"3D Game Development"

Project Report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology.

In

Computer Science & Engineering

under the Supervision of

Ms. Ramanpreet Kaur

By

Ushpinder Singh

Enrollment No.:111337

To

Jaypee University of Information and Technology

Waknaghat, Solan – 173234, Himachal Pradesh

CERTIFICATE

This is to certify that the work titled "**3D Game Development"** submitted by **Ushpinder Singh** in the partial fulfillment for the award of degree of Bachelor of Technology in Computer Science & Engineering from Jaypee University of Information Technology; Waknaghat has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Signature of Supervisor:

Name of Supervisor : Ms. Ramanpreet Kaur

Designation : Assistant Professor

Date : 15-05-2015

ACKNOWLEDGEMENT

I take this opportunity to express my profound gratitude and deep regards to **Prof. Dr.RMK Sinha Dean of CSE, JUIT** for providing me the opportunity to do this project as a part of my final year project.

I also take this opportunity to express my profound gratitude and deep regards to **Prof. Dr. Satya Prakash Ghrera, Brig. (Rtd.) and HOD of CSE, JUIT** for providing me the opportunity to do this project as a part of my final year project.

I also take this opportunity to express a deep sense of gratitude **to Ms.Ramanpreet Kaur, Associate Professor, Computer Science & Engineering Department, JUIT**, for his cordial support, valuable information and guidance, which helped me in completing this task through various stages.

Date: 15-05-2015 **Ushpinder Singh**

TABLE OF CONTENT

LIST OF FIGURES

ABSTRACT

With the wide application of 3D games and the improvement of Computer Graphics and Hardware performance, the competition of 3D game products is becoming increasingly furious. So, how to develop a 3D game that can match the Performance of PC platform 3D game with other games has become a hot-topic in the field of 3D game development application.

My project for 3D game development is to develop a 3D game using java programming which consists of multiple levels. The Java 3D API is an application programming interface used for writing three-dimensional graphics applications and applets. It gives developers high-level constructs for creating and manipulating 3D geometry and for constructing the structures used in rendering that geometry. Application developers can describe very large virtual worlds using these constructs, which provide Java with enough information to render these worlds efficiently. This report discusses the design and plan affection of the implementation of the 3D game for a joint Interactive Media and Game Development and Professional Writing Major Qualifying Project. This report will detail the game‘s vision, the artistic and technical designs, the game play features of the project, and research on effective strategies for providing in-game help. Theme of the game is to develop a 3d clue based game that consists of different clues for both online as well as offline using java.

First Level: First Level takes place in the 3d game map which consists of different entities(clues) that will be rendered on the floor and also there will be main character who will have to pick the clues with the help of instructions provided by computer i.e. AI part for up gradation to the next level within a given time. Once the all clues will be picked then player will be upgraded to next level and that level will consist of more clues. At the end the scores will be displayed on the screen

CHAPTER 1: INTRODUCTION TO 3D GAME DEVELOPMENT

1. Introduction

Recently, games have become one of the most profitable factors in the software industry. More specifically, during the last few years the game industry has been considered to produce revenue greater than the movie industry and its development rate has been one of the fastest growing

in the United States economy .Furthermore, game design and the methods used for easier and more efficient development constitute a very interesting open research field . It goes without saying that computer games play a very important role in modern lifestyle. Therefore, it is no longer necessary to explain what a computer game is .On the other hand; it is not so obvious Why game research is an extremely interesting field and simultaneously why game development is a very complicated task to accomplish.

A game could be described as a closed interactive system where the user tries to execute specific combinations of instructions in order to achieve completion of defined tasks to progress to an end goal, there are several more descriptions (Salen & Zimmerman 2004) and attempts to give a definitive description but this one is adequate for this paper. This concept has not changed since the beginning of games.

Creating games requires some sort of method/tool where every entity in the system could be given a set of attributes and how they interact with each other, entity to entity but even attribute to attribute. Some of the attributes even have to be able to change depending on specific circumstances, keeping track of all the variables and producing a consistent and rigid model of game mechanics is very complex.

Today this is done with tools and methods developed for games that was fairly simple, the technology to do so back then was, in comparison with today‘s technologies, simple but adequate. The games have evolved; the methods and tools have not seen a parallel evolution in capabilities.

The answer to the first question has many levels. As mentioned above, even though game development is a very strong industry, the research on this field is in its infancy. This fact leads game programming professionals to demand better developing methodologies and software engineering techniques .Furthermore, games are the first and sometimes the only market for advanced graphics techniques to demonstrate the quality of graphics they produce It has been acknowledged that game industry draws on research from academia, corporate R&D labs and in-house work by game developers. The distinction between games and other forms of software is that, in games, the development groups consist of people with Different fields of expertise. First of all, a script writer is required; this person will write the game script and fill in a document usually called __concept paper.

The lead Game designer will convert information from the concept paper into another called __design document 'which will be the guide throughout the development process. Apart From that, the company employs a group of programmers with several skills and expertise, such as engine and graphics programmers, artificial intelligence programmers, sound programmers and tool programmers.

2. Game Development Platforms

Take brief look at the most popular technologies used in game development. These vary from low-level graphics and input/output routines to more simplified libraries which handle most of the complicated work for the programmer.

2.1 OpenGL

One of the most popular graphics libraries, OpenGL was developed as an open standard, meaning that any external contributor could take part in the development process. OpenGL provides the user with a large set of functions that allow the programmer to render triangles and more complex polygon structures in a fairly straightforward way. It also supplies the user with some basic mathematical transformation functions for vectors and matrices which is essential when performing movement or rotation on the screen. OpenGL is known to be a highly portable graphics rendering platform and is available for nearly all system architectures, varying from Windows and Linux to MacOS. This graphics library is said to be one of the best documented, mainly due to abundance of contributors throughout the

world. On the other hand it also makes some parts of documentation inconsistent for the same reason. Nevertheless a computer graphics adept will feel quite comfortable while learning and applying OpenGL in his/her software.

2.2 DirectX

With the growing popularity of OpenGL, Microsoft decided to make a move of their own. In 1995 the company presented their break-through operating system called Windows 95 and along with it the first edition of their own graphics/audio/input/output programming library called DirectX. The audience was very excited, not only because of the fact that Microsoft is releasing an operating system which would support game development. The key reason behind it was the first game ported to DirectX: Doom by Id Software - the same game that in 1993 caused mass hysteria among the computer gamers all over the world. Even though the first releases of DirectX were far from successful (due to high amount of programming bugs and arguably low functionality in comparison to OpenGL), the library itself evolved through years, becoming a very tough opponent for OpenGL. DirectX not only was written natively for the Windows platform, it also had its own extensions for handling mouse and keyboard events (DirectInput) and sound (Direct Audio). Even though DirectX is designed only for Windows (there are no known ports of DirectX for other systems due to legal issues) it is widely used today both for PC development and console games (for example Xbox 360).

2.3 Selection of the programming language

Let‘s now take a deeper look at the technology we will use to develop the game engine: the programming language. This is a very important step in the design process which determines the future code structure as well as the flexibility of the framework. Another key issue is the desired performance of software using the engine which in this particular situation is crucial, such as handling of the drawing routines, most data processing and complex calculations. In order for our application to be efficient, we have to ensure that all tasks will be performed in the fastest way possible which (apart from code optimizations) is highly dependent on the structure of generated binary code. So how do we choose the best programming language for the job?

Programming languages in this area are: Java, C, C++, Python and Flash/Action Script along wit Macromedia Director. We will focus on development for the PC platform only, disregarding the console market. High popularity of these languages is a result of on different historical and marketing events: C++ has been a language used for most top-performance application development since 1980s. Java is known for its simple syntax and extensive amount of additional libraries, making it a fairly powerful language. Python gained popularity through almost the same reasons, as well as the built in force-mechanism to write clear code using indentations. Finally, Flash and Director have been long known in development of rich online content. Let‘s now learn something more about them and try to decide which one will best suit our needs.

