# AC 2007-495: PROGRAMMING GAMES TO LEARN ALGORITHMS

**Timothy Baibak, Kettering University**

Tim Baibak graduated Summa Cum Laude from Howell High School. He is a Computer Science Major at Kettering University and currently a Software Engineer Intern at Gabriel Roeder Smith and Company. Apart from programming, he enjoys tennis, fishing, and playing video games.

**Rajeev Agrawal, Kettering University**

Rajeev Agrawal received his BS and MS, both in Computer Science, from India and currently working on his PhD thesis at Wayne State University. Since 2001, he has been with Kettering University as a faculty member in Science and Math department. His research interests are Content Based Image Retrieval, Data Mining, eGovernment and Personal Data Privacy.

# Programming Games to Learn Algorithms

## Abstract

This paper discusses using the creation of computer games as a Computer Science course curriculum. It illustrates the benefits of such a curriculum, being that it would be a fun and engaging learning environment, it would attract new Computer Science students, and it would provide a solid foundation for the understanding of how to create algorithms. It goes on to suggest different kinds of games that could be assigned in a course, as well as the kinds of algorithms that can be learned in this process. Finally, this paper demonstrates different types of programming tools that can assist in the creation of the video games, eliminating some of the less important and more complicated algorithms necessary to make the games function, leaving the core ideas that are important for the students' growth.

## 1. Introduction

"Games are thus the most ancient and time-honored vehicle for education. They are the original educational technology, the natural one, having received the seal of approval of natural selection"[1]. Written by Chris Crawford, in his book *The Art of Computer Game Design*, this statement proves the importance of games in any aspect of education. Games have been used throughout time as an instrument of instruction for all different aspects of life. Puzzles to learn logic, mathematical games to enhance basic math skills, and even reading games to increase reading ability have all been used successfully to teach children the basic skills that they will need in life. It logically follows, then, that using computer games is an effective way to teach computing skills, and utilizing course curriculums that teach how to program computer games would invariably teach the basic skills required to program anything.

While learning basic programming skills, students who are assigned to program video games will learn the ability to formulate algorithms to solve particular problems, and will do so not only because are they having fun in creating these games, but they are also driven by a desire to solve the problems imposed in order to create something that they can be proud of creating, and show to their friends and relatives.

There are several benefits to teaching through the guided creation of video games. Not only does it provide a fun learning environment, but video games require several different kinds of algorithms to function, including database management algorithms, collision algorithms, input algorithms, path-finding algorithms, and even graphical algorithms. Even having students program simple arcade games, such as a Pong style game or a Pac-man style game, could cover pretty much all of the topics introduced in a traditional intro to programming course, while providing the educational drive for students to complete their work and discover more efficient algorithms to solve any problem they might encounter.

Programming games will encourage students to learn more, and to apply what they learn to create new things, reaching the ultimate goal of education. Through the establishment of programming games as a core curriculum of Computer Science classes, students will learn algorithms faster and with a deeper understanding, and will want to do this because of the fun and accomplishment associated with the creation of a computer game.

## 2. Benefits of Teaching by Games

The benefits of teaching algorithms by games are numerous, beginning with its potential to reverse a growing trend of lack of computer science popularity, and continuing to provide a fun learning environment in which students can approach abstract topics in a visual manner.

### 2.1 Attracting New CS students

A major problem facing Computer Science schools across the country is a lack of enrollment of new freshman. The Computing Research Organization reports that enrollment in computer science has dropped significantly in popularity from 2000 to 2005 [2]. This lack of interest in computer science could be attributed to a simple lack of interest from freshmen in programming, perhaps because they believe that they could never program anything useful. From personal experience, it seems that many students who take one of the current introductory Computer Science courses are merely introduced to the basic programming concepts and algorithms that allow them to create simple, mundane programs, with little or no graphical results for positive feedback.

Programming games as a part of the course curriculum would change this. Freshmen who take the initial computer science courses will have more fun, they will be learning to program things that they will be able to relate to. Students who enjoy programming the simple games taught in early classes would move on to take other more advanced programming classes, all the while improving their knowledge and understanding of algorithms and programming. Jessica Bayliss, a Computer Science instructor at the Rochester Institute of Technology, claims that using games to teach Computer Science engages students, and they are able to learn much better when engaged[3].

