2010 Seventh International Conference on Information Technology

Designing Computer Games to Teach Algorithms

Sahar S. Shabanah
George Mason University
sshaban1@gmu.edu

Dr. Jim X. Chen
George Mason University
jchen@gmu.edu

Dr. Harry Wechsler
George Mason University
wechsler@cs.gmu.edu

Dr. Daniel Carr
George Mason University
dcarr@gmu.edu

Dr. Edward Wegman
George Mason University
ewegman@gmu.edu

Abstract

Data structures and algorithms are important foundation topics in computer science education. However, they are often complex and hard to understand. Therefore, we introduce 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.

Key Words: algorithm learning, algorithm visualization, computer science education, educational games, game design.

1. Introduction

Despite being major components in computer science, algorithms are hard to comprehend because they usually either model complicated concepts, or refer to abstract mathematical notions, or describe complex dynamic changes in data structures to solve relatively difficult problems. Therefore, teaching algorithms is a challenging task that faces computer science instructors and teaching aids other than chalkboard and viewgraph are always needed to help students learn and understand algorithms better [1]. 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, which has been identified as *Algorithm Visualization.* Particularly, *Algorithm Visualization Systems* are systems that use graphics, sounds, and animations to communicate how algorithms work [2]. Since the production of the movie *Sorting Out Sorting* [3] in 1981, a great number of algorithm visualization systems have been built with the promise of improving algorithm learning. However, many researchers who conducted experiments to determine the efficiency of current algorithm visualization systems in teaching algorithms, reported unpromising results [4]. Some researchers found that there is no significant difference, in educational outcomes, between students who use visualizations and those who do not [5]. Moreover, in a recent effort to build a wiki for existing algorithm visualization systems, Shaffer *et al.* searched and analyzed hundreds of visualization systems and found that "most existing algorithm visualization systems are of low quality and the content coverage is skewed heavily toward easier topics [6]." Thus, researchers remain positive about visualizations in general as Shaffer *et al.* states "while many good algorithm visualization systems are available, the need for more and higher quality visualization systems continues. There are many topics for which no satisfactory visualization systems are available. Yet, there seems to be less activity in terms of creating new visualization systems now than at any time within the past ten years [6]." The focus on graphics and sound instead of teaching aspects in the design of many algorithm visualization systems is responsible for their unsatisfied pledge as helpful educational tools [7]. Despite being the most effective factor in the success of any algorithm visualization system [8], several algorithm visualization systems lack features that encourage students' engagement with the displayed visualization. *Computer Games* are systems that involve interaction with a user interface to generate visual feedback on a computer, or a video device and utilize fun, play, and competition [9]. In this paper, we address some required issues in creating tools

978-0-7695-3984-3/10 $26.00 © 2010 IEEE
DOI 10.1109/ITNG.2010.78

for teaching algorithms by:

- Comparing several algorithm visualization systems according to their level of engagement and presenting an alternative form of active engagement that produced by computer games (sec. 2).
- Reviewing the history of computer games usage in education (sec. 2).
- Introducing *Algorithm Game Visualization,* a novel method for visualizing algorithms using computer games to overcome the shortness in current algorithm visualization systems (sec. 3).
- Developing a game development tool to build algorithm games (sec. 4).
- Implementing several prototypes for algorithm games using the game development tool we have built (sec. 5).

Given that active engagement with algorithm visualization systems is far more important to learners than the graphics that they see, and computer games maximally engage their players, we conclude that algorithm games can help students learn algorithms better (sec. 6).