2.4 Java

Java was a secret project at first, the goal of which was to develop a high-performance language for internal use at Sun Microsystems. After a course of time the project evolved and turned into Java, which became a general-purpose programming language available for everyone to be used for free. Java very soon found recognition among developers, especially because of its syntax being close to the one used in C++. This made the learning curve somewhat gentler for programmers. Java featured automated memory management and a built in garbage collection (automatically taking care of unused data in memory), something that made the language popular among beginner coders or those who found the concept of memory management too difficult in C++. The major plus side of Java is the amount of additional packages and libraries that come with the standard edition of the language. This means that Java developers can start taking advantage of advanced language features right away, unlike the C++ programmers which have to get hold of any additional libraries themselves. Java also uses a concept of virtual machine, which makes it possible to run the exact same code on any operating system that has a Java VM installed on it This means no need to make code modifications in order to make it portable: something that was not always possible in C++. On the other hand, Java is considered a slow language: mainly because the programmer lacks the possibility to arrange data in memory by hand and slight delays introduced by using the virtual machine instead of direct code execution. Nevertheless, there are advanced graphical libraries available for Java, making it a potent platform for game developers.

3 Choosing A Game Development Platform

Java 3D is an application programming interface (API) developed at Sun Microsystems for rendering interactive 3D graphics using the Java programming language. Java 3D is a *client−side*

Java API. Other examples of Sun client−side APIs include the Abstract Windows Toolkit (AWT) and Java Foundation Classes (JFC/Swing), which are both Java class libraries for building applications with a Graphical User Interface (GUI). The client−side Java APIs are in contrast to Sun‘s server−side APIs such as Enterprise Java−Beans (EJB) and the other components of Java 2 Enterprise Edition (J2EE). Making 3D graphics interactive is a long−standing problem, as evidenced by its long history of algorithms, APIs, and vendors. Sun is not a major player in the 3D graphics domain, although its hardware has long supported interactive 3D rendering. The dominant industry standard for interactive 3D graphics is OpenGL, created by Silicon

Graphics (SGI). OpenGL was designed as a cross−platform rendering architecture and is supported by a variety of operating systems, graphics card vendors, and applications. TheOpenGL API is written in the C programming language, and hence not directly callable from Java. A number of open source and independent programming efforts have provided simple Java wrappers over the OpenGL API that allow Java programmers to call OpenGL functions, which are then executed in native code that interacts with the rendering hardware. One of the most popular is GL4Java, which you can find at However; there are few advantages to using a Java wrapper over OpenGL, as opposed to coding in C and calling OpenGL directly. Although programmers can use the more friendly Java APIs, they must incur the Overhead of repeated calls through the Java Native Interface (JNI) to call the native OpenGL libraries. Java 3D relies on OpenGL or DirectX to perform native rendering, while the 3D scene description, application logic, and scene interactions reside in Java code. When Sun set out to design Java 3D, although they did not have the resources or industry backing to replace OpenGL, they wanted to leverage more of Java‘s strengths as an object−oriented programming (OOP) language instead of merely delegating to a procedural language such as C. Whereas OpenGL‘s level of description for a 3D scene consists of lists of points, lines, and triangles, Java 3D

can describe a scene as collections of objects. By raising the level of description and abstraction, Sun not only applied OOP principles to the graphics domain, but also introduced scene optimizations that can compensate for the overhead of calling through JNI.

The foremost strength of Java 3D for Java developers is that it allows them to program in 100 percent Java. In any sizeable 3D application, the rendering code will compose only a fraction of the total application. It is therefore very attractive to have all the application code, persistence, and user interface (UI) code in an easily portable language, such as Java. Although Sun‘s promise of Write−Once−Run−Anywhere is arguably more of a marketing dream than a reality, especially for client−side programming, Java has made important inroads toward enabling application developers to write applications that can be easily moved between platforms. The platforms of most interest today are Microsoft Windows 98/NT/2000, Sun Solaris, LINUX, and Macintosh OS X. Java has arguably become *the* language of networked computing and the

Internet. High−level support for remote method invocation (RMI), object serialization, platform independent data types, UNICODE string encoding, and the security model all provide persuasive arguments for adopting the Java language for applications that are increasingly gravitating away from a desktop−centric worldview. Many of the state−of−the−art 3D graphics applications being built with Java 3D today are leveraging the strengths of Java

as a language for the Internet. The Java 3D API itself has much to offer the application developer. By allowing the programmer to describe the 3D scene using coarser−grained graphical objects, as well as by defining objects for elements such as appearances, transforms, materials, lights, and so forth, code is more readable, maintainable, reusable, and easier to write. Java 3D uses a higher level scene description model, the *scene graph*, which allows scenes to be easily described, transformed, and reused.

Java 3D includes a view model designed for use with head−mounted displays (HMDs) and screen projectors. By insulating the programmer from much of the complex trigonometry required for such devices, Java 3D eases the transition from a screen−centric rendering model to a projected model, where rendering in stereo allows for greater realism.

Java 3D also includes built−in support for sampling 3D input devices and rendering 3D spatial sound. By combining all of the above elements into a unified API, Java 3D benefits from a uniformity of design that few other APIs can match. Java 3D‘s higher level of abstraction from the mechanics of rendering the scene have also opened the field of interactive 3D graphics to a new class of audience, people who would typically have been considered 3D content creators.

3.1 Benefits of Choosing Java

Java was designed with several goals in mind. Chief among them is high performance. Several design decisions were made so that Java 3D implementations can deliver the highest level of performance to application users.

- Other important Java goals are to:
- Provide a rich set of features for creating interesting 3D worlds, tempered by the need to avoid nonessential or obscure features(Rendering).
- Provide a high-level object-oriented programming paradigm that enables developers to deploy sophisticated applications and applets rapidly.
- Provide support for runtime loaders. This allows Java to accommodate a wide variety of file formats and allow to run on multiple platforms.

CHAPTER 2: LITERATURE SURVEY

1. Evaluation of Object Oriented Design Patterns in game

1.1 Summary

This paper aimed at evaluating the use of object-oriented design patterns in game development. In order to achieve this goal we examined two open-source games. The results extracted by the two games were almost identical and indicate that patterns can be beneficial with respect to maintainability. The game version that includes the patter under study has reduced complexity and coupling compared to a prior version without the pattern. Additionally, the application of patterns tends to increase the cohesion of the software. In contrast to that, the size of the projects has increased in the pattern version. Consequently, due to the evolving nature of games we believe that the appropriate employment of design patterns should be encouraged in game programming.

1.2 Game Architecture

One of the most interesting aspects of game research is the architecture that the developer will use. In recent papers, there are a few references to the modules that the programs are being decomposed to, however, without extensive discussion of maintainability and code reusability issues. Such issues have been examined in detail in classical object-oriented programming, but those ideas are extremely immature in game programming. Designing and programming large-scale software is a very complicated job that requires many human work hours. Consequently, software is usually divided, logically, into subprograms that are autonomously designed, programmed and tested by separate programmers groups.

These subprograms are called modules. Decomposing software into modules is an important decision that plays a main role in the architecture and further design of the program.

In this section, the modules proposed for games are examined and briefly discussed. In, Bishop et al. described a general game‘s architecture as shown in Fig. 1. This schema presents an interactive game‘s vital modules.

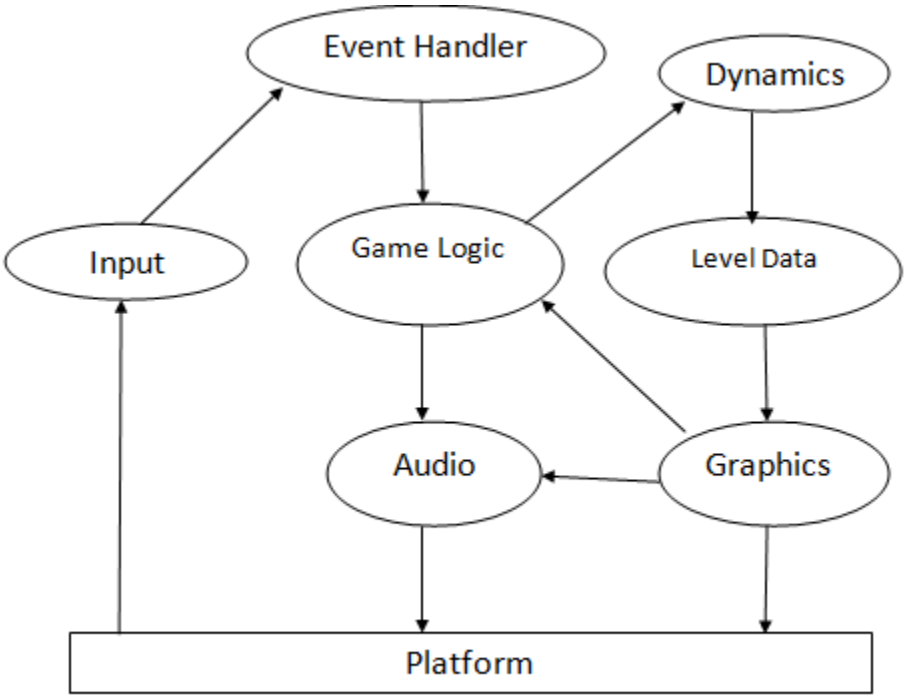

Figure 1: Game Architecture

The items with solid outlines are essential to every game while the dashed outlines refer to modules that are found in more complicated and demanding games. The game logic is the part that holds the game‘s story. The audio and graphics are the modules that help the writers narrate the story to the player. The event-handler and the input modules, supply the game logic with the player‘s next action. The level data module is a storage module for details about static behavior and the dynamics module configures dynamic behavior of game‘s characters.