### 2.2 Making Learning Enjoyable

Learning should be fun. This saying has almost become the motto for western education, and it brings a deep level of truth with it. When a student is doing something that he deems as fun, he is able to grasp its concepts much more effectively, because he has a desire to learn more about it, and to work harder at it [4]. There isn't a student in the world that would object to homework if it were a fun experience; instead they would yearn for more.

From personal experience, programming a video game and watching it come to life is one the most enjoyable experiences available. An instantaneous gratification is obtained for a student who, after devoting time and effort into creating a video game, is able to play a game that he himself created. The student would benefit from this fun experience by learning or reinforcing their knowledge of simple programming concepts and algorithms, and afterward they would have a desire for more. They will also spread the word to fellow students, and encourage them to take the same classes, possibly motivating them to even consider Computer Science as a major.

## 2.3 Promoting a Synthesis Level of Learning

The creation of a video game involves the combination of several different algorithms and factors. A completed computer game reflects a significant understanding of all portions required, in other words the algorithms and concepts needed. So by teaching students to program a game, it would also teach students several of the core concepts behind devising and programming algorithms, with the motivation to use those concepts to create new games. Therefore the level of synthesis has been reached, the students will create new games, and within those new games, they will learn and devise new algorithms and programming techniques.

Teaching through programming games therefore becomes an advanced technique to achieve a level of application that many current Computer Science curriculums cannot truly reach. Normally, only core concepts are taught, given no real aspiration for creation of anything beyond homework, but when the capability to create games is added, it brings a new challenge to every student who takes the course, and that is to take the things they learned within the course and synthesize them into new games.

## 3. Examples of Programming Assignments

### 3.1 The Beginning: Pong

Widely accepted as one of the oldest and perhaps simplest games of all time, Pong is a logical choice for a programming assignment for a beginner's level computer science course. The game is basically an electronic version of the game table tennis, with two onscreen paddles, and a ball. The game can be played by either 2 players, or 1 player playing against a computer player. The goal is to move the paddle into the path of the ball, not allowing it to travel beyond your paddle and the other person to score [5]. Figure 1 shows a very simple Java version of Pong.

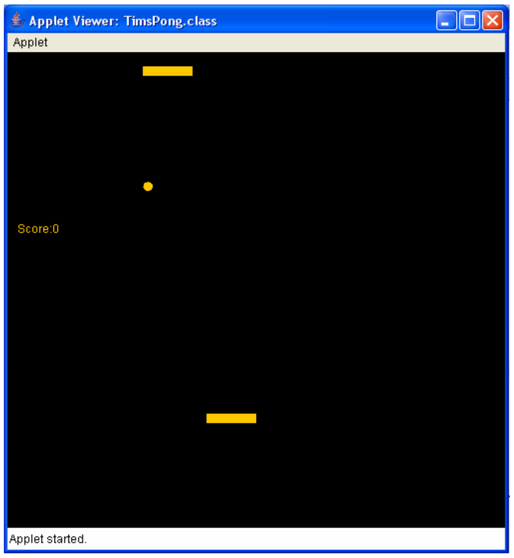

**Figure 1. My Very Simple Homemade Pong**

The programming of a version of pong is a fairly simple ordeal. It contains many of the elements that make up a beginner's computer science course, such as looping, control statements, as well as some basic input and output. Even basic object-oriented coding styles may be taught here; by making the ball and the paddles simple objects, students will be able to understand how objects work in a visual manner, which is extremely beneficial to their total understanding of the concept.

Students programming Pong will begin to understand the creation of algorithms to solve basic problems. Problems needing solving in Pong include how to handle collision between the ball and the paddle, how the artificial intelligence should behave, and how to determine if a player has scored. In order for a student to create a working version of pong, he or she will need to solve all of these problems. This is where learning to create an algorithm to solve a problem is introduced.

Students will have to come up with their own solutions for all of these problems, which may lead to variations between the games created, and a uniqueness that is important for the students' growth. For example, one student may choose to have the ball bounce in a ninety-degree angle no matter what the paddle is doing, while another may decide to create an algorithm to decide where on the paddle it hits and where the paddle is moving at the time of collision to determine which way the ball will travel. Each solution requires students to formulate a different algorithm, and this synthesis is vital to the learning process.

