in sorting an array.

   _____ Selection Sort

   _____ Insertion Sort

   _____ Bubble Sort

B.2 Post-Test

B.2.1 Sorting Algorithms

This part of the test measures your knowledge after studying the sorting algorithms.

Please answer each question:

1. Apply the bubble sort algorithm to the following array. Use the empty table cells to show the contents after each pass. The array may or may not be sorted after the third pass.

Array Index	0	1	2	3	4
Initial State of Array	48	87	35	12	37
show the array contents after the first pass					
show the array contents after the second pass					
show the array contents after the third pass					

2. Given an initial array with N items that is already sorted, how many comparisons (not passes) would bubble sort make? Number of Comparisons _____

3. Apply the insertion sort algorithm to the following array. Use the empty table cells to show the contents after each pass. The array may or may not be sorted after the third pass.

Array Index	0	1	2	3	4
Initial State of Array	48	87	35	12	37
show the array contents after the first pass					
show the array contents after the second pass					
show the array contents after the third pass					

4. Given an initial array with N items that is already sorted, how many comparisons (not passes) would insertion sort make? Number of Comparisons _____

5. Apply the selection sort algorithm to the following array. Use the empty table cells to show the contents after each pass. The array may or may not be sorted after the third pass.

Array Index	0	1	2	3	4
Initial State of Array	48	87	35	12	37
show the array contents after the first pass					
show the array contents after the second pass					
show the array contents after the third pass					

6. Given an initial array with N items that is already sorted, how many comparisons (not passes)

would selection sort make? Number of Comparisons _____

7. Consider Selection Sort, Insertion Sort, and Bubble Sort. Which of the three sorting algorithms (if any) will always perform the same number of passes and comparisons regardless of the initial array? The correct answer might include more than one algorithm.

8. Order the Selection Sort, Insertion Sort, and Bubble Sort algorithms according to their speed in sorting an array.

   _____ Selection Sort

   _____ Insertion Sort

   _____ Bubble Sort

B.2.2 Algorithm Games

This part provides statements about the algorithm games you just played. Read the following items and write the number that best expresses how you feel about the statement.

Please answer by choosing from 1-5: Strongly Disagree (1), Disagree (2), Neutral (3), Agree (4), Strongly Agree (5)

1. I continued to play the games out of curiosity. (_)

2. I felt discouraged while playing the games. (_)

3. I found the organization of information in the games confusing. (_)

4. I liked the graphics and images used in games. (_)

5. The organization of information in the games made sense to me. (_)

6. The screens layout of the games was visually pleasing. (_)

7. The games appealed to my visual senses. (_)

8. The games were attractive. (_)

9. The games were easy to play. (_)

10. The games offered me sufficient information about the algorithms. (_)

11. I consider my gaming experience a success. (_)

12. I was really drawn into the games. (_)

13. I would play the games again voluntarily. (_)

14. I would recommend playing the games to my friends. (_)

15. I would play games in the future to learn algorithms. (_)

16. Playing the games was worthwhile. (_)

Bibliography

Bibliography

[1] J. T. Stasko, M. H. Brown, and B. A. Price, Eds., *Software Visualization- Programming as a Multimedia Experience*. The MIT Press, 1997-1998.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Alorithms*, 2nd ed. MCGraw Hill, 2001.

[3] R. Baecker, "Sorting out sorting: A case study of software visualization for teaching computer science," in *Software Visualization: Programming as a Multimedia Experience*. The MIT Press, 1998, pp. 369–381.

[4] P. D. Eades and K. Zhang, Eds., *Software Visualization*. World Scientific, 1996, vol. 7.

[5] R. Baecker and D. Sherman, "Sorting out sorting," 30 minute colour sound film, Dynamic Graphics Project, University of Toronto, 1981, excerpted and reprinted in SIGGRAPH Video Review 7,1983.

[6] M. H. Brown and R. Sedgewick, "A system for algorithm animation," in *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*. ACM Press, 1984, pp. 177–186.