1.3 Object-oriented design patterns in game logic

Although until now there is not much work found on object-oriented design patterns‘ use in games, we believe that such a use can be proven very useful in this domain.

This fact can be examined by investigating the source code of games for the existence of design patterns. As a first approach, we will provide examples of how object-oriented design patterns could be used in simple games. In addition to that, in Section 4 we will present real games that could use these object-oriented design patterns and improve their design. The strategy pattern, defines a family of algorithms, encapsulates each one, and makes them interchangeable.

2. On frame rate and player performance in first person shooter games

2.1 Summary

The rate at which frames are rendered in a computer game directly impacts player performance, influencing both the game playability and enjoy ability. However, despite the importance of frame rate and the wide-spread popularity of computer games, to the best of our knowledge, there is little quantitative understanding of the effects of frame rate on player performance in computer games. This paper provides a unique classification of actions in First Person Shooter (FPS) games based on interaction requirements that allow qualitative assessment of the impact of frame rates on player performance. This qualitative assessment is supported by quantitative analysis from two large user studies that measure the effects of frame rate on the fundamental player actions in a FPS game. Nearly 100 users participated in the two user study experiments, providing performance and perception data over a range of frame rates commonly studied for video streaming and inclusive of frame rates found in many computer game platforms. In general, the analysis shows that actions that require precise, rapid response, such as shooting, are greatly impacted by degradations in frame rates, while actions with lower precision and response requirements, such as moving, are more tolerant of low frame rates. These insights into the effects of frame rates on player performance can guide players in their choice for game settings

and new hardware purchases, and inform system designers in their development of new hardware.

2.2 Collision detection:

Collision detection is an important part of the interaction function of 3D games. According to the result of the collision detection, by triggering different interaction effect, game system gives timely feedback, which can make the players get more vivid and real game experience. While in the game physics engine is used to calculate collisions between objects, if the calculation is relatively complex, which leads to increased responding time and the overall system may slowdown.AI enemies in this game, for example, the enemy will automatically detect the location of the protagonist and approach. When collision detection occurs, the enemy will attack. At the beginning of the original game, it will appear automatically every 5 seconds one enemy. But when the enemy number is up to 20, the game can not run almost. It really shows that complex physics calculation has a greater influence on the performance of the game. If set the maximum number of enemy for six, when the enemy dies automatically appear new enemies, the performance of ascension is inevitable. Most of the earlier work in collision detection has focused on algorithms for convex polytopes. A number of algorithms with good asymptotic performance have been proposed in the computational geometry literature. Using hierarchical representations, an $O(\log^2 n)$ algorithm is given in [DK90] for polytope-polytope overlap problem, where *n* is the number of vertices. This elegant approach has not been robustly implemented in 3D, however.

Good theoretical and practical approaches based on linear complexity of the linear programming problem are known . Minkowski difference and convex optimization techniques are used in to compute the distance between convex polytopes by finding the closest points.

In applications involving rigid motion, geometric coherence has been exploited to design algorithms for convex polyhedra based on local features These algorithms exploit the spatial and temporal coherence between successive queries and work well in practice.

A number of hierarchies have been used for collision detection between general polygonal models. Typical examples of bounding volumes include axis-aligned boxes

(cubes are a special case) and spheres, and they are chosen for their fast overlap tests. Other structures include cone trees, k-d trees and octrees sphere trees trees based on S-bounds etc. Binary space partitions and
extensions to multi-space partitions , and spatial partitionings based on space-time bounds or four-dimensional testing have been used. All of these hierarchical methods do very well in performing ``rejection tests'' whenever two objects are far apart. However, when the two objects are in close proximity and can have multiple contacts, these algorithms either use subdivision techniques or check very large number of bounding volume pairs for potential contacts. In such cases, their performance slows down considerably.

3. Research On Intelligent 3D Path Finding In Game Development

3.1 Summary

For the path-finding problem of movement attitude changes with flight direction (or the swimming direction) of the object or role in the three-dimensional games, this article proposed the off-surface path finding algorithm in three dimensional game, through improve and optimize the A *algorithm in two-dimensional path finding, and this way can meet the requirements of calculation in three-dimensional.

Experiments show that the optimized algorithm meet the optimization of three-dimensional grid nodes ,set of obstacles in the three-dimensional scene, modify and optimization of the valuation function , computing of node coordinates, maintenance of OPEN table and CLOSED table. It has application and extending value to the study of path finding in the three-dimensional scene aimed at the cling surface object motion simulation**.**

3.2 Comparison of Path finding algorithms

There are many mature path finding algorithms in game development, and most of them belong to the category of state space search. State is a mathematical description of the progress of a problem in some time, state space search is process to go through all the nodes in the state space, to find out a path from the start node to target node, can be divided into two broad categories of blind search and heuristic search. There are many mature path finding algorithms in game development, and most of them belong to the category of state space search. State is a mathematical description of the progress of a problem in some time, state space search is process to go through all the nodes in the state space, to find out a path from the start node to target node, can be divided into two broad categories of blind search and heuristic search.

3.3 The main idea of A * algorithm

In variety of heuristic search algorithms, A * algorithm is one of the more mature algorithms. The idea is that when state space search, assess each node searched by evaluation function to guide the search forward direction, until find the target node. General form of evaluation function is: f (n) = g (n) + h (n), where, f (n) is the evaluation function, g (n) is currently known shortest path moving from the starting point S to the node n along the path generated, h (n) is estimated moving cost moving from node n to the end point D. Evaluation function definition has no certain model, in A * algorithm, h (n) have been restricted, so for all nodes x are h (x) $\leq$ h * (x), where h * (x) is minimum cost from the node X to the target node, namely actually the shortest distance. This evaluation function designed like this can find out the shortest path, is called admissibility of path finding, and A * algorithm is an adopted path finding algorithm. In the A * algorithm two chained lists will be used, OPEN lists and CLOSED lists, were used to save the node to be examined and nodes do not need to be examined again differently. The specific path searching process is as follows:

In the A * algorithm two chained lists will be used, OPEN lists and CLOSED lists, were used to save the node to be examined and nodes do not need to be examined again differently.

The specific path searching process is as follows:

1) Starting from the starting point S and put it into the OPEN list as the first node to be processed, this time, S is also called the current node.

2) detection 8 adjacent nodes around the current node, to find out the node x which can be reached or through, if x is in the OPEN list (or the CLOSED list), check that whether g (x) value with the new path to the x, is less than the original g (x) value of x. If so, set the current node as the parent node of node x, and recalculate g (n) and f (n) value of x node. Otherwise, do not make any modifications with the node x. If x is not in OPEN list (or the CLOSED list), add them to OPEN list, calculate g (n) and h (n) value of x , and set the current node as the parent node of node x. 3) Remove the current node from the OPEN list, add it to the CLOSED list. 4) In order to continue to search, select a node which has the minimum evaluation function f(n) value from the OPEN list as a new current node, if the OPEN list is empty, said the path does not exist from the beginning to the end. If the new current node is the destination D, then the search end, and backdate from the starting point to end point according to the father-son relationship, get the request path from S to D. Otherwise, return (2) to continue the search. Three-dimensional gaming industry has developed rapidly, industry prospects, research of game-related technology is more in-depth, in particular put forward higher requirements to solve three-dimensional scene path finding namely three-dimensional path search. For the three-dimensional path finding technology need in three-dimensional game , the author proposed the off-surface path finding algorithm in three-dimensional game, through improve and optimize the A *algorithm in two-dimensional path finding. The Improvement and optimization of the algorithm include optimization of three-dimensional grid nodes , set of obstacles in the three-dimensional scene, modify and optimization of the valuation function , computing of node coordinates, maintenance of OPEN table and CLOSED table. Solve the path finding problem of movement attitude changes with flight direction (or the swimming direction) of the object or role in the three-dimensional games.