## 3.2 Pac-Man

Perhaps one of the most popular arcade games of all time, Pac-Man makes a great choice as a game of moderate difficulty for instructors to teach to their students. Pac-Man consists of an onscreen maze filled with pellets, and the goal of the game is to pick up all of the pellets. To make things interesting, there are ghosts controlled by artificial intelligence in the maze, who are trying to catch the player. The traditional game continues until all the lives of the player are spent, and records scores for a scoreboard [6].

A student implementation of Pac-Man would cover a large range of problems, and therefore several algorithms to implement. Starting with basic collision, an algorithm needs to be created to detect collisions between the player and the walls, as well as between the player and the ghosts, and the ghosts and the walls. Algorithms must also be created to determine the motions of the ghosts' artificial intelligence in order to follow the player while finding the shortest path through the walls.

A Pac-Man implementation would further improve students' understanding of object-oriented programming, as the player, each ghost, pellets, and even walls could be represented through objects.

A scoreboard added to Pac-Man would allow for the introduction of storing names and scores within a database. This database of high scores can then form the basis for sorting and searching algorithms, allowing instructors to teach some of the most important algorithms, while still maintaining the fun experience of creating a video game.

Variants of the standard looks and algorithms of this game would further establish the students' ability to formulate algorithms and their own ideas, allowing them to follow their natural desire to learn. For instance, one possible implementation of Pac-Man would be a grid system where each object occupies one square of the grid. While this may simplify the collision and path-finding algorithms, it may negatively impact the appearance of the game. These choices all need to be made by the student, so that they may learn to choose the best algorithm for a given problem.

### 3.3 Minesweeper

Another classic game that has been popular throughout the years is Minesweeper, a game where the player attempts to locate hidden "mines" by at first randomly clicking on the game board, and then devise their location using the numbered squares on the board. The objective is to mark or ignore all of the spaces on the board where a mine is located, without actually pressing on this square.

The implementation of this game can be very simple, as the game board may be set up with pre-rendered GUI components easily through Java or a visual programming language. This not only allows students to familiarize themselves with GUI programming, but it also allows them to focus on creating algorithms for the other components of the game.

This game was suggested by Jeff Lehman of Huntington College as an assignment for the ACM SIGCSE Nifty Assignments 2004, to "demonstrate two-dimensional array processing including passing arrays to functions/procedures/methods, object-oriented design and programming, and GUI design and programming, and recursion."[7] Apart from teaching basic programming concepts to beginners, more advanced students can be challenged to devise the fastest algorithms for both clearing the connecting tiles when a player uncovers an empty tile, building upon their understanding of time-complexity. These students may also analyze the time-complexity of their choice of algorithms for populating the grid, and even devise algorithms for the most efficient ways to solve the puzzle.

### 3.4 Guided Freeform Games

An alternative idea to selecting an individual game for students to create would be to provide a set of algorithms to be implemented by the student, instruct in their use, and have the students create their own design for a game using the algorithms given.

For example, if a instructor wants to teach the use of path-finding algorithms, sorting algorithms, and searching algorithms, the student would then be responsible for making a game that utilizes some kind of path-finding, most likely for an artificial intelligence, and some kind of database from which to sort and search.

This idea of freeform games leaves the stage wide open for almost any type of game, which could prove advantageous to any student wishing to apply their own skills to their own games. This idea of guided freeform games is more suited for an upper level course, after the students have already learned the basics for simple game development, and the core concepts of programming. It allows for these students who have already learned many of the basics to further learn new ideas while building upon their own.

Guided freeform games also allow students to delve into the world of analyzing algorithms. By initially establishing a game with the assigned algorithms, students can then test the efficiency of these algorithms. It also makes it possible to discover more efficient algorithms than others, where students can swap out the algorithms in an actual test situation of which they created.

## 4. Learning Different Types of Algorithms

There are countless numbers of algorithms to be implemented within video games, if only for the simple fact that there are a countless number of problems that could be solved. Students can learn a lot about algorithms in general, simply by creating their own to solve a given problem. At the same time, students can use pre-existing algorithms in their games to gain a better understanding of these established solutions to common problems.

