Handbook of Research on Improving Learning and Motivation through Educational Games:

Multidisciplinary Approaches

Patrick Felicia
Waterford Institute of Technology, Ireland

Volume I

Information Science
REFERENCE

Senior Editorial Director: Kristin Klinger
Director of Book Publications: Julia Mosemann
Editorial Director: Lindsay Johnston
Acquisitions Editor: Erika Carter
Development Editor: Myla Harty
Production Coordinator: Jamie Snavely
Typesetters: Michael Brehm and Milan Vracarich, Jr.
Cover Design: Nick Newcomer

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com/reference

Library of Congress Cataloging-in-Publication Data

Handbook of Research on Improving Learning and Motivation Through Educational Games : Multidisciplinary Approaches / Patrick Felicia, editor.
p. cm.
Includes bibliographical references and index.
Summary: “This book provides relevant theoretical frameworks and the latest empirical research findings on game-based learning to help readers who want to improve their understanding of the important roles and applications of educational games in terms of teaching strategies, instructional design, educational psychology and game design”--Provided by publisher.
ISBN 978-1-60960-495-0 (hardcover) -- ISBN 978-1-60960-496-7 (ebook) 1. Educational games. 2. Simulation games in education. 3. Cognitive learning. 4. Learning, Psychology of. I. Felicia, Patrick.
LB1029.G3H36 2011
371.33’7--dc22
2010054437

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Chapter 48
Computer Games for Algorithm Learning

Sahar Shabanah
King Abdul-Aziz University, Saudi Arabia

ABSTRACT

Data structures and algorithms are important foundation topics in computer science education. However, they are often complex and hard to understand. Therefore, this chapter introduces a new learning strategy that benefits from computer games' popularity and engagement to help students understand algorithms better by designing computer games that visualize algorithms. To teach an algorithm, an educational computer game, namely an Algorithm Game must have a game-play that simulates the behavior of the visualized algorithm and graphics depict the features of its data structure. Algorithm games attract students to learn algorithm using active engagement, enjoyment, and internal motivation. Algorithm Games attributes and genres that make them suitable to visualize algorithms have been specified. Various concepts in computer game design have been applied in the development of several algorithm games prototypes for algorithms, such as Binary Search, Bubble Sort, Insertion Sort, Selection Sort, Linked List, and Binary Search Tree Operations.

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*" (Corner & Leiserson, 2001). Algorithms are at the heart of every nontrivial computer application. Every computer scientist, every professional programmer, and every computer science student should know about basic algorithms and generic data structures. Moreover, teaching algorithms is one of the main activities that takes place in the most of computer science classes.

DOI: 10.4018/978-1-60960-495-0.ch048

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 challenging 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 (Baecker, 1998). Researchers have been trying to find the best way to learn and teach algorithms. One of the best known approaches, known as *Algorithm Visualization,* uses graphics, sounds, and animations to communicate how algorithms work (Eades & Zhang, 1996).

Since the production of the movie *Sorting Out Sorting* (Baecker & Sherman, 1981), a great number of algorithm visualization systems have been built with the promise of improving algorithm learning. However, 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., 1992), 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 on algorithm visualization systems efficiency concluded that algorithm animations have unreliable effects on student understanding of algorithms (Stasko et al., 1993). In addition, (Byrne et al., 1996) state, "the benefits of animations are not obvious." In a meta-study of over twenty empirical experiments conducted to assess existing algorithm visualization systems, (Hundhausen & Brown, 2002) 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 (Stern et al., 1999). 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 (Hundhausen et al., 2002) since "what the students do, not what they see, has the greater impact on learning" (Hundhausen & Douglas, 2000). Moreover, the more learners interact with algorithm visualization systems, the better they understand the algorithms these systems visualize (Naps, 2005).

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* (Wolf, 2002). Despite the frustration with educational computer games in the past; recently, they re-emerged as *Serious Games*, which have serious purposes, such as training, advertising, simulation, and education (Nielsen, 2005). (Chamberlin, 2003) describes computer games to have many roles, such as "tutoring, amusement, exploring new skills, promoting self-esteem, drill and practice, or creating a change in attitudes." Computer games fully interact with players and encourage them to think and act. Furthermore, according to (Rieber, 1996), "playing computer games involves active engagement" because computer games support all components of flow (Jones, 1998) as defined by (Csikszentmihalyi, 1993). 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 (Gee, 2004).