Chapter 3: Algorithms Description

1. The BSP Algorithm

At the dawn of computer game development there were many problems to be solved. Possibly the greatest mischief was rendering on home computers, which at the time did not posses too much computation power. This was the reason why different techniques of scene management had to be developed: one of them was a method that involved the Binary Space Partitioning algorithm (BSP) invented in early 1970s at the University of Texas at Dallas. At first its purpose was to create representations of 3D objects at various research facilities, but it was soon discovered that it could be used to render complex 3D environments in real-time even on computers that did not have any additional hardware support for graphics processing, such as the home computers. What developers didn't know at that time, however, was that BSP would revolutionize the computer entertainment .

Before further explanation of the BSP algorithm some basic terminology must be explained. In geometry there are 2 types of polygons that are especially important in 3D graphics: the convex polygons, and conclave polygons. Convex polygons words are a type of primitives that do not have any dents. This means that the inner angles of convex polygon are never greater than 180 degrees. A polygon that doesn't meet these demands is classified as a conclave polygon.

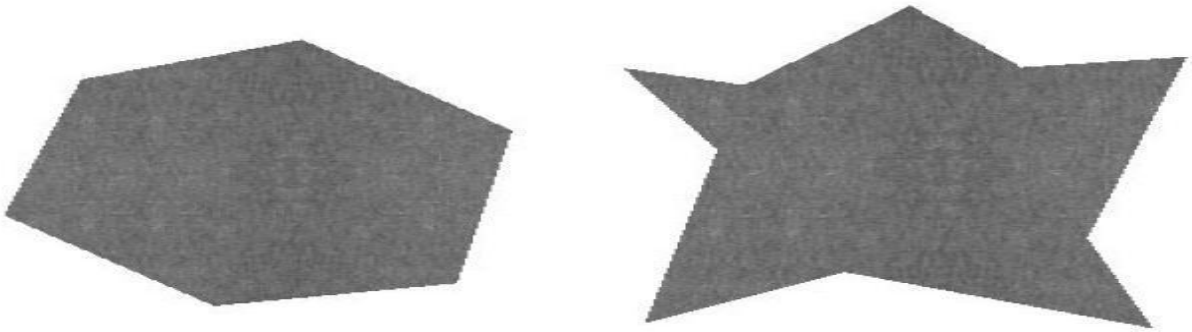

Figure 2: A convex (left) and a conclave (right) polygon

These terms can be easily translated into 3D space. If we were to be locked in a convex room, we would be able to see its every corner, no matter where we would stand (we disregard the fact that we would have to move our heads in order to see what's right behind us). Analogically, in a conclave room there would be certain areas, which could be only seen when standing in certain positions (it would be possible to hide from the viewer, hence some areas would be occluded). The same rules also apply to groups of convex or conclave polygons. This means that a group of convex polygons never occlude each other – a key property, that BSP algorithm employs.

The idea behind the BSP algorithm is very straightforward. Its objective is to split the game geometry into convex partitions using arbitrary partitioning planes. Each split results in two distinct groups: Geometry behind the splitting plane (called the backlist) and geometry in front of the splitting plane (called the front list). Each resulting set is then again partitioned using new partitioning plane. We perform these steps as long as the result of a split can produce a backlist and a front list – should the result contain only one of them, there is no further way of splitting the geometry.

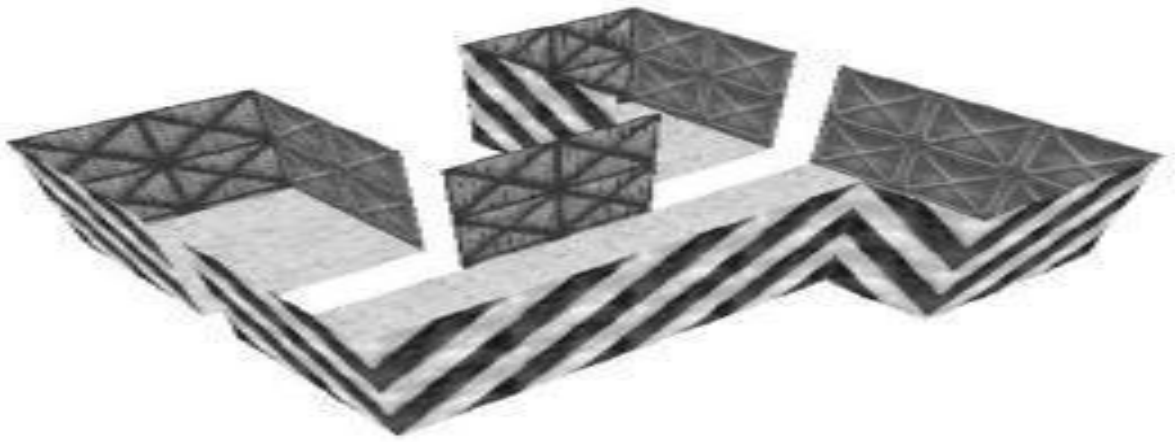

Figure 3 Shows the creation of a BSP tree for a simple game map.

Rendering is the process of generating an image from a 2D or 3D model (or models in what collectively could be called a *scene* file), by means of computer programs. Also, the results of such a model can be called a rendering.

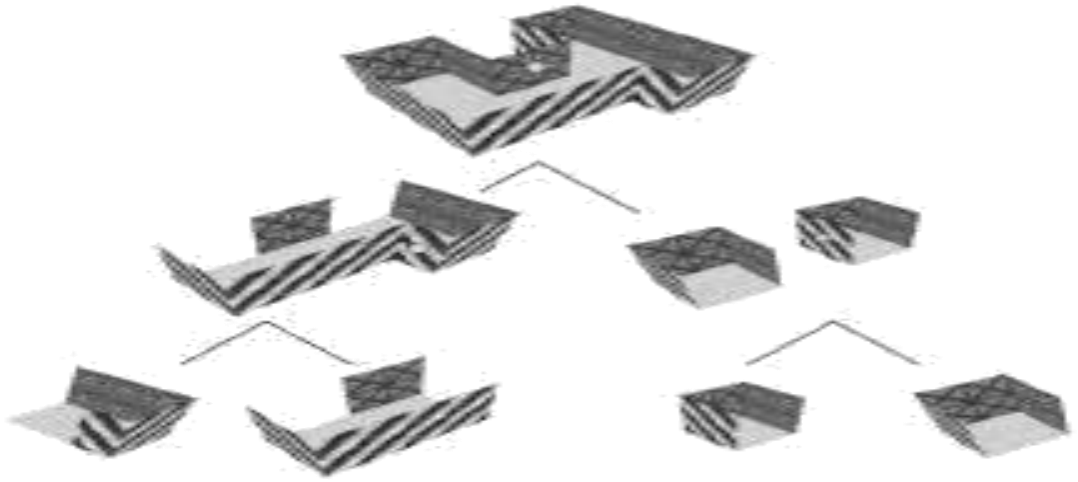

Figure 4: Demonstration of how the BSP algorithm works .

The idea of ―front‖ and ―back‖ for a given plane is usually solved by calculating a normal vector for the plane. By definition, these vectors always point outwards the surface and are aligned at the angle of 90 degrees to the surface, therefore they are very often used in graphics rendering for determination whether we are facing the front or the back of the surface. This is especially important during the rendering process when we want to skip the surfaces that the viewer cannot see. While the BSP algorithm is easy to perform, there are more problems than meets the eye at first. In real life application it is important to choose the best splitting plane in order to get the BSP tree in the shortest time possible. Also, it is always good to get a balanced BSP tree, which would keep the search time at pretty much the same level for every rendered part of geometry.