[7] G. Rossling and B. Freisleben, "Animal: A system for supporting multiple roles in algorithm animation," *Visual Languages and Computing*, vol. 13, no. 3, pp. 341–354, 2002.

[8] C. Hundhausen, J. Wingstrom, and R. Vatrapu, "The evolving user-centered design of the algorithm visualization storyboarder," in *VLHCC '04: Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing*. IEEE Computer Society, 2004, pp. 62–64.

[9] E. Carson, I. Parberry, and B. Jensen, "Algorithm explorer: visualizing algorithms in a 3d multimedia environment," in *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education,Covington*. ACM Press, 2007, pp. 155–159.

[10] A. Badre, M. Beranek, J. M. Morris, and J. Stasko, "Assessing program visualization systems as instructional aids," in *ICCAL '92: Proceedings of 4th International Conference on Computer Assisted Learning, Wolfville*. Springer-Verlag, 1992, pp. 87–99.

[11] J. Stasko, A. Badre, and C. Lewis, "Do algorithm animations assist learning?: an empirical study and analysis," in *CHI '93: Proceedings of the SIGCHI conference on Human factors in computing systems, Amsterdam*. ACM Press, 1993, pp. 61–66.

[12] M. D. Byrne, R. Catrambone, and J. T. Stasko, “Do algorithm animations aid learning?” Tech. Rep. GIT-GVU-96-18, 1996.

[13] C. Hundhausen, S. Douglas, and J. Stasko, “A meta-study of algorithm visualization effectiveness,” *Visual Languages and Computing*, vol. 13, no. 3, pp. 259–290, 2002.

[14] L. Stern, H. Sondergaard, and L. Naish, “A strategy for managing content complexity in algorithm animation,” in *ITiCSE ’99: Proceedings of the 4th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education, Cracow*. ACM Press, 1999, pp. 127–130.

[15] C. Hundhausen and S. Douglas, “Using visualizations to learn algorithms: Should students construct their own, or view an expertś?” *IEEE Symposium on Visual Languages*, vol. 0, p. 21, 2000.

[16] T. L. Naps, “Jhave: Supporting algorithm visualization,” *IEEE Computer Graphics and Applications*, vol. 25, no. 5, pp. 49–55, 2005.

[17] M. Wolf, *The Medium of the Video Game*, "1st" ed. University of Texas Press, 2002.

[18] S. Egenfeldt Nielsen, “Beyond edutainment exploring the educational potential of computer games,” Ph.D. dissertation, IT-University of Copenhagen, February 2005. [Online]. Available: http://www.itu.dk/people/sen/egenfeldt.pdf

[19] B. Chamberlin, “Creating entertaining games with educational content,” Ph.D. dissertation, University of Virginia, 2003.

[20] G. Rossling and T. L. Naps, “A test-bed for pedagogical requirements in algorithm visualizations,” in *ITiCSE ’02: Proceedings of the 7th annual conference on Innovation and technology in computer science education, Aarhus*. ACM Press, 2002, pp. 96–100.

[21] S. Grissom, M. F. McNally, and T. Naps, “Algorithm visualization in cs education: comparing levels of student engagement,” in *SoftVis ’03: Proceedings of the 2003 ACM symposium on Software visualization, San Diego*. ACM Press, 2003, pp. 87–94.

[22] L. P. Rieber, “Seriously play: Designing interactive learning environments based on the blending of microworlds, simulations and games.” *Educational Technology Research and Development*, vol. 44, no. 2, pp. 43–58, 1996.

[23] M. G. Jones, “Creating electronic learning environments: Games, flow. and the user interface,” in *National Convention of the Association for Educational Communications and Technology*, Houston, 1998.

[24] M. Csikszentmihalyi, *The evolving self. A psychology for the third millennium*. New York: Harper Collins, 1993.

[25] J. Gee, “Good games, good teaching: Interview with james gee.” *UW-Madison School of Education Online News*, 2004, accessed 09/03/2009. [Online]. Available: http://www.education.wisc.edu/news/details.asp?fldIdNews=64

[26] C. A. Shaffer, M. Cooper, and S. H. Edwards, "Algorithm visualization: a report on the state of the field," in *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education, Covington*. ACM Press, 2007, pp. 150–154.