### 4.1 Game-related Algorithms

The algorithms specific to game development are important learning tools for students. By learning how to implement collision or physics algorithms, it reinforces the student's basic understanding of creating algorithms. Every time a student implements or devises an algorithm, his or her knowledge of algorithms and programming in general increases.

#### *4.1.1 Collision Detecting*

How to determine when one object collides with another is a common problem when programming almost any type of video game. There are several kinds of possible solutions to implement, from simple boundary detection, to a two-dimensional game operating through a grid setup. In such a grid system, the only checks that would need to be made would be if the grid space into which an object is attempting to move were occupied.

Focusing on boundary detection, an analysis on the time complexity of algorithms can be performed. Consider an example with one object moving through the screen. Apart from this object, there are N other objects on screen that this object could collide with. When making sure that the moving object does not collide, should its boundaries be tested against all other objects on the screen? This solution would result in N comparisons. Or perhaps the screen should be cordoned off into four separate zones, where each object belongs to a particular zone, and collision detections are only done on the objects sharing the same zone. This would reduce the overall number of comparisons done on the system. What if every object on the screen was moving, such as in a game that might include an asteroid field? To check each object with each other every time, it would result in an overall time complexity of $O(N^2)$, while a zone system would reduce the best and average case scenarios.

These algorithmic solutions to what would appear to be a fairly simple problem excellently demonstrate the elements of time complexity and analysis of algorithms to students. It begins the process of learning to choose the best algorithm that will operate in the least amount of time, all while allowing students to create their own games, and formulate their own ideas.

### *4.1.2 Physics Algorithms*

Several video games also require a solution for the problems of determining how game objects will move, or if there is a "gravity" force pulling an object down, as well as determining momentum and the paths of objects after collisions. All of these problems require algorithmic solutions, which not only teach important aspects of Computer Science, but Physics too.

Having students implement these algorithms may be as simple as determining the trajectory of a bounce in Pong, or as complicated as handling gravity, acceleration, velocity, and mass. Were these things included as properties of the game objects, students could then convert physics formulas and concepts of momentum and trajectory into algorithms. To use the concepts of acceleration of gravity, or velocity, the formulas and constants that typically are used to calculate in SI units, such as meters, into either pixels, or some other length which has significance on a computer monitor. Time must also play a factor here, and the programmer will need to consider algorithms in order to handle the "per second" portion of any velocity or acceleration.

Having students solve these physics and conversion problems through algorithms will further increase their understanding of how to solve any general problem. The more that they learn through different applications that relate closely to the real world, the more they will be able to get out of a course.

## 4.2 Teaching Object-Oriented Solutions

Almost every possible kind of video game can be implemented through object-oriented programming [8], a concept important for beginner students to learn, as object-oriented programming is becoming an integral part of many modern languages. Utilizing objects can also play a vital role in implementing algorithms, especially through concepts of inheritance, and data storage.

By providing students with a visual representation of the objects they learn to create, students can learn abstract concepts much simpler [9]. It is important for students who will be programming algorithms in object-oriented languages to be familiar with objects, as they can play a crucial role. For example, when one is presented with the problem of needing to store a database of a type of record, (it could be an employee, or a CD, or any number of things) and this must be sorted, then using objects to store the data would simplify the sorting algorithms greatly. A possible solution to this problem without objects would be an implementation using arrays, but this solution may be significantly harder to implement and understand, as well as have a larger time-complexity.

In short, having introductory students program simple object-oriented games would provide an understanding of the subject quickly. Instructors Jimenez-Peris, Khuri, and Patino-Martinez have observed that Computer Science students learn much better through visual methods as opposed to abstract methods [9], and programming a video game provides an excellent visual approach to programming problems.

## 4.3 Sorting and Searching Algorithms

The problem of organizing data yields perhaps the most important kinds of algorithms for students to learn and analyze. There are numerous standard algorithms as to how to sort and search for data. Any video game that displays more than one game object on the screen needs to implement some kind of decision on what to be rendered when.

The set of game objects within any video game can be placed within a database, in order to be able to process and render them in the order desired. Through this, students can also learn the implementation of data structures, such as arrays, linked lists, and binary search trees, as well as the time complexities of the algorithms used to sort and search through them.