The BSP tree is created by inserting each segment in numbered order into the tree. In order to allow the user to more easily understand the demo, no attempt is made to select the BSP tree that produces the minimum splitting of segments. This is normally done because it minimizes the size of the tree and makes it more efficient. In addition, no attempt is made to produce a balanced (or nearly balanced) tree, which would also normally be desirable since it prevents degenerate cases such as those where the depth of the tree is approximately equivalent to the number of partitions.

Because each partition must be classified with respect to O(lg(n)) other partitions, the expected running time for constructing this BSP tree is O(n*lg(n)).

BSP Algorithm is used to render complex 3D environments in real-time even on computers that did not have any additional hardware support for graphics processing, such as the home computers.

- Start with a set of polygons and an empty tree
- Select one of them and make it the root of the tree
- Use its plane to divide the rest of the polygons in 3 sets: front, back, coplanar. Any polygon crossing the plane is split
- Repeat the process recursively with the front and back sets, creating the front and back sub trees respectively

2. The Rendering Algorithm

The pseudo-3D scene is rendered by classifying the eye point with respect to the root segment, then recursively drawing all segments on the same side of that segment. If the eye point intersects a segment, we‘re seeing it edge-on, so it isn‘t drawn. Because each segment is visited exactly once while drawing the scene, the scene can be rendered in *O(n)* time. Here is pseudo code for a method on a BSP tree node class that implements this algorithm:

```
draw3DScene()
if location(eye. point) =front
Side back.draw3DScene()
draw Polygon()
front.draw3DScene()
else if location(eye. point) ==
backside front.draw3DScene()
draw Polygon()
back.draw3DSc
ene()
else
front.draw3DScene()

back.draw3DScene()
```

When we‘re ready to draw the scene, the eye point and look vector are used to determine the coordinate system for the camera space. Then each polygon is drawn by using the following algorithm:

- Draw Polygon() transform segment endpoints to camera space clip segment to view frustum
- convert segment to polygon:
- set the width to the x values scaled down by the y values
- set the height to a constant value scaled down by the y values

3. Discretized Space Algorithm

In collision detection algorithm for computing all the contacts between multiple moving objects in a large environment. It uses the visibility pruning algorithm . The overall algorithm is general and applicable to all environments. We also highlight many optimizations and the visibility queries used to accelerate the performance of our algorithm.

1. Algorithm: DISCRETIZE SPACE

- Assume the Configuration Space has some fixed size in all its dimensions. Discretized each dimension so that it has a fixed number of cells. For each cell whose center is inside an obstacle in the Configuration Space, mark it Impassable.
- Likewise, for each cell whose center is outside an obstacle, mark it **Passable**.
- Each Passable cell is now a Node.
- Each Node connects to all its "adjacent" Passable neighbors in the graph.

3.1 Potentially Colliding Set (PCS)

We compute a PCS of objects that are either overlapping or are in close proximity. If an object does not belong to the PCS, it implies that does not collide with any object in the PCS. Based
on this property, we can prune the number of object pairs that need to be checked for exact collision.
This is similar to the concept of computing the potentially visible set (PVS) of primitives from a viewpoint for occlusion culling 4. We perform visibility computations between the objects in image space to check whether they are potentially colliding or not. Given a set *S* of objects, we test the relative visibility of an object *O* with respect to *S* using an image space

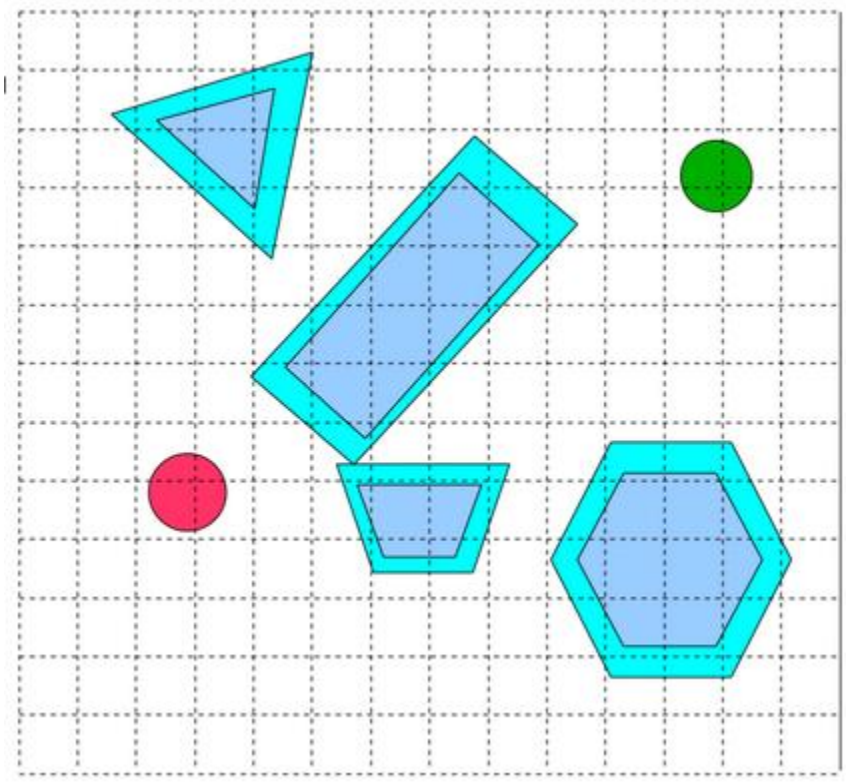

Figure 5:Showing the Discretized Space Algorithm

3.2 Sub-Object level pruning

We perform multiple level pruning to identify the potentially intersecting triangles among the objects in the PCS. We group adjacent local triangles (say *k* triangles) to form a sub-object used in multi-level pruning and prune the potential regions considerably. This improves the performance of the overall algorithm because performing a fully-visible query for each single triangle in the PCS of objects can be expensive. At the next level, we consider the PCS of sub objects and perform pruning using each triangle as a sub object. The multiple-level sub-object pruning is performed across each axis.

3.3 Intersection Tests

We perform exact collision detection between the objects involved in the potentially colliding pairs by testing their potentially intersecting triangles