[27] J. M. Randel, B. A. Morris, C. D. Wetzel, and B. V. Whitehill, "The effectiveness of games for educational purposes: a review of recent research," *Simul. Gaming*, vol. 23, no. 3, pp. 261–276, 1992.

[28] T. W. Malone, "What makes things fun to learn? heuristics for designing instructional computer games," in *SIGSMALL '80: Proceedings of the 3rd ACM SIGSMALL symposium and the first SIGPC symposium on Small systems, Palo Alto*. ACM Press, 1980, pp. 162–169.

[29] J. V. Dempsey, B. Lucassen, W. Gilley, and K. Rasmussen, "Since malone's theory of intrinsically motivating instruction: What's the score in the gaming literature?" *Journal of Educational Technology Systems*, vol. 22, no. 2, pp. 173–183, (1993).

[30] S. Thiagarajan and H. Stolovitch, *Instructional Simulation Games*, ser. Instructional Design Library. Educational Technology Pubns, February 1978, vol. 12.

[31] C. Garvey, *Play*. Cambridge: Harvard University Press, 1990.

[32] J. Gee, *What Video Games Have to Teach Us About Learning and Literacy*, 1st ed. New York: Palgrave Macmillan, 2003.

[33] S. Thiagarajan, "Laws of learning: 14 important principles every trainer should know," Workshops by Thiagi, Inc.–Website, June 2003, accessed 1/1/2010. [Online]. Available: http://www.thiagi.com/laws-of-learning.html

[34] S. Shabanah, J. Chen, H. Wechsler, E. Wegman, and D. Carr, "Designing computer games to teach algorithms," in *ITNG 2010: Proceedings of 7th International Conference on Information Technology: New Generations*. Las Vegas, Nevada, USA: IEEE Computer Society, 2010.

[35] T. L. Naps, G. Rossling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodger, and J. A. Velazquez-Iturbide, "Exploring the role of visualization and engagement in computer science education," in *ITiCSE-WGR '02: Working group reports from ITiCSE on Innovation and technology in computer science education, Aarhus*. ACM Press, 2002, pp. 131–152.

[36] S. Shabanah and J. X. Chen, "Simplifying algorithm learning using serious games," in *WCCCE '09: Proceedings of the 14th Western Canadian Conference on Computing Education*. Burnaby, BC, Canada: ACM, 2009, pp. 34–41.

[37] J. Simpson and C. Weiner, Eds., *The Oxford English Dictionary*, 2nd ed. Oxford University Press, 1989.

[38] R. Baecker and A. Marcus, *Human Factors and Typography for More Readable Programs*. Addison-Wesley, 1990.

[39] M. Kolling, B. Quig, A. Patterson, and J. Rosenberg, "The bluej system and its pedagogy," *Computer Science Education*, vol. 13, no. 4, pp. 249–268, 2003.

[40] B. A. Price, R. M. Baecker, and I. S. Small, "A principled taxonomy of software visualization," *Visual Languages and Computing*, vol. 4, no. 3, pp. 211–266, 1993.

[41] J. Haajanen, M. Pesonius, E. Sutinen, J. Tarhio, T. Terasvirta, and P. Vanninen, "Animation of user algorithms on the web," in *VL '97: Proceedings of the 1997 IEEE Symposium on Visual Languages, Washington*. IEEE Computer Society, 1997, p. 356.

[42] R. Levy, M. Ben-Ari, and P. Uronen, "The jeliot 2000 program animation system," *Computers and Education*, vol. 40, no. 1, pp. 15–21, 2003.

[43] T. D. Hendrix, J. H. Cross, and L. A. Barowski, "An extensible framework for providing dynamic data structure visualizations in a lightweight ide," in *SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education, Norfolk*. ACM, 2004, pp. 387–391.

[44] A. Korhonen, "Visual algorithm simulation," Doctoral dissertation (tech rep. no. tko-a40/03), Helsinki University of Technology, 2003.

[45] M. H. Brown, *Algorithm Animation*. The MIT Press, 1988.