Given that there is an increasing correlation between the level of student learning and engagement in algorithm visualization systems (Grissom et al., 2003), and given that computer games fully interact with players, we address the shortness in current algorithm visualization systems by developing computer games, namely *Algorithm Games* that visualize and teach algorithms. Our approach benefits from the computer games active engagement, intrinsic motivation, popularity, and entertainment to maximally motivate and engage students in algorithm learning.

In the remaining of this chapter: the motivation of this research is explained; then, several algorithm visualization systems are compared according to their level of engagement including the proposed approach. Next, algorithm games are defined and specified in more details. Moreover, concepts related to computer games design in general are described, and how those terms can be applied to design our Algorithm Games is illustrated. Finally, example prototypes of algorithm games are outlined.

MOTIVATION

Is there a need for another Algorithm Visualization system? Over twenty years, many algorithm visualization systems have been produced. However, in their effort to build a wiki for current algorithm visualization systems, (Shaffer et al., 2007) 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.* 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 use 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, the typical college student spends half the time reading that s/he spends on playing computer games (Randel et al., 1992). Therefore, the use of computer games for teaching today's students is a good method.
- 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, 1980) stress on the importance of Intrinsic motivation in improving learning. Moreover, (Dempsey et al., 1993) state, "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. (Thiagarajan, 1978) 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, students understanding of algorithms can be evaluated using computer games

that simulate and imitate the behavior of those algorithms.

- Computer games utilize entertainment. (Garvey, 1990) defines playing games as "pleasurable, spontaneous, and voluntary." Moreover, (Gee, 2003) describes the knowledge earned by playing games as "the cycle of expertise" that gives people pleasure. Importantly, (Chamberlin, 2003) 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 (Thiagarajan, 2003).

RELATED WORK

(Naps et al.,2002) 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, we review some examples of current algorithm visualization systems by classifying them according to the type of engagement they support.

Sorting Out Sorting (SOS) is the first well-known algorithm visualization system, which has been produced by (Baecker & Sherman,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 items. However, *Brown Algorithm Simulator and Animator (BALSA)*, which was created in 1984 by (Brown & Sedgewick, 1984), 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: Display Primitives that allow the user to create, size, and position the algorithm and Interpretive Primitives that allow the user to start, stop, slow down, or even run an algorithm backward. A recent 2007 algorithm visualization system *Algorithm Explorer*, uses three-dimensional objects to represent common data structures and animations to visualize algorithms, which 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 represent data blocks and arrays as 3D boxes (Block World), graphs as spherical nodes (Graph World), and recursion and general function tracing (Stack World) (Carson et al., 2007). *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 (Naps et al., 2002).

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 (Hyvonen &.Malmi., 1993). In 2001, *Matrix* 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 (Korhonen et al., 2001). *Trakla2* (2003) is a newer version of *Trakla* that supports the creation and publishing of interactive exercises for data structures and algorithms. *MatrixPro* (2004) is a successor of *Matrix*, which was built to provide instructors with an easy usage of the functionalities of *Matrix* with lecture support and easy demonstration (Karavirta et al., 2004). 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 (video recorder buttons)-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 (Naps, 2005).

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 (Bäsken & Näher, 2002). 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 (Cattaneo & Ferraro-Petrillo, 2002).

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 (Stasko, 1992). In 1993, Stasko developed two more systems: *Polka* to visualize parallel algorithms (Stasko & Kraemer, 1993) and *Polka3D* to explore the use of 3D graphics (Stasko & Wehrli, 1993). 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 parameterized 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 (Stasko, 1997). *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 (Rossling & Freisleben, 2002). *Algorithm Visualization Storyboarder (ALVIS)* is an interactive environment was built by (Hundhausen et al., 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. *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 (Hundhausen & Brown, 2005).

A project called *Algorithm Studio*, which has been designed by (Hundhausen, 2002), 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.