2. Related work

Alternative form of active engagement

Naps *et al.* define six different forms of learner engagement with a visualization technology [10]. 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* [3], *BALSA* [11], and *Algorithm Explorer* [12]. 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* [13]. Other systems support changing by allowing students to change the visualization data or some other features such as *GeoWin* [14]. The systems that support constructing encourage students to build and construct their own visualizations for the algorithms under study such as *ANIMAL* [15] and *ALVIS* [16]. Finally, systems that support presenting provide tools that help students to present visualizations to an audience for feedback and discussion such as *Algorithm Studio* [17]. Similarly, computer games support viewing, responding, changing, constructing, and presenting too. 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. While 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 [18]. Moreover, computer games fully interact with their players and encourage them to think, talk, and act. Therefore, we introduce "playing" as a new form of active engagement that maximally engages students through repetition, challenging, and enjoyment in addition to combining all five forms of active engagement presented by *Active Engagement Taxonomy.*

Computer games in education

In 1960, the University of Illinois built an educational computer system named *Plato* that uses computer games [19]. Afterward, many research centers started to develop educational computer games, but *Oregon Trail* (1971) was the first known educational game [20]. The success of research-based educational games, when presented commercially, led to the production of commercial educational games. Usually, educational games are known as *Edutainment* games for embedding education with entertainment, exemplars of such titles are *Snooper Troops* and *The Incredible Machine* [21]. For a while, edutainment titles dominated the educational computer games market; however, they started to lose their good reputation, because of using similar game play in many titles and having low-budgets [21]. Despite the frustration with edutainment games in the past, they reemerged recently as *Serious Games*, which have been developed for serious purposes other than entertainment such as training, advertising, simulation, and education. Many factors have contributed into the appearance of serious games such as the failure of edutainment games, the advancements in the computer games technology and the increased number of people playing games. In 2002, the "Serious Games Initiative" was launched to support the development of serious games that address policy and social issues. Moreover, many research projects focused on serious games such as the *Games to Teach Project [22]* and *Immune Attack* [23]. However, despite the use of educational computer games in teaching many subjects, still there is a need for more games

to teach several other subjects such as algorithms.

3. Algorithm game visualization

In general, computer games have many features that encouraged us to consider them to teach and visualize algorithms. Computer games have been broadly played all over the world by adolescents and young adults; in particular, the number of hours the standard college students spend on reading is half the time that they spend on playing computer games [24]. Therefore, the best method for teaching today students is using computer games. Moreover, the use of computer games converts an unpleasant and tedious operation of algorithm learning into an enjoyable and interesting experience. In addition, computer games fully interact with players and attract them to think and respond to different events. Since there is an increasing correlation between algorithm learning and the level of students' engagement [8], the use of computer games in algorithm learning will improve it and fulfill the lack of engagement in current algorithm visualization systems. Playing a computer game is an intrinsically motivating activity in which players engage for no reward other than the interest and enjoyment that accompanies it. The research shows that *Intrinsic Motivation* (A.1) improves learning [25], thus, the use of computer games will motivate students to learn and understand algorithms better. Furthermore, computer games players actively construct knowledge about the game topic and story through active interaction with the game. Also, the wining/losing of computer games eliminate the need for grades and tests, and assessment becomes part of the learning process so that students judging their own progress. Therefore, the use of computer games to teach algorithms is an application of the *Constructivism Learning Theory* (A.2), which provides better algorithm learning.

3.1. Algorithm game properties

We have called a computer game that teaches and visualizes an algorithm, an *Algorithm Game.* Each algorithm game has six common properties and features. First, the algorithm game can be created either by designing a totally new game or by modifying the gameplay of an existing game to simulate the algorithm steps. For example, we created the Binary Search Game by modifying the known (Pong) game to simulate the binary search algorithm as shown in section (5.1). Second, the algorithm game must be simple and not complicated, so students do not lose concentration and distracted, but it must be challenging or the student will be bored. Third, the algorithm game can be of any genre, but it must visualize the algorithm and its data structure, for example, we modify the Pong game to visualize the binary search algorithm by presenting the array data structure as a set of sequential boxes with values and the game play algorithm steps. Fourth, to support intrinsic motivation, the algorithm game should challenge the player by setting clear goals with appropriate difficulty levels and giving clear and encouraging feedback. Also, the information in the game should be complex and unknown to increase the player curiosity. Moreover, the game must give the most control to the player by providing many options to customize it and increase the player imagination. Lastly, the game should increase competition, collaboration and the recognition of peers. Fifth, the algorithm game graphics items must depict the features of the data structure that associated with the algorithm. For example, if the data structure is an array, we can use a group of boxes since the boxes has content we can use to store the array numbers and can be arranged sequentially as the array, for the tree data structure we can use an actual tree with leaves that represent the numbers. Sixth, the game-play of the algorithm game must simulate the behavior of the algorithm that game is visualizing.