[46] M. Brown, "Exploring algorithms using balsa-ii," *Computer*, vol. 21, no. 5, pp. 14–36, 1988.

[47] Brown, "Zeus: A system for algorithm animation and multi-view editing," in *Proceedings of 1991 IEEE Workshop on Visual Languages*, Oct 1991, pp. 4–9.

[48] M. Brown and J. Hershberger, "Color and sound in algorithm animation," *Computer*, vol. 25, no. 12, pp. 52–63, DEC 1992.

[49] M. H. Brown and M. A. Najork, "Algorithm animation using 3d interactive graphics," in *ACM Symposium on User Interface Software and Technology*, Nov 1993, pp. 93–100.

[50] J. Hyvonen and L.Malmi, "Trakla- a system for teaching algorithms using email and a graphical editor," in *Proceedings of HYPERMEDIA in Vaasa*, 1993, pp. 141–147.

[51] A. Korhonen and L. Malmi, "Algorithm simulation with automatic assessment," in *ITiCSE '00: Proceedings of the 5th annual SIGCSE/SIGCUE ITiCSEconference on Innovation and technology in computer science education, Helsinki*. ACM, 2000, pp. 160–163.

[52] A. Korhonen, L. Malmi, and R. Saikkonen, "Matrix - concept animation and algorithm simulation system," in *ITiCSE '01: Proceedings of the 6th annual conference on Innovation and technology in computer science education, Canterbury*. ACM, 2001, p. 180.

[53] V. Karavirta, A. Korhonen, L. Malmi, and K. Stalnacke, "Matrixpro - a tool for demonstrating data structures and algorithms ex tempore," in *ICALT '04: Proceedings of the IEEE International Conference on Advanced Learning Technologies, Washington*. IEEE Computer Society, 2004, pp. 892–893.

[54] M. Bäsken and S. Näher, "Geowin - a generic tool for interactive visualization of geometric algorithms," in *Revised Lectures on Software Visualization, International Seminar*. Springer-Verlag, 2002, pp. 88–100.

[55] G. Cattaneo, G. Italiano, and U. Ferraro-Petrillo, "Catai: Concurrent algorithms and data types animation over the internet," *Visual Languages and Computing*, vol. 13, no. 4, pp. 391–419, August 2002.

[56] J. T. Stasko, "Tango: A framework and system for algorithm animation," Providence, RI, USA, Tech. Rep., 1989.

[57] J. Stasko, "Animating algorithms with xtango," *SIGACT News*, vol. 23, no. 2, pp. 67–71, 1992.

[58] J. Stasko and E. Kraemer, "A methodology for building application-specific visualizations of parallel programs," *Parallel and Distributed Computing*, vol. 18, no. 2, pp. 258–264, 1993.

[59] J. Stasko and J. Wehrli, "Three-dimensional computation visualization," in *Proceedings of the 1993 IEEE Symposium on Visual Languages*, Aug 1993, pp. 100–107.

[60] J. T. Stasko, "Using student-built algorithm animations as learning aids," in *SIGCSE '97: Proceedings of the twenty-eighth SIGCSE technical symposium on Computer science education, San Jose*. ACM Press, 1997, pp. 25–29.

[61] C. Hundhausen and S. Douglas, "Salsa and alvis: A language and system for constructing and presenting low fidelity algorithm visualizations," vol. 0. IEEE Computer Society, 2000, p. 67.

[62] C. D. Hundhausen and J. L. Brown, "What you see is what you code: A radically dynamic algorithm visualization development model for novice learners," in *VLHCC '05: Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE Computer Society, 2005, pp. 163–170.

[63] C. Hundhausen, "The "algorithms studio" project: using sketch-based visualization technology to construct and discuss visual representations of algorithms," in *HCC '02: Proceedings of the IEEE 2002 Symposia on Human Centric Computing Languages and Environments, Arlington*. IEEE Computer Society, 2002, pp. 99–100.

[64] J. Baker, I. Cruz, G. Liotta, and R. Tamassia, "Algorithm animation over the world wide web," in *Proceedings of the International Workshop on Advanced Visual Interfaces*, May 1996, pp. 203–212.