Alternatively, 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. In addition, 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. Moreover, according to (Rieber, 1996) "play is usually voluntary, holds intrinsic motivation, involves active engagement, and contains a make-believe quality." Therefore, we introduce "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.*

ALGORITHM GAME SPECIFICATION

Algorithm Game Definition

(Thiagarajan, 1996) 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" (Thiagarajan, 1996). 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 a simulation instructional game that has been designed to teach and visualize an algorithm by imitating the behavior of the algorithm and by using graphics that portray its data structure features.

Algorithm Game Attributes

(Thiagarajan, 1996) 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 hardness 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, 1983) 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, we define the following common attributes and features for our algorithm games:

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, we have modified the Pong game to visualize the binary search algorithm by making one player uses a ball and a paddle to hit a set of boxes instead of two players hitting the ball against each other (Section Binary Search Game Prototype). On the other hand, we have created a new game to visualize the minimum spanning tree of a graph, which requires the player to travel between several cities in the least time.
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 have several levels with increasing difficulties.
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, we can use a deck of cards to visualize it since the cards have values and can be arranged sequentially to simulate the array elements; for the tree data structure, we can use an actual tree with leaves that represent the numbers.
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 Prototype (last section) does not allow the player to uncover more than two cards at the same time or swap non-adjacent cards. In the Binary Search Game (last section) if the player hits a non-middle box, he loses one point.
8. To simplify the student assessment, the algorithm game must keep a record for each player progress in the game. Therefore, each algorithm game prototype that have been

described in this chapter 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 have been described in Chapter includes a High Scores screens that displays the most five high scores achieved among all players.

Algorithm Games Genres

Game genre distinguishes one type of game-play from another. Genres are mainly focused one style of interaction (Apperley) and thus provide a good basis to find out whether the interaction type influences the general structure of a game. In this section, we will give an overview of the most important game genres by presenting a high-level overview of the different game genres (Crawford, 1996). The most important game genres are as follows:

- **Ball and Paddle Games:** a player controls a paddle object that moves back and forth on a single axis, such as Pong game.
- **Bin ball 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 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.
- **Simulation Games:** games imitate various aspects of the life such as vehicle simulation, sports simulation, card simulation, biological simulation etc.
- **Adventure Games:** players solve various puzzles by interacting with people or the environment.
- **Role-playing Game:** the game-play is centered on one or more avatars with characteristics that evolve over the course of the game.
- **Strategy Games:** the game-play requires careful and skillful thinking and planning in order to achieve victory.
- **Board Games:** games which are similar to board games in their design and play even if they did not previously exist as board games, such as Backgammon, Othello, Checkers, and Chess.

(Thiagarajan, 1996) 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, in last section, we give prototypes of algorithm games that have been designed as board games, such as Bubble Sort and Insertion Sort Games in addition to algorithm games that have been designed as ball and paddle games, such as Binary Search Game.

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 interact with each other.

Educational Games Design

At the end of her research on educational games effects, and after reviewing many theories on designing such games, (Chamberlin, 2003) 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 supports the players' advancement during the game. The use of scores and rewards can support players 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 exaggerate it.
4. Games should engage users with activity: player involvement and decision making within the game should be increased.
5. Build challenge into 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 users' engagement, but also to 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 during as design associates.

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

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 (xna), 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 (Schell, 2008) 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 behavior 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 (action, adventure).

Game Design Document

Prepared by game designers, the game 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 a 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 (Dobbe, 2007).

Game General Design Elements

In a crash course about game design, (Packard, 2001) defines eight general game design elements that can be used to describe a game prototype:

1. **Game Idea:** a game must have an idea.
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 an 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 gauge 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?

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 screen. They can be 2D Sprites or 3D Models. Each character 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 event that triggers them, they fall into three categories:
   a. **Theme Music:** five to ten or more seconds set the tone of the game. They are played at and during major milestone events, such as TitleSequence, StartNewGame, NewLevel, and YouDie.
   b. **Background Music:** begin at milestone events, like StartGame, or Intermission, and continue 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 isn't really Milestone related, but is important enough that a sound effect need 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 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 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 texture files define the 2D sprites of the game, model files define the 3D models of game and their effect files, which specify their appearance, font files define the properties of the fonts used in the game, and sound files determine the sound clips used in game.