### *4.3.1 Sorting Algorithms*

With a database of game objects established within a program, students can then learn the fundamentals behind established sorting algorithms, as well as the time complexity of these algorithms. These game objects can be sorted in any number of ways, from priority of objects to be rendered, to represent their locations on the screen in relation to a reference point, or even in a game implementation which needs to know which object gets drawn on top of which.

If no other part of a game requires its objects to be sorted, any game, which gives a score based on a player's performance, could be made to maintain a high score database, and sort this database based on the scores of the players. Further, this could even introduce concepts of file storage, and algorithms to save and load data from the hard disk. By implementing established sorting algorithms to sort in-game data, students can further analyze the time complexities of algorithms, building upon their previous knowledge.

### *4.3.2 Searching Algorithms*

Just as the problem of sorting the objects within a video game helps to teach basic algorithms, the problem of picking out a particular object also provides an important demonstration of established algorithms. The two most important searching algorithms to choose from include the sequential search, which does not require the target list to be sorted, and the binary search, which while requiring a sorted list, dramatically reduces search times. The student must make this choice, as he or she is required to compare the time complexity of sorting and then searching, or just searching.

Analyzing the time complexity of each is an important step in students deciding which algorithm implementation is better. In other words, students may have to decide between the time complexity of simply searching for game objects sequentially in an unsorted list, and the time complexity it would take to first sort that list and run a binary search on it. These kinds of decisions will prove crucial to the development of the student within Computer Science, as choosing the best algorithm for a particular situation is something that programmers do every day.

## 4.4 Path-finding Algorithms

Robert Snapp, from the University of Vermont, uses games successfully to teach algorithms to his students. One such algorithm that he teaches to his students through a game in a corn maze is Tremaux's algorithm, an algorithm which assists in finding a path through a maze [10]. This same concept can be applied by programming artificial intelligence within a video game to traverse a maze.

Critical to the creation of several forms of artificial intelligence, path finding algorithms and shortest path algorithms can be learned well through programming video games. Especially in games such as Pac-Man, the problem of how the computer players should move is important. Some kind of path-finding algorithm is required. It could be as simple as a basic coordinate comparison, with collision detection for walls along the way, or the game could make use of graph techniques to calculate a shortest path.

One example of a very basic path finding algorithm was implemented in a homemade Pac-Man, written in Basic. For each of the ghost characters, it checked its coordinates' relationship to the Pac-Man character. If it could move in the direction of the Pac-Man without colliding into a wall, it did so, and if not, it would move in another direction, in the hope that it would eventually find a way around the wall.

While this algorithm is incredibly simplistic, it runs very quickly, but does not always decide the best possible solution. This algorithm can provide a fine lesson to students about choosing the right solution for the right problem. This algorithm does not require a lot of thought, coding, or time to execute, and it works well some of the time, but it may not be the best choice. It is up to the student then to learn to decide what algorithms work the best.

Another possible implementation for the same Pac-Man path-finding problem might be a graph solution, and a shortest-path algorithm. This could be implemented by creating a graph where the nodes are intersections of the paths in the maze, and the edges between the nodes are the actual paths of the maze, whose weights are the distance between them. Each object will keep track of which edge they are on, and the distance between itself and the closest node. Then by implementing a shortest-path algorithm whenever the ghost decides to move, the ghost can choose the quickest route to reach the Pac-Man. While this solution is more complicated than the previous, and has a larger time-complexity, it has the potential to be much better at guiding the ghosts toward their pellet-munching adversary, as well as teaching students about graphs, and shortest-path algorithms.

As part of a course curriculum, students could be assigned both algorithms to implement and test, in order that they may see the advantages and disadvantages of both. The comparison between algorithms is one that is stressed in Computer Science curriculums, and through the programming of video games, students can learn to make these comparisons on their own. Furthermore, they will want to find the best possible solution so that their game will function as well as possible, and this will advance their education considerably, because they will learn how to trade off between speed and accuracy.