[65] M. Brown and M. Najork, "Collaborative active textbooks: A web-based algorithm animation system for an electronic classroom," in *Proceedings of 1996 IEEE Symposium on Visual Languages*, Sept. 1996, pp. 266–275.

[66] M. Najork, "Web-based algorithm animation," in *DAC '01: Proceedings of the 38th conference on Design automation*. ACM Press, 2001, pp. 506–511.

[67] S. Hansen, N. Narayanan, and D. Schrimpsher, "Helping learners visualize and comprehend algorithms," *Computer- Enhanced Learning*, vol. 2, no. 1, 2000. [Online]. Available: http://imej.wfu.edu

[68] A. Kerren, T. M. ldner, and E. Shakshuki, "Novel algorithm explanation techniques for improving algorithm teaching," in *SoftVis '06: Proceedings of the 2006 ACM symposium on Software visualization, Brighton*. ACM Press, 2006, pp. 175–176.

[69] A. Tal and D. Dobkin, "Visualization of geometric algorithms," *IEEE Transactions on Visualization and Computer Graphics*, vol. 1, no. 2, pp. 194–204, 1995, 1077-2626.

[70] M. Shneerson and A. Tal, "Visualization of geometric algorithms in an electronic classroom," in *VIS '97: Proceedings of the 8th conference on Visualization '97, Phoenix*. IEEE Computer Society Press, 1997, pp. 455–ff.

[71] T. Hung and S. H. Rodger, "Increasing visualization and interaction in the automata theory course," in *In The proceedings of the 31st ACM SIGCSE Technical Symposium on Computer Science Education*. ACM press, 2000, pp. 6–10. [Online]. Available: http://citeseer.ist.psu.edu/hung00increasing.html

[72] V. Ciesielski and P. McDonald, "Using animation of state space algorithms to overcome student learning difficulties," in *ITiCSE '01: Proceedings of the 6th annual conference on Innovation and technology in computer science education,Canterbury*. ACM Press, 2001, pp. 97–100.

[73] N. Baloian, H. Breuer, and W. Luther, "Algorithm visualization using concept keyboards," in *SoftVis '05: Proceedings of the 2005 ACM symposium on Software visualization, St. Louis*. ACM Press, 2005, pp. 7–16.

[74] "World of warcraft," World of Warcraft Community Site– Website, accessed 10/15/2009. [Online]. Available: www.worldofwarcraft.com

[75] E. Avendon and B. Sutton-Smith, *The Study of Games*. New York: John Wiley and Sons, Inc., 1971.

[76] M. Prensky, *Digital Game-Based Learning*, 1st ed. McGraw-Hill Companies, December 2000.

[77] A. J. Faria, "Business simulation games: current usage levels - an update," *Simul. Gaming*, vol. 29, no. 3, pp. 295–308, 1998.

[78] P. Saettler, *A history of instructional technology*. New York: McGraw-Hill, 1968.

[79] S. L. Calvert, *Children's journeys through the information age*. Boston: McGraw-Hill, 1999.

[80] G. Loftus and E. Loftus, *Mind at Play: The Psychology of Video Games*. New York: Basic Books, 1983.

[81] D. Bitzer, P. Braunfeld, and W. Lichtenberger, "Plato: An automatic teaching device," Master's thesis, University of Illinois, Coordinated Science Laboratory, 1961.

[82] E. V. Meer, "Plato: From computer-based education to corporate social responsibility," Iterations-Website, November 2003, accessed 03/07/2008. [Online]. Available: http://www.cbi.umn.edu/iterations/vanmeer.html

[83] D. Rawitsch, "Oregon trail," The Learning Company–Website, accessed 11/26/2007. [Online]. Available: http://www.learningcompany.com/

[84] J. H. Dysentery, "The greatest games of all time," Game Spot–Website, 1997, accessed 10/26/2007. [Online]. Available: http://www.gamespot.com/gamespot/features/all/greatestgames/p-34.html