3.2. Algorithm game architecture

The architecture (fig1) of each algorithm game consists of five basic modules; however, more modules can be added depending on the game and the algorithm it visualizes.

1. Game screens module: 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 Main Menu Screen (displays available options on game start up), Title Screen (what the player sees when she first starts the game), Game Play Screen (displays the game common properties such as scores and number of levels), and Game Level Screen (displays one level of the game). Each algorithm game must have at least the following basic game screens: Title, Main Menu, Play Game, Won, Lost, and Pause.

2. Game graphics items module: game graphics are everything that contributes to the visual appearance of the game such as fonts, (2D) Sprites, and (3D) Models. Game Graphics Items are the individual visual objects, which build the game world. They are either non-animated (background and foreground) or animated objects. Characters are the graphics items that animate on the screen and are either controllable

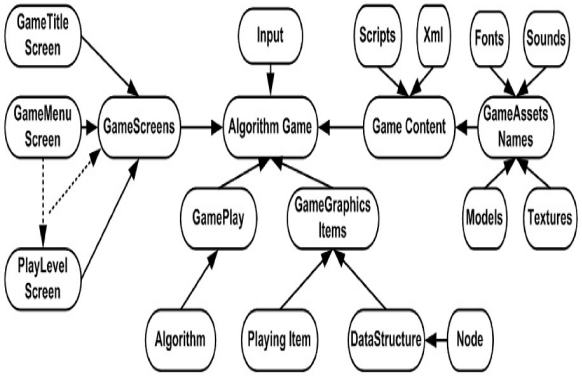

Figure 1. Algorithm Game Architecture

(Player Characters PC) or non-controllable (Non-Player Characters NPC) by the player. The algorithm game graphics items are either 2D Sprites or 3D Models that have attributes such as size, position, and name in addition to behaviors like render, move, and update. Exemplars of algorithm game graphic items are Nodes, which are used to visualize the nodes of the algorithm data structure; Playing Tools, which are used to play the game (ball, paddle, shooter, etc.) and Buttons, which are used in the screens general design.

3. Game content module: game assets are the element files that make the game world such as texture, font, model, and sound files. The game content module handles the loading, saving and building operations related to the game assets. In addition, it handles the game script files that contain the game custom script codes and the data storage files such as XML files.

4. Input module: the game input is the players' tactile contact with the game, which they used to respond to the game events and enter their choices. An excellent game offer the player a large number of meaningful options in addition to the choice of several input devices such as keyboard, joystick, game-pad or mouse. The input module handles the game input from the mouse, keyboard, and game-pad and the designer of the game must implement the required feedback for each player input event.

5. Game play module: 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, which involves drawing to the screen, playing appropriate audio, rumbling a controller, and providing any other form of output to the user. This module is responsible for implementing the game loop and playing rule according to the visualized algorithm.

4. Algorithm Game Implementation

The computer games development is a time-consuming and tedious process that requires experience in computer graphics and game design [10], which may discourage instructors from developing algorithm games to teach their students. However, after the release of Microsoft XNA Game Studio [26], which has been designed for simplifying game creation for students and hobbyists, the process has become relatively easier. Therefore, we have built a game development tool, namely *Algorithm Game Designer,* and a game engine, called *SAVGEngine (Serious Algorithm Visualization Game Engine)*, specifically, for implementing algorithm games as XNA games with as little code as possible.