Game Mechanics Design

Game mechanics describe flow of the game, 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 includes the game user interface design, the game screens structure, and the game-play rules.

Game Properties

What are the game name, number of levels, number of the player lives, level time? How the current level number, remaining lives, player score, and level timer are 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 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 a Main Menu screen. Then depending on the player choice, the Playing 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 a state a 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 (Carter, 2009).

Game-Play

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.

ALGORITHM GAME DESIGN

Based on game design document, game general design elements, game appearance design, and game mechanics design sections; this section introduces and defines 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.

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= zero, 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.

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:

- **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.

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, such as name of the game, maximum number of levels, maximum number of the player lives, and maximum level time. Other game attributes are calculated and displayed to show the game current state such as player remaining lives, current level timer, player current 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;
    1. 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;
    1. 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 to 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. **Pause Screen:** allows the player to stop and resume the game at any time.
7. **Exit Screen:** allows the player to stop and end the game at any time.
8. **Won Level Screen:** displays a "Level Completed" message when the player wins one level and asks the player to continue the game; then, it displays the next level for the player.
9. **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.

10. **Won Game Screen:** displays a won message when the player wins the game.
11. **Lost Game Screen:** displays a lost message when the player loses the game.
12. **Options Screen:** displays the game options for the player.
13. **Game Demo Screen:** displays the self- running demo of the game.
14. **High Scores Screen:** shows the five high scores achieved during the game by all players.
15. **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.
16. **Credits Screen:** shows the game developers names.

ALGORITHM GAMES PROTOTYPES

This section 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.

Binary Search Game Prototype

The binary search game is based on the original Pong game, where the player hits a ball character who moves around with a paddle (Kent, 2001). Binary Search game, 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. The binary search Algorithm finds the index of a specific value (Search Number) in a sequential list of sorted elements (array*)* by selecting the middle element (median) of the array and comparing it with the Search Number, then:

1. if (median > Search Number*)*, the index of the median-1 becomes the new upper bound of the list;
2. else if (median < Search Number), the index of the median+1 becomes the new lower bound;
3. else if (median = Search Number), 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 Search Number index or else determines that it is not in the list.

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

Play Screen

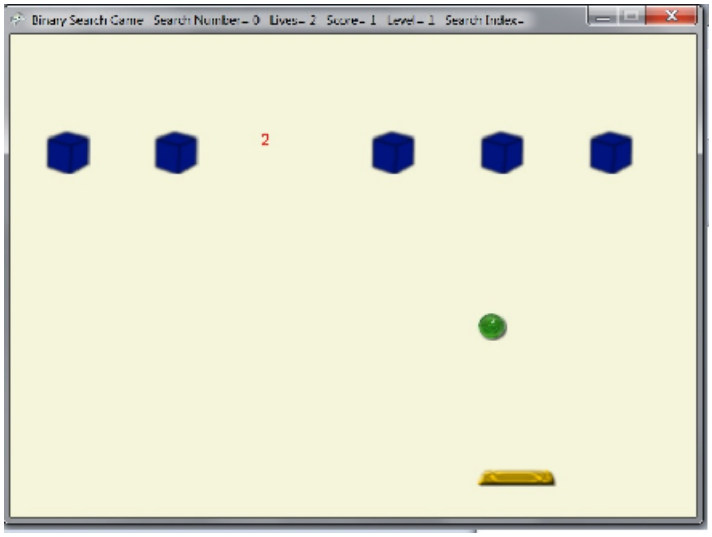
- Figure 1 shows the Play screen with the middle block value shown after hit by the player using the Ball and the Paddle.
- Figure 2 shows other screen shot of the game, the HUD items are Search Number= 16, Score=7, and Level=2. The progress of the game as follows:
  - When player hit (block=7) <Search Number, the player went left,
  - then when hit (block=11) <Search Number, the player went left,
  - then when hit (block =13) <Search number, the player went left.