[85] "Basic math," MobyGames–Website, Atari, accessed 03/09/2008. [Online]. Available: http://www.mobygames.com/game/basic-math

[86] "Electric company math fun," Intellivision: Children's Learning network–Website, accessed 10/12/2009. [Online]. Available: http://www.intellivisionlives.com/bluesky/games/credits/learning.html#math

[87] "Rockys boots," Warren Robinett's Home Page–Website, 2004, accessed 10/12/2009. [Online]. Available: http://www.warrenrobinett.com/rockysboots/

[88] P. Relf and D. Kovacs, "The snooper troops," Spinnaker– Website, 1982, accessed 03/17/2008. [Online]. Available: http://www.the-underdogs.info/game.php?id=1436

[89] "In search of the most amazing thing," Spinnaker– Website, Tom Snyder Productions, 1983, accessed 03/17/2008. [Online]. Available: http://www.the-underdogs.info/game.php?id=1438

[90] K. Anderson, "Where in the world is carmen sandiego?" Classic Gaming-Games Museum–Website, accessed 03/05/2009. [Online]. Available: http://classicgaming.gamespy.com/View.php?view=GameMuseum.Detail&id=23

[91] "Where in the world is carmen sandiego?" Moby Games–Website, accessed 03/9/2008. [Online]. Available: http://www.mobygames.com/game/dos/where-in-the-world-is-carmen-sandiego

[92] O. Softscape, "The seven cities of gold," The House of Games–Website, accessed 03/16/2008. [Online]. Available: http://www.thehouseofgames.net/index.php?t=10&id=193

[93] "Robot odyssey," The Robot Odyssey Resource–Website, Last Updated 1/30/01, accessed 12/11/2009. [Online]. Available: http://members.aol.com/Fractal101/odyssey.htm#Anchor6

[94] "Mavis beacon," The Learning Company–Website, The Software Toolworks, accessed 03/11/2008. [Online]. Available: http://www.learningcompany.com/

[95] "Incredible machine series," Moby Games–Website, 1993, accessed 01/10/2008. [Online]. Available: http://www.mobygames.com/game-group/incredible-machine-series

[96] "Typing of the dead," Moby Games–Website, accessed 01/10/2009. [Online]. Available: http://www.mobygames.com/game/typing-of-the-dead

[97] "Reader rabbit series," Moby Games– Website, accessed 03/11/2008. [Online]. Available: http://www.mobygames.com/game/reader-rabbit

[98] "Gizmos and gadgets," Moby Games– Website, accessed 03/11/2008. [Online]. Available: http://www.mobygames.com/game/super-solvers-gizmos-gadgets

[99] S. Osterweil and C. Hancock, "Logical journey of the zoombinis-series," The Learning Company–Website, accessed 03/11/2008. [Online]. Available: http://www.learningcompany.com/

[100] "Clue finders series," Moby Games– Website, accessed 03/11/2008. [Online]. Available: http://www.mobygames.com/game/windows/cluefinders-4th-grade-adventures

[101] "Jumpstartseries," Knowledge Adventure–Website, Knowledge Adventure, accessed 02/15/2009. [Online]. Available: http://shop.knowledgeadventure.com/Departments/JumpStartSeries.aspx

[102] "Math mission," GZ Kids Zone– Website, accessed 03/17/2008. [Online]. Available: http://www.gzkidzone.com/gamesell/p22771.asp

[103] "Eduprofix," eduProfix– A Website, accessed 12/12/2009. [Online]. Available: http://www.eduprofix.com/index.htm

[104] "Bcompris," GCompris–Website, accessed 03/10/2008. [Online]. Available: http://gcompris.net/-en

[105] "Toontown," Disney World– Website, accessed 12/12/2007. [Online]. Available: http://www.toontown.com

[106] "Arcademic skill builders," Arcademic– Website, accessed 10/10/2009. [Online]. Available: http://arcademicskillbuilders.com/

[107] M. Casamassina, "Big brain academy: Wii degree review," IGN–Website, 2007, accessed 25/8/2007. [Online]. Available: http://uk.wii.ign.com/articles/795/795126p2.html

[108] "Serious games," Serious Games–Website, accessed 03/10/2008. [Online]. Available: http://www.seriousgames.org/index2.html.