## 5. Tools to Create Games

While it is argued above that through programming video games, students will learn the basic concepts and programming skills behind algorithms, it still may be seen as a daunting task to have students program a fully functional video game, no matter how simple the game itself may seem [11]. Fortunately, there are several different programming tools that can be employed to create video games, some of which that take away some of the more difficult and less educational parts.

### 5.1 Java Applets

One possible form into which a simple game can be created is a Java Applet. As a fairly simple and powerful object oriented language for which many open-source compilers exist, Java is an ideal language for learning programming skills. Through Java Applets, simple graphical applications can be created easily, which do not require a deep understanding of Java itself.

Instructing a student on how to program using Java Applets will therefore not be difficult, as there is little else to learn but the algorithms themselves. Some things that should be provided to the students to save them some time include a double buffering algorithm to eliminate flickering in the game, some basic event listener methods, and the shell of a typical Java Applet. From here, the student will be able to add his or her own code and see some immediate results.

### 5.2 Microsoft XNA Game Studio Express

A rather new and powerful technology for beginner game programmers to use is Microsoft's XNA Game Studio Express [12]. While this technology is currently in its beta phase, after just a few hours of experimentation and tutorials were enough for me to create a functional and rather cool Microsoft Windows Application. Using the C# programming language with this new set of libraries, the syntax is almost identical to Java, with some features that make it much easier.

The XNA libraries are also capable of handling a lot more difficult concepts, such as 3D rendering through built-in Direct X functions that are easy to use. This adds to the feeling that a student using this tool is really creating something close to the video games made today.

Figure 2 is a screenshot of a personally homemade program created after some basic tutorials on 3D rendering provided by the XNA homepage [12]. Each of the four cubes shown below is actually the same size, with varying z components, giving them the appearance that some are closer than others. The XNA library itself, along with a simple Direct X effect provided by tutorials, renders this appearance, removing the need for the programmer to spend a significant amount of time creating their own 3D renderer. The user controls one of the cubes, while the other three cubes merely wait spinning.

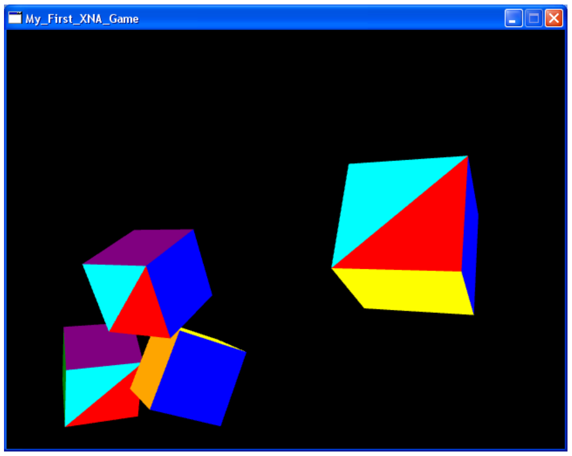

**Figure 2. My Bouncing Cube Program Made With XNA**

The user input is handled through a few simple method calls which read the condition of a plugged in Xbox 360 controller. Each object has its own physics elements that are an x position, a y position, and a z position, along with vectors representing current velocities and accelerations. The position of the left analog stick on the controller dictates the acceleration of the user-controlled cube. A simple collision detection algorithm and conservation of momentum were added to allow the cubes to bounce off of one another. When the cubes are set in motion, they will not stop unless a friction force is placed upon the objects, which will slow and eventually stop them.

This simple exercise alone demonstrates how students could learn different algorithms using this particular tool to create something. The collision and input algorithms are an example of this. The physics algorithms are another, and even more basic algorithms could be applied to this example. The non-user controlled cubes could be programmed with a simple path-finding artificial intelligence algorithm, where they would chase the player cube around. There are so many possibilities to learn new algorithms with programming games that they're almost endless.