Binary Search Tree Game Prototype

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 lesser than the values of its right sub-tree and greater than the left sub-tree values. The binary search tree operations are search, add value, and remove value algorithms.

- Search Algorithm

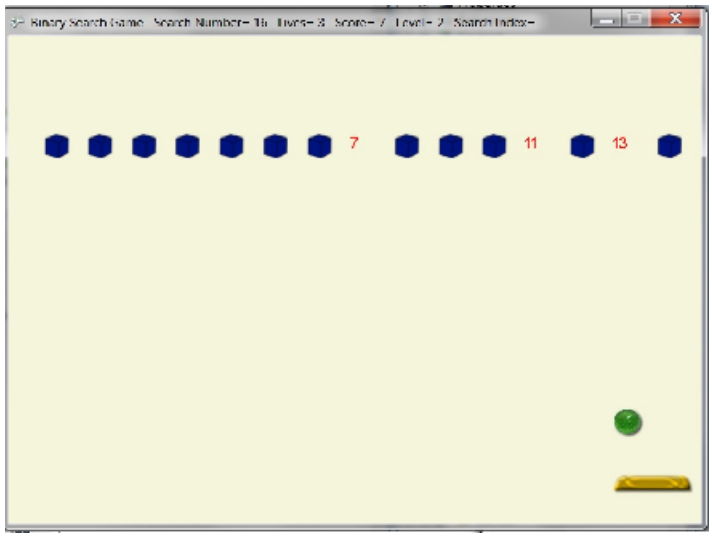
Figure 1. Play Screen (1) of Binary Search Game



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

Figure 2. Play Screen (2) of Binary Search Game



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 sub-tree) 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 sub-tree, replaces the value of the node to be removed with found minimum. Now, right sub-tree contains a duplicate, so the algorithm applies remove to the right sub-tree to remove the duplicate.

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 Graphics Items:
    - Nodes of the tree, each node having a value.
- Game Assets:
    - **Textures:** Node.
    - **Sounds:** AddNode, RemoveNode, Won, and Lost sounds.
    - **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.

At each new level, the number of the nodes, Level Timer, and Level Number are increased.

The Linked List Game Prototype

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. Operations on a single-linked list are traversal, inserting 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.

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 Graphics Items:
  - A pack of dominoes with their values covered.
- Game Assets:
  - Textures: 28 dominoes textures.
  - Sounds: Won, and Lost sounds.
  - 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.

Bubble Sort Game Prototype

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.

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 Graphics 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: LostLive, Won, and Lost sounds.
  - Fonts: Arial.

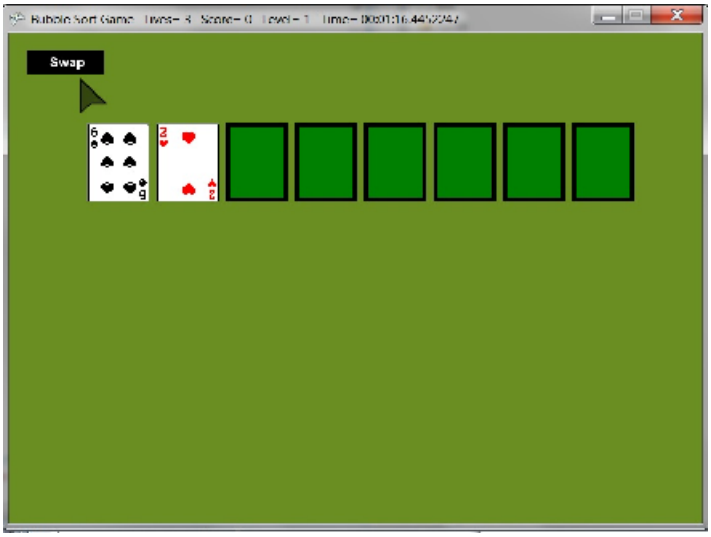
Third– Game Mechanics Design

- **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=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.

Play Screen Shots

- Figure 3 shows the Play screen after the player uncovered the first two adjacent cards, HUD Score=0, since the second card value less than the first card value, the player must click on Swap button to swap the cards.