[109] H. Jenkins, "The games to teach project," Comparative Media Studies-MIT–Website, 2001, accessed 03/10/2008. [Online]. Available: http://www.educationarcade.org/gtt/proto.html

[110] Jenkins, "The games to teach project prototypes," MIT–Website, 2001, accessed 03/17/2008. [Online]. Available: http://www.educationarcade.org/gtt/proto.html

[111] S. Barab, T. Dodge, M. Thomas, C. Jackson, and H. Tuzun, "Our designs and the social agendas they carry," *Learning Sciences*.

[112] "Quest atlantis," Indiana University–Website, accessed 02/02/2007. [Online]. Available: http://atlantis.crlt.indiana.edu

[113] C. Pelletier, "The making of games project," London Knowledge Lab– Website, 2003, accessed 03/17/2008. [Online]. Available: http://www.lkl.ac.uk/research/pelletier.html

[114] H. Kelly, K. Howell, E. Glinert, L. Holding, C. Swain, A. Burrowbridge, and M. Roper, "How to build serious games," *Communication ACM*, vol. 50, no. 7, pp. 44–49, 2007.

[115] "Colobot," EPSiTEC Games– Website, 2002, accessed 03/17/2008. [Online]. Available: http://www.colobot.com/

[116] "Plant," Genomics Digital Lab– Website, 2007, accessed 03/17/2008. [Online]. Available: http://www.genomicsdigitallab.com/home.html

[117] "Global conflict: Palestine," Serious Games Interactive– Website, 2007, accessed 03/17/2008. [Online]. Available: http://www.globalconflicts.eu/

[118] M. Curtin, N. Carpenter, and C. Ritzo, "Adding fun and games to training programs," in *SIGUCCS '06: Proceedings of the 34th annual ACM SIGUCCS conference on User services, Edmonton*. ACM Press, 2006, pp. 50–54.

[119] K. Claypool and M. Claypool, "Teaching software engineering through game design," in *ITiCSE '05: Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education, Monte de Caparica*. ACM Press, 2005, pp. 123–127.

[120] "Directx developer center," MSDN– Website, accessed 09/27/2009. [Online]. Available: http://msdn.microsoft.com/en-us/directx/default.aspx

[121] "About the opengl architecture review board," OpenGL.org–Website, accessed 09/27/2009. [Online]. Available: http://www.opengl.org/about/arb/overview.html;

[122] "Xna developer center," MSDN–Website, accessed 09/27/2009. [Online]. Available: http://msdn.microsoft.com/en-us/xna/default.aspx

[123] "Game maker," GameMaker.net– Website, accessed 09/27/2009. [Online]. Available: http://www.gamemaker.nl/

[124] "Gamexml," GameXML.org– Website, accessed 09/27/2009. [Online]. Available: http://www.gamexml.org/

[125] S. A. Wallace and A. Nierman, "Addressing the need for a java based game curriculum," *Computing Sciences in Colleges*, vol. 22, no. 2, pp. 20–26, 2006.

[126] "3d engines database," DevMaster.net– Website, accessed 09/27/2009. [Online]. Available: http://www.devmaster.net/engines

[127] "Ogre 3d: Open source graphics engine," Ogre3d.org–Website, accessed 09/27/2009. [Online]. Available: http://www.ogre3d.org;

[128] "Torque game engine," GarageGames.com– Website, accessed 09/27/2009. [Online]. Available: http://www.garagegames.com/pg/product/view.php?id=1;

[129] R. Coleman, S. Roebke, and L. Grayson, "Gedi: a game engine for teaching videogame design and programming," *Computing Sciences in Colleges*, vol. 21, no. 2, pp. 72–82, 2005.

[130] A. Furtado, "Sharpludus: Improving game development experience through software factories and domain-specific languages," Master's thesis, Federal University of Pernambuco, Recife, Brazil ,, 2006, mSc Thesis.