## 5.3 Game Development Engines for Mac

There are several game development engines created by third party vendors for Mac users such as Torque Game Engine (TGE) and Torque Game Builder (TGB) from GarageGames, Unity from Over the Edge, and PTK from Phelios [13]. Except Unity, all these tools are also available under Microsoft Windows. TGE supports both development and deployment on Mac OS X, Windows, and Linux with XBox 360. TGE comes with complete C++ source code to help the developers to change the inner workings of the underlying engine itself. Torque Game Builder (TGB), needs only the 2D description of the game from the developer. It allows massive numbers of sprites, parallax scrolling backgrounds, intermixed 3D objects, TorqueNET Lite networking, swept polygon rigid body physics, and ultra-fast collision detection. Unity has been used to produce the critically acclaimed game GooBall. Unity offers extensive scripting support via JavaScript, C#, and Boo (like Python) and the scripts compile down to fast native code. PTK simply provides the source code to a game engine framework and handles the complexities of multi-platform development. Recent commercial games developed using PTK include Atlantis, Universal Boxing Manager, and DoulberGold. PTK engine can be downloaded for classroom usage. These tools are designed for serious game developers, but can also be used in classroom setting to help the students to understand important computer science concepts. In Figure 3, one example belonging to each game engine is shown.

**Figure 3. Examples of games developed using Mac OS X tools**

## 5.4 Open Source Game Engines

In addition to above, there is a long list of open source game development engines like Kochol Game Engine (KGE), Maelstrom Game Engine (MGE), Kool Object-oriented Game Engine (KoGe), GNE (Game Networking Engine) etc. The complete listing and downloads are available on *http://sourceforge.net/*.

## 6. Conclusion

Understanding how to solve problems through algorithms is an integral part of Computer Science. Without this part, there would be no Computer Science, as without algorithms, there is no code. Allowing students to program video games as part of a course curriculum would let students understand the core concepts behind programming algorithms, as well as teach some of the basic kinds of algorithms, and it would do this in a fun and engaging learning environment where the student is eager to learn, as he or she is eager to see the results of his or her labor.

A visual approach to programming helps students to visualize abstract ideas that otherwise would be difficult to grasp. Not only can programming games help students to learn the concepts behind algorithms, but it can also be used to teach basic programming concepts like object-oriented programming.

The several tools available to program games will help instructors in providing a learning environment that is not too difficult for introductory students to understand. They can also allow for much more complex results with less knowledge required about the inner workings of the hardware being programmed upon. A course curriculum focused on programming video games is an excellent solution to attract new students and better teach the concepts of algorithms.

## Bibliography

1. Crawford, C. *The Art of Computer Game Design.* Washington State University, 1997, URL: http://www.vancouver.wsu.edu/fac/peabody/game-book/Coverpage.html
2. Vegso, J. *Interest in CS and CE as Majors Drops in 2005.* CRA Bulletin, 2006, URL: http://www.cra.org/wp/index.php?p=75
3. Hoffmann, L. *The Nerd Boat.* Forbes.com, 2006, URL: http://www.forbes.com/digitalentertainment/2006/01/30/videogames-development-microsoft_cx_lh_0131nerdboat.html
4. Grigoriadou, M. and Maragos, K. *Towards the design of Intelligent Educational Gaming Systems.* The University of British Columbia.
5. *Pong.* URL: http://www.atari.com/
6. *Pac-Man.* URL: http://www.namcobandaigames.com/
7. Lehman, J. *Minesweeper.* ACM SIGCSE Nifty Assignments, 2004, URL: http://nifty.stanford.edu/2004/LehmanMinesweeper/
8. Becker, K., and Parker, J. R. *All I Ever Needed to Know About Programming, I Learned From Re-writing Classic Arcade Games*. Future Play, 2005, URL: http://www.futureplay.org/papers/paper-184_becker.pdf
9. Jimenez-Peris, R., Khuri, S., and Patino-Martinez, M. *Adding Breadth to CS1 and CS2 Courses Through Visual and Interactive Programming Projects.* The proceedings of the thirtieth SIGCSE technical symposium on Computer science education, 1999, pp. 252-256.
10. Reidel, J. *The Learning Game.* The View, 2003, URL: http://www.uvm.edu/theview/article.php?id=960
11. Lewis, C. and Repenning, A. *Playing a Game: The Ecology of Designing, Building, and Testing Games as Educational Activities.* Trails, URL: http://www.trails-project.org/resources/papers/Colorado_EdMedia_paper.pdf
12. XNA: http://msdn.microsoft.com/directx/XNA/default.aspx
13. Developing Games on Mac Os X Using Third-Party Game Engines *http://developer.apple.com/games/gameenginesonmac.html*