Figure 3. Play Screen of Bubble Sort Game shows Card Swap Operation (step 1)



- Figure 4 shows the cards after have been swapped, HUD Score =5 and decreased Time.

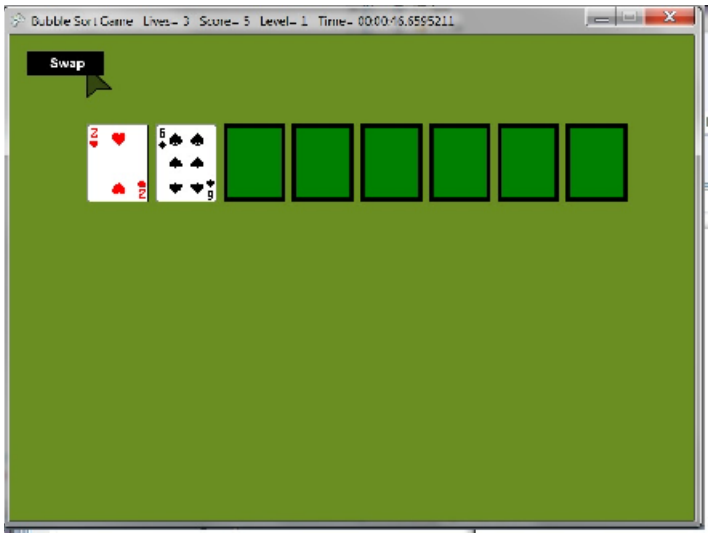
Selection Sort Game Prototype

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).

First– General Game Design Elements

Same as Bubble Sort Game.

Figure 4. Play Screen of Bubble Sort Game shows Card Swap Operation (step 2)



Second– Game Appearance Design

Same as Bubble Sort Game.

Third– Game Mechanics Design

Same as Bubble Sort Game, except for game-play.

- **Game Play:** the game play simulates the algorithm steps as follows:
  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 one; 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.
     a. If (player Lives > zero), the player repeats the same level and the player Lives is decreased by one.
     b. Else the player loses the game.
  4. If completes the level in the specified time, the player goes to the next level where the number of the cards in the deck, level timer value, and level number are increased.
  5. If completes all levels on time, the player wins the game and the player score is displayed.

Insertion Sort Game Prototype

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 already-sorted list).

First– General Game Design Elements

Same as Bubble Sort Game.

Second- Game Appearance Design

Same as Bubble Sort Game.

Third- Game Mechanics Design

Same as Bubble Sort Game, except for game-play.

- **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.

CONCLUSION

Algorithm games are part of algorithm visualization approach, namely AVuSG (Algorithm Visualization using Serious Games), that has been defined in (Shabanah & Chen, 2009). AVuSG produces three visualization forms: 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. Moreover, 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 by including several viewing and development tools that support the creation and viewing of the three algorithm visualization forms (text, flowchart, and game), which are produced by AVuSG (Shabanah et al., 2010). 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.

In the future, 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.

Algorithm games benefit from the players' desire to win, love to compete and entertaining resulted from playing to motivate students learning algorithms. Algorithm games provide richer visualizations that take better advantage of modern graphics and audio technology. These visualizations are designed to improve student's learning experience by creating a more engaging and immersive environment. By using algorithm games to visualize algorithms, we have introduced "playing" as a new form of algorithm visualization engagement that maximally engages students and combines all five forms of active engagement. Moreover, we facilitate the student's assessment using the algorithm game winning-losing criteria without the need for external questions.

REFERENCES

Apperley, T. H. (2008). *Genre and game studies: Toward a critical approach to video game*. University of Melbourne.

Badre, A., Beranek, M., Morris, J. M., & Stasko, J. (1992). Assessing program visualization systems as instructional aids. In *ICCAL '92: Proceedings of 4th International Conference on Computer Assisted Learning, Wolfville*, (pp. 87–99). Springer-Verlag.

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

Baecker, R., & Sherman, D. (1981). *Sorting out sorting. 30 minute colour sound film. SIGGRAPH Video Review, 7. Dynamic Graphics Project*. University of Toronto.