[131] J. Dobbe, "A domain-specific language for computer games," Master's thesis, Delft University of Technology, Delft, the Netherlands, 2006-2007.

[132] "Cannibal game studios," Cannibal Game Studios–Website, accessed 09/27/2009. [Online]. Available: http://www.cannibalgamestudios.com/

[133] W. Huitt and J. Humme, *Educational Psychology*. College of Education, 1999.

[134] R. S. Feldman, *Understanding Psychology*. McGraw-Hill, 1996.

[135] K. Crawford, "Vygotskian approaches to human development in the information era," *Educational Studies in Mathematics*, vol. 31, pp. 43–62, 1996.

[136] D. H. Bailey, "Constructivism and multimedia: Theory and application: Innovation and transformation," *International Journal of Instructional Media*, vol. 23, no. 2, pp. 161–166, 1996.

[137] L. J. Issing, "From instructional technology to multimedia didactics," *Educational Media International*, vol. 31, no. 3, pp. 171–182, 1994.

[138] M. Resnick, A. Bruckman, and F. Martin, "Constructional design: Creating new construction kits for kids," in *The design of children's technology*, A. Druin, Ed. San Francisco: Morgan Kaufmann Publishers, Inc, 1999, pp. 149–167.

[139] A. Hejdenberg, "The psychology behind games," Gamasutra– Website, April 26, 2005, accessed 10/10/2008. [Online]. Available: http://www.gamasutra.com/features/20050426/hejdenberg/_01.shtml

[140] S. B. Bloom, *Taxonomy of educational objectives*. Pearson Education, 1984.

[141] R. Gagne, L. Briggsm, and W. Wager, *Principles of Instructional Design*, 3rd ed. New York: Holt, Rinehart & Winston, 1988.

[142] S. Thiagarajan, "Instructional games, simulations, and role-play," *The ASTD Training and Development Handbook*, pp. 517–533, 1996.

[143] T. W. Malone, "Guidelines for designing educational computer programs," *Childhood Education*, vol. 59, no. 4, pp. 241–247, 1983.

[144] T. H. Apperley, "Genre and game studies: Toward a critical approach to video game," University of Melbourne– Website, accessed 11-05-2008. [Online]. Available: http://www.culture-communication.unimelb.edu.au/research-students/tom-apperley.pdf

[145] J. Schell, *The Art of Game Design: A book of lenses*. Morgan Kaufmann, 2008.

[146] M. Packard, "A crash course in game design and production," Lord Generic Productions– Website, 1996-2001, accessed 10/26/2008. [Online]. Available: http://www.berighteous.com/euphoria/1.html

[147] C. Carter, *Microsoft XNA Game Studio 3.0 Unleashed*, 2nd ed. Indiana,USA: SAMS, 2009.

[148] S. Kent, *The Ultimate History of Video Games: From Pong to Pokemon–The Story Behind the Craze That Touched Our Lives and Changed the World*. Three Rivers Press, 2001.

[149] E. L. Wynters, "3d video games: no programming required," *Computing Sciences in Colleges*, vol. 22, no. 3, pp. 105–111, 2007.

[150] R. C. Hoetzlein and D. I. Schwartz, "Gamex: a platform for incremental instruction in computer graphics and game design," in *SIGGRAPH '05: ACM SIGGRAPH 2005 Educators program, Los Angeles*. ACM Press, 2005, p. 36.

[151] A. Rollings and D. Morris, *Game Architecture and Design*. The Coriolis Group, 2000.

Curriculum Vitae

Sahar Siraj Shabanah was born on March 12, 1968, in Jeddah, Saudi Arabia, and in a Saudi citizen. She graduated from the Seventh High School, Jeddah, Saudi Arabia, in 1985. She received her Bachelor of Science in Computer Science from King Abdulaziz University in 1990. She has worked as an instructor at the Computer Science Department at King Abdulaziz University Since 1992. At 1997 she won a scholarship to continue her graduate study at the United States. She attended George Washington University and received a Master of Science degree in Computer Science/Multimedia and Graphics in 2001. Since 2002 she has been a graduate student at George Mason University.