Algorithm game designer. The *Algorithm Game Designer* has been implemented as a Visual Studio Isolated-Shell using Microsoft .Net Framework and C#. It has been built, specifically, to automate the development of an algorithm game by providing two types of editors: a *Code Editor* that supports the creation, debugging, compiling, and execution of a new algorithm game project and a *Script Editor* that provides an environment to write game-specific script codes. It also has a set of five graphics editors: *Game Assets, Properties, Screens, class, and Graphics Items Editors* that support the creation of several algorithm game components using a flexible, user-friendly graphical user interface. In addition, it provides an *Algorithm Game Template* to be used as a blueprint to create a new algorithm game and provides it with all needed basic classes and operations.

SAVGEngine. *SAVGEngine* is geared toward a two and three-dimensional environment. Specifically, *SAVGEngine* has no specific game logic coded directly in it; it only provides a common set of base functionality that can be reused for any algorithm game. Overall, the engine is responsible for various major modules that encapsulate all functionality for creating an algorithm game such as graphics, sound, input, game screens, physics, storage, and script managers. Also, it includes *BaseGame* and *PlayGame* classes that provide the new algorithm game with game timing and rendering loops in addition to the implementation of all basic game-play rules of the game. Moreover, *SAVGEngine* provides ready to use algorithm game components that can be altered and plugged-in into the new game such as data structures, algorithms, graphics items, game screens, and game assets.

Algorithm game creation. To create a new algorithm game using the *Algorithm Game Designer*, the devel-

oper needs to create a new algorithm game project using the included *Algorithm Game Template.* Then, the developer must set up the game properties, assets, graphics items, classes and screens using the *Properties, Assets, Graphics Items, classes,* and *Screens Game Editors* respectively. After that, the developer needs to implement the required methods in the *BaseGame* and *PlayGame* classes of the algorithm game, and adds any needed code using the *Script Editor*. Finally, the developer can use the *Code Editor* to compile, debug, and execute the created algorithm game.

5. Algorithm game prototypes

This section describes three algorithm game prototypes: binary search algorithm, singly-linked list and binary search tree. Each algorithm game prototype is described using ten design elements.

Algorithm game design elements:

1. Game idea: describes the game main goal and topic. Each algorithm game idea is visualizing an algorithm steps and its data structure.
2. Game start: describes the game start up screen components.
3. Game level: describes how the difficulty increases, how a level ends. Each completed level must achieve a learning sub-goal.
4. Game milestone events: points of the game at which the player rewarded or penalized.
5. Game end: explains what happens when the player loses, or wins or gets a high score.
6. Game input: the player's contact with the game. The default input devices are keyboard, mouse, and Xbox game-pad.
7. Game graphics: the algorithm game graphics must depict the characteristics of its data structure. For example, a block can be used to visualize one element of a data structure, while a set of blocks used to visualize an array.
8. Game sounds: musical sounds that play at game goal events or sound effects that play at other game events.
9. Game screens: a collection of visual and audio components that describe the state of the game at any one time during the game life cycle. The basic game screens of every algorithm game are Title, Main Menu, Play, Won, and Lost screens.

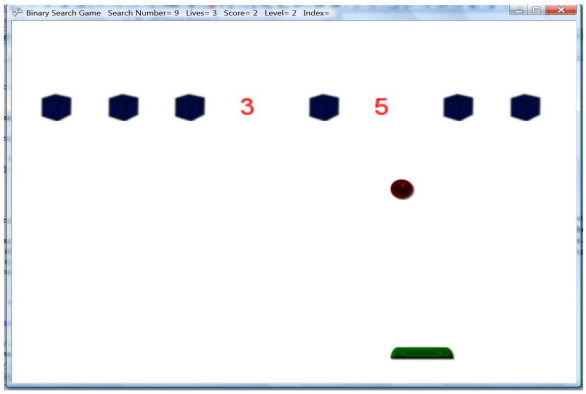

Figure 2. Binary Search- Play Screen

10. Game play: explains how the game is played and simulates the functionality of the algorithm.

5.1. Binary search game prototype

The binary search algorithm finds an index of a (target) value in a sequential list of sorted elements (array) by selecting the middle element (median) of (array) and compares it with the (target). Then, if (median) > (target), the index of (median)-1 becomes the new upper bound of (array); else if (median) < (target), the index of (median) + 1 becomes the new lower bound; else if (median) = (target), returns the index of (median). The algorithm pursues this strategy iteratively for the new list; 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.