Bäsken, M., & Näher, S. (2002). Geowin-a generic tool for interactive visualization of geometric algorithms. In *Revised Lectures on Software Visualization, International Seminar*, (pp. 88–100). Springer-Verlag.

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

Byrne, M. D., Catrambone, R., & Stasko, J. T. (1996). *Do algorithm animations aid learning?* (Technical Report GIT-GVU-96-18).

Carson, E., Parberry, I., & Jensen, B. (2007). Algorithm explorer: Visualizing algorithms in a 3d multimedia environment. In *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education, Covington*, (pp. 155–159). ACM Press.

Carter, C. (2009). *Microsoft XNA game studio 3.0 unleashed*. Indiana: SAMS.

Cattaneo, G. I., & Ferraro-Petrillo, U. (2002). Catai: Concurrent algorithms and data types animation over the internet. *Visual Languages and Computing*, *13*(4), 391–419. doi:10.1006/jvlc.2002.0230

Chamberlin, B. (2003). *Creating entertaining games with educational content*. Unpublished doctoral thesis, University of Virginia.

Crawford, K. (1996). Vygotskian approaches to human development in the information era. *Educational Studies in Mathematics*, *31*, 43–62. doi:10.1007/BF00143926

Csikszentmihalyi, M. (1993). *The evolving self. A psychology for the third millennium*. New York: Harper Collins.

Dempsey, J. V., Lucassen, B., Gilley, W., & Rasmussen, K. (1993). Since Malone's theory of intrinsically motivating instruction: What's the score in the gaming literature? *Journal of Educational Technology Systems*, *22*(2), 173–183.

Dobbe, J. (2006-2007). *A domain-specific language for computer games*. Unpublished Master's thesis, Delft University of Technology, Delft, the Netherlands.

Eades, P. D., & Zhang, K. (Eds.). (1996). *Software visualization* (*Vol. 7*). World Scientific.

Egenfeldt Nielsen, S. (2005). *Beyond edutainment: Exploring the educational potential of computer games*. Unpublished doctoral thesis, IT-University of Copenhagen.

Garvey, C. (1990). *Play*. Cambridge, MA: Harvard University Press.

Gee, J. (2003). *What video games have to teach us about learning and literacy*. New York: Palgrave Macmillan.

Gee, J. (2004). *Good games, good teaching: Interview with James Gee*. UW-Madison School of Education Online News.

Grissom, S., McNally, M. F., & Naps, T. (2003). Algorithm visualization in CS education: Comparing levels of student engagement. In *SoftVis '03: Proceedings of the 2003 ACM symposium on Software visualization, San Diego*, (pp. 87–94). ACM Press.

Hundhausen, C. (2002). 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*, (pp. 99–100). IEEE Computer Society.

Hundhausen, C., & Douglas, S. (2000). *Using visualizations to learn algorithms: Should students construct their own, or view an expert's?* (p. 21). IEEE Symposium on Visual Languages.

Hundhausen, C., Douglas, S., & Stasko, J. (2002). A meta-study of algorithm visualization effectiveness. *Visual Languages and Computing*, *13*(3), 259–290. doi:10.1006/jvlc.2002.0237

Hundhausen, C., Wingstrom, J., & Vatrapu, R. (2004). 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*, (pp. 62–64). IEEE Computer Society.

Hundhausen, C. D., & Brown, J. L. (2005). 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*, (pp. 163–170). IEEE Computer Society.

Hyvonen, J., & Malmi, L. (1993). Trakla-a system for teaching algorithms using email and a graphical editor. In *Proceedings of HYPERMEDIA in Vaasa*, (pp. 141–147).

Jones, M. G. (1998). *Creating electronic learning environments: Games, flow, and the user interface*. In National Convention of the Association for Educational Communications and Technology, Houston.

Karavirta, V., Korhonen, A., Malmi, L., & Stalnacke, K. (2004). 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*, (pp. 892–893). IEEE Computer Society.

Kent, S. (2001). *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.

Korhonen, A., Malmi, L., & Saikkonen, R. (2001). 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*, (p. 180). ACM.

Malone, T. W. (1980). 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*, (pp. 162–169). ACM Press.