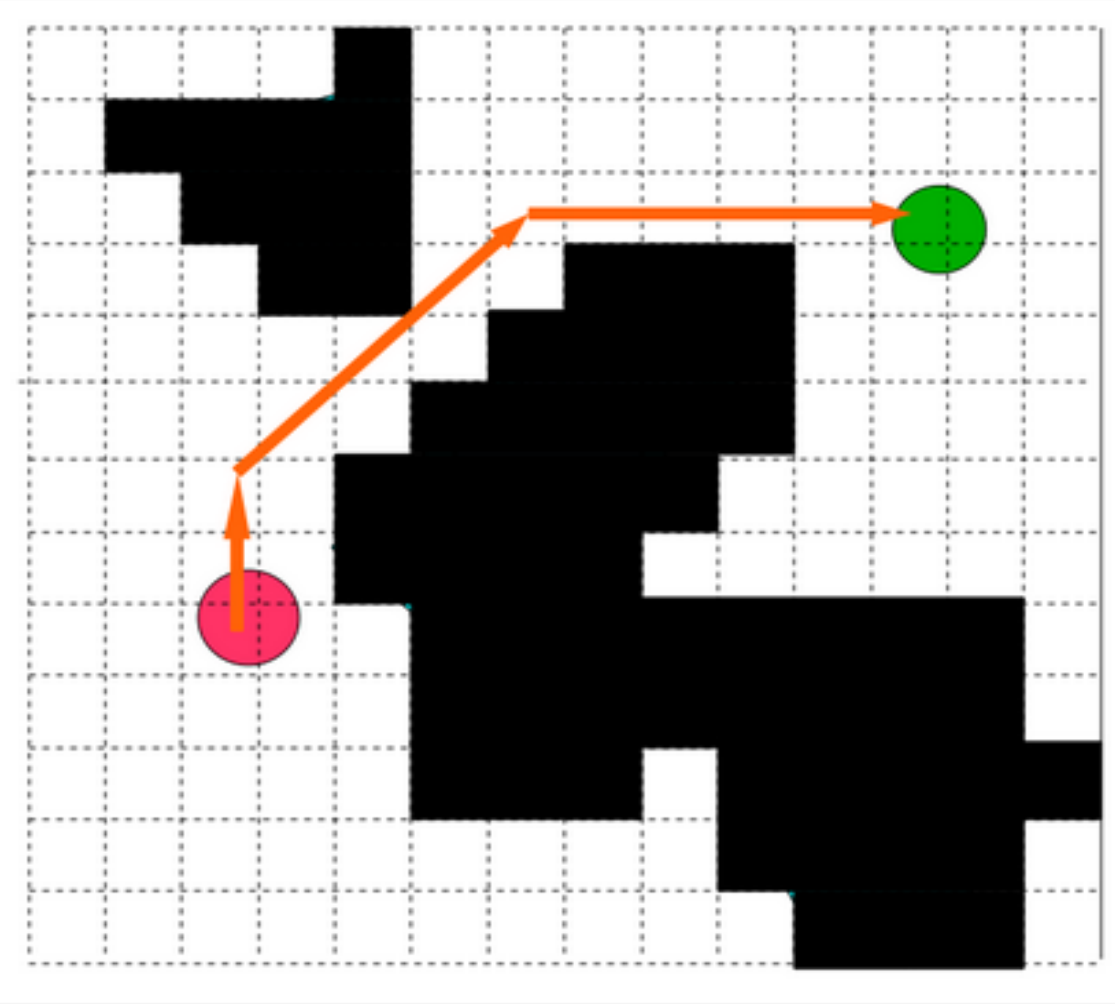

Figure 6:Showing Path to move between obstacles

CHAPTER 4: DESIGN AND IMPLEMENTATION

1. Game Development Pipeline

Game development pipeline is the process of being developed, provided, or completed; in the works; under way. Following sequence of operations required to move art assets from concept to the finished product.

- Experience design—UI, game play, narrative
- Cinematic
- Artwork—3D modeling, surfacing, lighting, environment, animations, sound
- Programming—graphics, UI, physics, AI, networking, game tools
- Level design
- Sound engineering
- Testing
- Maintenance (bug fixing, adding levels and features)

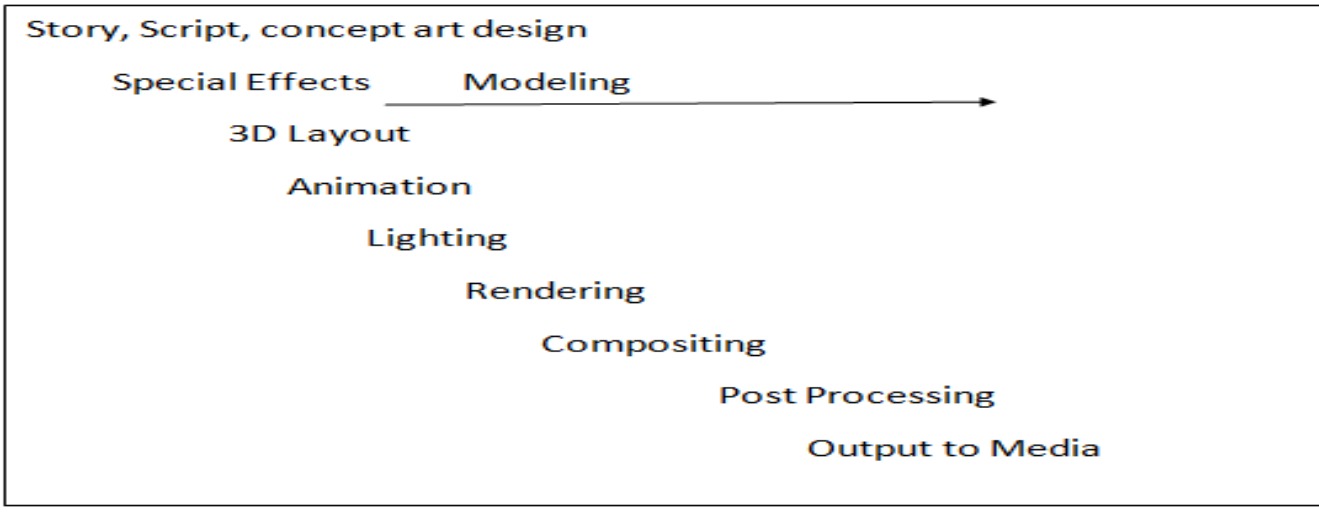

Figure 7:Game Development Pipeline

2. Flow Chart

2.1 Flow Chart showing flow of work

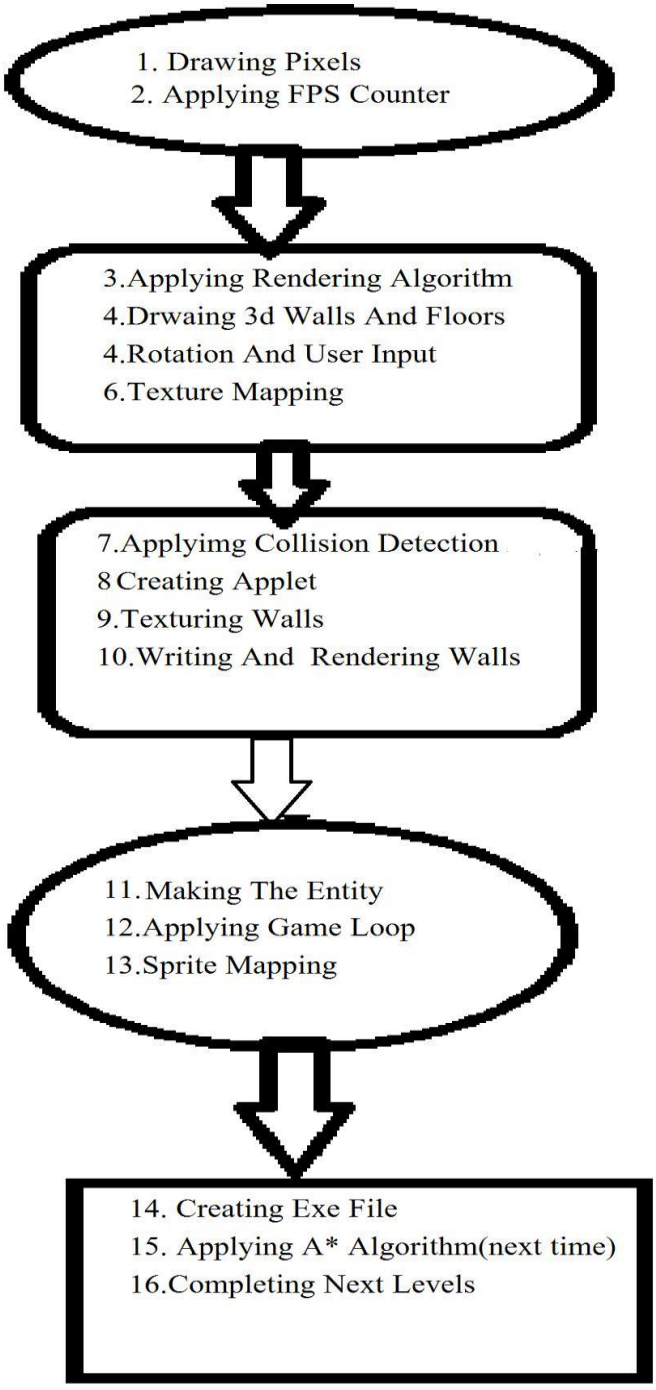

Figure 8:Flow Chart Showing flow of work

3. Game Design Models

3.1 Definition

3D game design models represent a 3D object using a collection of points in 3D space, connected by various geometric entities such as triangles, lines, curved surfaces, etc. Being a collection of data (point and other information), 3D models can be created by hand, algorithmically (procedural modeling), or scanned.

3D models are widely used anywhere in 3D graphics. Actually, their use predates the widespread use of 3D graphics on personal computers. Many computer games used pre-rendered images of 3D models as sprites before computers could render them in real-time.