Binary search game design elements:

1. Game idea: hitting an array of blocks with a ball using a paddle.
2. Game starts: displaying one game level that includes a random number of blocks with hidden values (array), a ball, a paddle, the Level Number, the Player Lives, the Search Number (target value) and the Player Score.
3. Game levels: several, at each new level the number of blocks is increased to make the game more challenging.
4. Game ends: when the player either loses all his lives or completes all game levels successfully.
5. Game milestone events: start of new level and a lost live.
6. Game input: default devices.

7. Game graphics items: a set of blocks where each block has a value that represents one element of the array, a ball, and a paddle.

8. Game sounds: HitBall, LostLive, Won and Lost.

9. Game screens: basic screens such as play (fig 2), menu, etc.

10. Game play: the player starts playing by hitting one of the displayed blocks (array) with the ball using a paddle. If the player hits a block in the middle (median), the player scores one point; then, if (Search Number > median), the player continues playing on the blocks on the right of (median); else If (Search Number < median), the player continues playing on the blocks on the left of (median); else if (Search Number = median), the level ends. Else if the block is not in the middle, the player loses one live; then, if (Player Lives=0), the player loses the game; else the player repeats the same level. When a level ends, if it is the last level and (Player Lives > 0), the player wins the game; else the (Level Number) is increased and the player starts new level.

5.2. Singly-linked list game prototype

A singly-linked list is a data structure, in which every element (node) contains some data and a (next) link to the next element. The list has two links (head), which point to the first node and (tail), which points to the last node. The singly-linked list has three basic operations: traversal, insert, and remove node.

1. Traversal: beginning from the head, 1. check, if the end of the list has not been reached yet; 2. visit current node; 3. current node becomes previous and next node becomes current. Go to step 1.

2. Insert: there are four cases for adding a node to a list. First, when the list is empty, set both head and tail links to point to the new node. Second, when adding before first node, update the new node link to point to first node; then, update the head link to point to the new node. Third, when adding after last node, update both the link of last node and the tail link to point to the new node. Fourth, when adding a node between two nodes, update the previous node link to point to the new node; then, update the new node link to point to the next node.

3. Remove: there are four cases for removing a node from a linked list. First, when the list has only one node, sets both head and tail links to NULL. Second, when removing first node, update the head link to point to the node next to the first. Third, when removing last node, update the tail link to point to the node before the last. Then, set the link of the new tail to NULL. Fourth, when removing a node between two nodes, update the previous node link, to point to the next node, relative to the removed node.

Singly-linked list game design elements:

1. Game idea: building a chain of connected nodes, according to the linked list insert and remove algorithms.

2. Game starts: displaying one game level that includes a node, the Level Number, the Player Lives and the Player Score.

3. Game levels: several, at each new level the number of the nodes in the required chain is increased to make the game more challenging and the new nodes will appear faster than before.

4. Game ends: when the player either loses all his lives or completes all game levels successfully.

5. Game milestone events: start of new level.

6. Game input: default devices.

7. Game graphics items: nodes with handler to be connected to each other.

8. Game sounds: FallingNode, ConnectingNode, Won and Lost.

9. Game screens: basic screens.

10. Game play: depending on their color, the player must build a chain of nodes as fast as she can, where nodes with the same color must be adjacent. During the game, a node with a different color is presented to the player continuously, so the player must add and delete nodes until reached to the required combination. All nodes are connected with each other using a chain, if a node added or deleted improperly, it will fall, and the Player Lives decreases by one, otherwise, when the node added correctly the Player Score increased by one.

5.3. Binary search tree game prototype

The Binary Search Tree (BST) is a data structure, in which each node has at most two children. Each node in the BST contains a value, which is lesser than the values of its right sub-tree and greater than the values of its left sub-tree. The binary search tree has three operations: search, add, and remove value.