Malone, T. W. (1983). Guidelines for designing educational computer programs. *Childhood Education*, *59*(4), 241–247.

Naps, T. L. (2005). Jhave: Supporting algorithm visualization. *IEEE Computer Graphics and Applications*, *25*(5), 49–55. doi:10.1109/MCG.2005.110

Naps, T. L., Rossling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., et al. (2002). 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*, (pp 131–152). ACM Press.

Packard, M. (1996-2001). *A crash course in game design and production*. Lord Generic Productions– Website.

Randel, J. M., Morris, B. A., Wetzel, C. D., & Whitehill, B. V. (1992). The effectiveness of games for educational purposes: A review of recent research. *Simulation & Gaming*, *23*(3), 261–276. doi:10.1177/1046878192233001

Rieber, L. P. (1996). Seriously play: Designing interactive learning environments based on the blending of microworlds, simulations and games. *Educational Technology Research and Development*, *44*(2), 43–58. doi:10.1007/BF02300540

Rossling, G., & Freisleben, B. (2002). Animal: A system for supporting multiple roles in algorithm animation. *Visual Languages and Computing*, *13*(3), 341–354. doi:10.1006/jvlc.2002.0239

Schell, J. (2008). *The art of game design: A book of lenses*. Morgan Kaufmann.

Shabanah, S., Chen, J., Wechsler, H., Wegman, E., & Carr, D. (2010). Designing computer games to teach algorithms. In *ITNG 2010: Proceedings of 7th International Conference on Information Technology: New Generations. Las Vegas,* IEEE Computer Society.

Shabanah, S., & Chen, J. X. (2009). Simplifying algorithm learning using serious games. In *WCCCE '09: Proceedings of the 14th Western Canadian Conference on Computing Education, Burnaby, (*pp. 34–41). ACM Press.

Shaffer, C. A., Cooper, M., & Edwards, S. H. (2007). 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*, (pp. 150–154). ACM Press.

Sivasailam Thiagarajan, H. D. S. (1978). *Instructional simulation games instructional design library*. Educational Technology Pubs.

Stasko, J. (1992). Animating algorithms with xtango. *SIGACT News*, *23*(2), 67–71. doi:10.1145/130956.130959

Stasko, J., Badre, A., & Lewis, C. (1993). 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*, (pp. 61–66). ACM Press.

Stasko, J., & Kraemer, E. (1993). A methodology for building application-specific visualizations of parallel programs. *Parallel and Distributed Computing*, *18*(2), 258–264. doi:10.1006/jpdc.1993.1062

Stasko, J., & Wehrli, J. (1993). Three-dimensional computation visualization. In *Proceedings of the 1993 IEEE Symposium on Visual Languages*, (pp. 100–107).

Stasko, J. T. (1997). 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*, (pp. 25–29). ACM Press.

Stern, L., Sondergaard, H., & Naish, L. (1999). 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*, (pp. 127–130). ACM Press.

Thiagarajan, S. (1996). Instructional games, simulations, and role-play. In *The ASTD training and development handbook*. (pp. 517–533).

Thiagarajan, S. (2003). *Laws of learning: 14 important principles every trainer should know. Workshops by Thiagi, Inc. Cormen, T.H. & Leiserson, C.E. (2001). Introduction to algorithms*. McGraw Hill.

Wolf, M. (2002). *The medium of the video game*. University of Texas Press.

KEY TERMS AND DEFINITIONS

Algorithms: A sequence of computational steps that transform the input into the output.

Data Structure: A data structure in computer science is a way of storing data to be used efficiently.

Algorithm Visualization: Showing all the states of the data structures during the execution of an algorithm.

Computer Games: 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.

Educational Games: Computer games that involve learning of certain knowledge.

Games Design Document: Prepared by game designers, the design document contains information about the core elements that make a game.

Algorithm Games: Computer games with a game-play that simulates the behavior of the visualized algorithm and graphics depict the features of its data structure.

Push: Insert an item into the top of a stack data structure.

Pop: Delete an item from the top of a stack data structure.

RPG: A role-playing game in which players assume the roles of characters.

VCR: Video recorded.