Today, 3D models are used in a wide variety of fields. The medical industry uses detailed models of organs; these may be created with multiple 2-D image slices from an MRI or CT scan. The movie industry uses them as characters and objects for animated and real-life motion pictures. The video game industry uses them as assets for computer and video games. The science sector uses them as highly detailed models of chemical compounds. The architecture industry uses them to demonstrate proposed buildings and landscapes through Software Architectural Models. The engineering community uses them as designs of new devices, vehicles and structures as well as a host of other uses. In recent decades the earth science community has started to construct 3D geological models as a standard practice. 3D models can also be the basis for physical devices that are built with 3D printers or CNC machines

3.2 Game Development Cycle

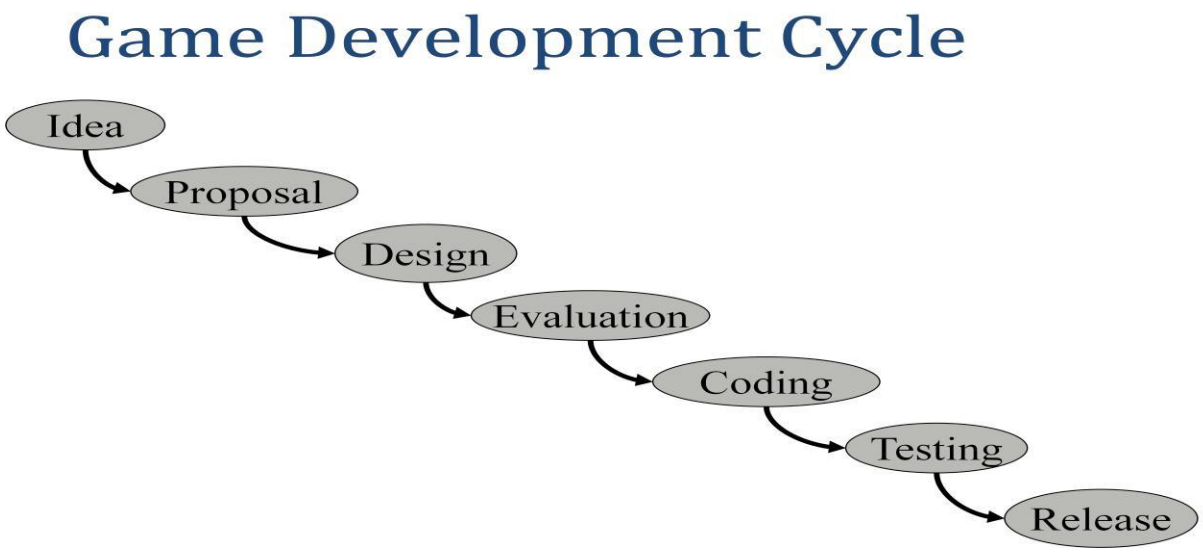

Figure 9:Game Development Cycle

3.2.1 STEP ONE: INITIAL PLANNING

The Genesis Gaming Design and Marketing teams will meet with the client to determine the key concepts driving the development of a strategic game portfolio. This will address issues such as analysis of an existing portfolio, current demographics, additional player acquisition and retention, emerging trends and profiles.

We will further discuss a range of themes, volatility levels, features, bonus games and any other aspect required for bespoke development of a strategic portfolio. It is also important that there is an overall marketing discussion to determine how the games can be used to further extend the client‘s brand.

3.2.2 STEP TWO: INITIAL THEMES & CONCEPT ART

In this phase, submit themes, names, concept art and descriptions for consideration and approval or modification. We will then work with the client to determine portfolio development priorities. This allows the ability to set expectations as to game delivery. The client can then implement a schedule for marketing and release.

3.2.3 STEP THREE: FEATURES AND MATHEMATICS

In this, unique math models that best represent the client‘s objectives as stated through the analytical process and the client‘s objectives. Math will be verified and all required percentages will be provided.

3.2.4 STEP FOUR : ART AND CREATIVE DESIGN

In this, static art and design elements for approval. Subsequently, our Animation and Music Composition teams will finalize the game. This will require the client to provide appropriate documentation such as templates and other substantive game specifications for consistency. Upon completion, our Demo team will provide a playable version of the game for additional review.

3.2.5 STEP FIVE: INTEGRATION

In this, integrate the game to the appropriate provider platform. This requires the necessary documentation, such as an API, from the client along with server support, assuming it is not an

asset only‖ delivery. We place a significant emphasis on product assurance and quality control so the game will be very client provider friendly.

3.2.6 STEP SIX: DELIVERY

All game assets, in the requested file formats, including all release documentation will be provided according to contractually specified delivery requirements. Genesis Gaming is also happy to provide any material that may be requested for the client‘s marketing campaign.

Since success is our mutual objective, it is our goal to make certain that the client has all tools possible to promote the best exclusive content in the industry.

4. Game Structure and Design

Game will consist of three levels and before levels, there will be a menu layer that will consist of game options, sound options and video options, then there will be a loading screen. Main game frame will consists of introduction, game play and pause options. After main game ,there will be game over and victory screen if player will pass through all three levels within a given time.

4.1 Game Structure:

- Start up
- Menu Layer
- Front End
  - Game Options
  - Sound Options
  - Video Options
- Loading Screen
- Main Game
  - Introduction
  - Game play
  - Pause Options
- Game Over
- Victory Screen
- Shut down

In this game, user can choose the menu options consisting of game options, sound options .

In the main game part, first of all introduction about the game will be there, then user can control the entity in the first level that is in 3-d maze to find the clues with the help of keyboard options .

4.2 Game Flow Chart:

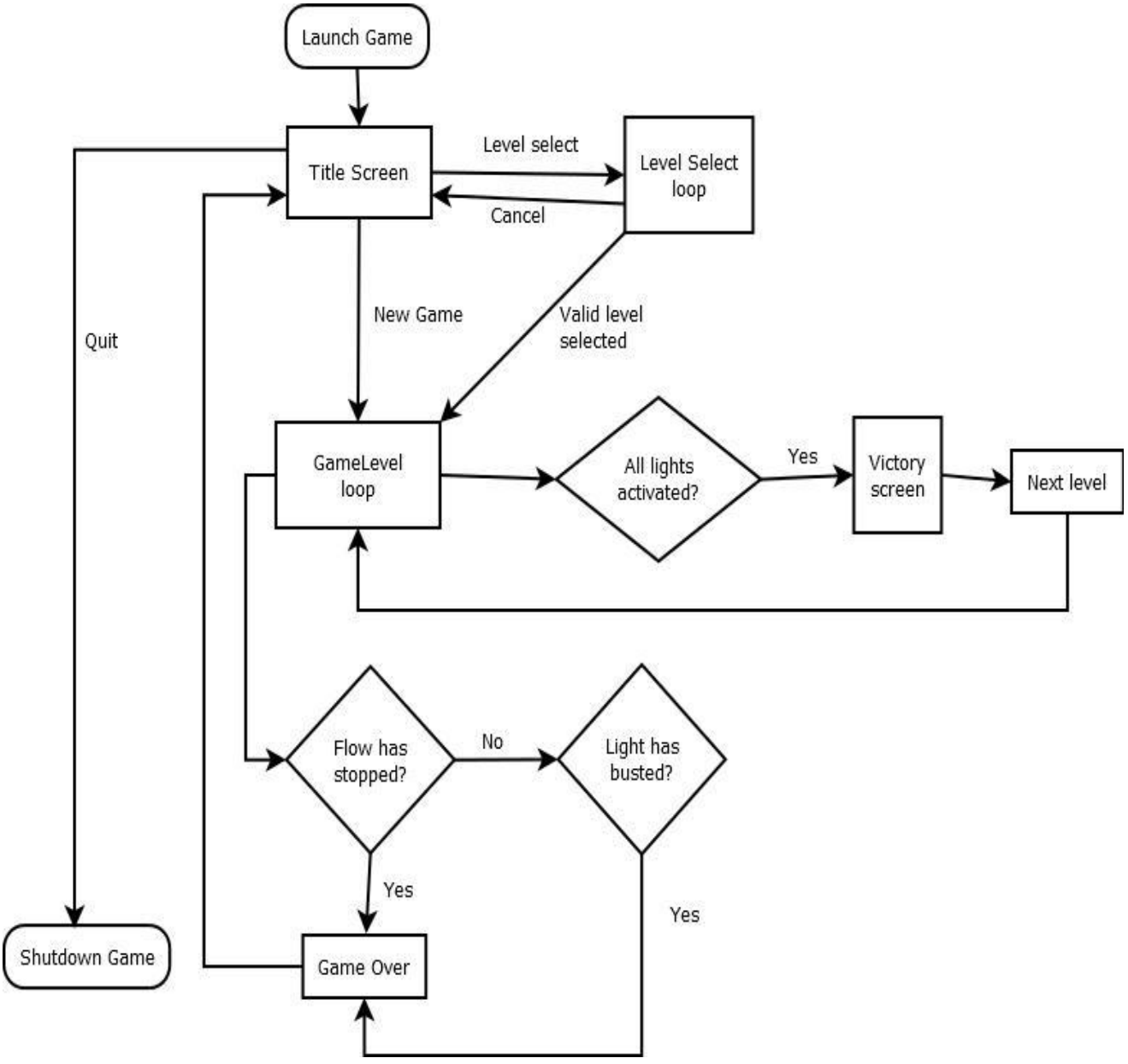

Figure 10: Game Design

4.3 Implementation Steps

- First Step is to draw the pixels on the screen by using the simple geometry methods by using functions of math‘s and by using buffered strategy techniques.