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

2. Add: apply the search algorithm to find a place to put a new element; insert the new element to this place.

3. Remove: apply the search algorithm to find the parent of the node that has the value to be deleted. Then, if the node to be removed has no children, set corresponding link of the parent to NULL and disposes the node. Else if the node to be removed has one child, the node is cut from the tree and link single child (with its sub-tree) directly to the parent of the removed node. Else if the node to be removed has two children, find a minimum value in the right sub-tree, replace the value of the node to be removed with found minimum. Now, right sub-tree contains a duplicate! So apply remove to the right sub-tree to remove the duplicate.

Binary search tree game design elements:

1. Game idea: building the binary search tree, given values one after another, as fast as possible.
2. Game starts: displaying one game level that includes the root of the tree and a given value, the Level Number and the Player Score.
3. Game levels: several, at each new level the number tree nodes is increased to make the game more challenging.
4. Game ends: either when the Player Score becomes less than zero or when the player completes all game levels successfully.
5. Game milestone events: start of new level.
6. Game input: default devices.
7. Game graphics items: nodes of the tree, each node has a value.
8. Game sounds: AddNode, RemoveNode, Won and Lost.
9. Game screens: basic screens.
10. Game play: the player must build a binary search tree as fast as he can, by inserting new nodes into their correct places in the tree. If the player adds a node in its correct place, the Player Score increased by one. Otherwise, the Player Score decreased by one and he must remove the node, then reinsert it in its correct place.

6. Conclusion

Algorithm games benefit from the players' desire to win, love to compete and entertaining resulted from playing to motivate students learning algorithms. Compared to prior algorithm visualization systems, algorithm games provide richer visualizations that take better advantage of modern graphics and audio technology. These visualizations are designed to improve students' 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 students' assessment using the algorithm game winning-losing criteria without the need for external questions. Finally, we have designed the *Algorithm Game Designer* to easily create algorithm games with minimal training and effort. However, more improvements are needed. For example, *SAVGEngine* repository needs to be updated with more components such as implementations of known data structures, algorithms, graphics items, game screens, and game assets. In addition, *SAVGEngine* needs more modules to handle more game operations.

A. Learning theories

A.1 Motivation theory: is concerned with the factors that stimulate or inhibit the desire to engage in a behavior. Malone and Lepper distinguish between two types of motivation: *Extrinsic* that is supported by factors external to the activity and *Intrinsic* that arises directly from doing the activity. [25].

A.2 Constructivism theory: states that human beings actively construct knowledge for themselves through their active interaction with the environment, individual 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.

References

[1] 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.

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

[3] 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.

[4] M. D. Byrne, R. Catrambone, and J. T. Stasko, "Do algorithm animations aid learning?" Tech. Rep. GIT-GVU-96-18, 1996.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] M. Wolf, *The Medium of the Video Game*, "1st" ed. University of Texas Press, 2002.

[10] 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.

[11] 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.

[12] 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.

[13] T. L. Naps, "Jhave: Supporting algorithm visualization," *IEEE Computer Graphics and Applications*, vol. 25, no. 5, pp. 49–55, 2005.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] "World of warcraft," World of Warcraft Community Site– Website, accessed 10/15/2009. [Online]. Available: www.worldofwarcraft.com

[19] D. Bitzer, P. Braunfeld, and W. Lichtenberger, "Plato: An automatic teaching device," Master's thesis, University of Illinois, Coordinated Science Laboratory, 1961.

[20] 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

[21] S. Egenfeldt Nielsen, "Beyond edutainment exploring the educational potential of computer games," Ph.D. dissertation, IT-University of Copenhagen as, February 2005, accessed 11/12/2007. [Online]. Available: http://www.itu.dk/people/sen/egenfeldt.pdf

[22] 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

[23] H. Kelly, K. Howell, E. Glinert, L. Holding, C. Swain, A. Burrowbridge, and M. Roper, "How to build serious games," New York, NY, USA, pp. 44–49, 2007.

[24] 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.

[25] 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.

[26] "XNA Developer Center," MSDN, accessed 09/27/2009,. [Online]. Available: http://msdn.microsoft.com/en-us/xna/default.aspx