- Second step is to apply the BSP tree algorithm and draw the 3d world that is the requirement of the first level .

- Third step is to apply the rendering the 3d walls and floor to give the 3d effects
- And also applying rendering algorithm using FPS counter and by using the java inbuilt functions.

- Fourth step is to apply the rotation on the given 3d maze .

- Fifth step is to give the input to user to walk across the maze by using the keyboard options by implementing the mouse listener classes.

- Next step is to draw the entity and setting the camera position on its head.

- Next step is to set the victory screen and to declare the winner according to the game logic.

- For the Future, implementation of the AI algorithm will be there.

4.4 Snapshots of the first level of the 3d game

4.4.1 Snapshot-1: Showing the front launcher of the game

Figure 11:Front Launcher of game

4.4.2 Snapshot-3: Showing the 3d maze with clues of first level

Figure 12:Player Inside The Game Map

4.4.3 Snapshot-2: Showing entities of 3d maze first level

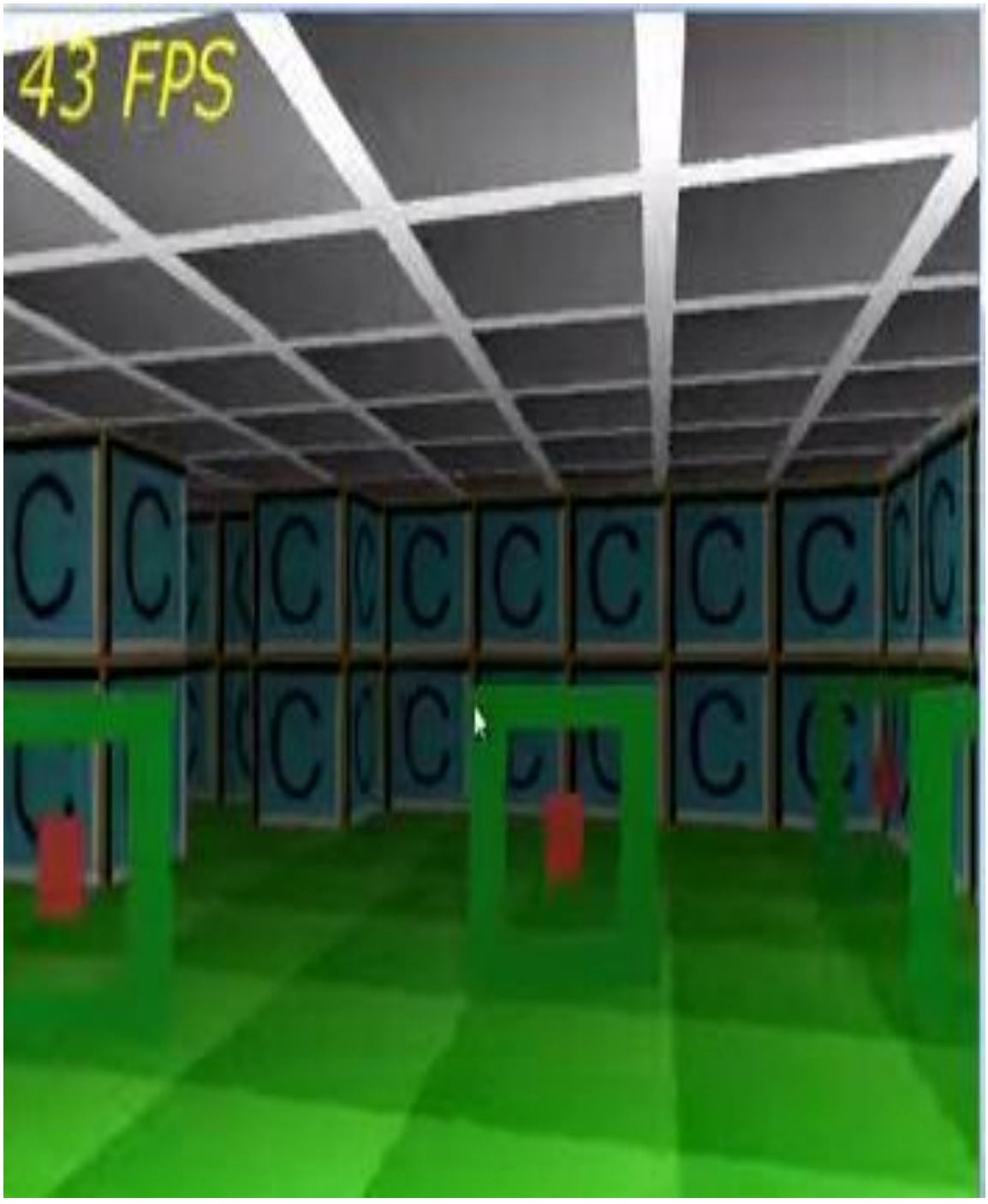

Figure: 13:3d maze with entities

Till now, I have completed the level-1 of my game successfully and the control is given to user as follows:

W-forward

A-Left

S-Backward

D-Right

Up-arrow-left
forward Down
arrow-right
forward.

CONCLUSION

My work presents a in-depth background study of that how to develop a 3d game .It includes understanding the rendering, rotation and also how to use the FPS Technologies.

It also includes study of how to develop a game using Java 3d API by using the Applet, Swing, Graphics functions.Also,this study further purposes how to include the AI part in the game.

FUTURE WORK

- Future work includes implementing the Discretized Space algorithm proposed
- The completion of next 2 levels of the 3d game.
- Further implementing the prototype developed in a real life scenario
- Measuring the efficiency of the algorithm proposed.

REFERENCES

[1] Apostolic Ampatzoglou, Alexander Chatzigeorgiou "Evaluation of object-oriented design patterns in game development". Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece, 2006,10.

[2] Krzysztof Kondrak" Design and implementation of application independent easy-to-use game engine". Department of Computer and Information Science, 2009,41.

[3] Miao Wang**,** Hanyu Lu,"Research On Algorithm Of Intelligent 3D Path Finding In GameDevelopment".InternationalConferenceonIndustrialControlandElectronics Engineeing,2012,5.

[4]Daniel Selman, Textbook of "Java 3D Programming", 2011,352.

[5] Staffan Björk**,** Jussi Holopainen, **"**Describing Games -An Interaction-Centric Structural Framework". Nokia Research Center,Finland,2010,13.

[6] Kyle Ingols, Aileen Tang, Lawrence Wang," Adventure Game: Navigating The Jungles Of MIT".6.001,2014,12.

APPENDIX

Code:

Display.java

```
package com.mine.minegame;

import java.awt.*;
import java.awt.image.BufferStrategy;
import java.awt.image.BufferedImage;
import java.awt.image.DataBufferInt;
import java.io.File;
import java.io.IOException;

import javax.imageio.ImageIO;

import com.mine.minegame.graphics.show;
import com.mine.minegame.input.InputHandler;

public class Display extends Canvas implements Runnable {

	private static final long serialVersionUID = 1L;
	public static final int WIDTH = 900;
	public static final int HEIGHT = 700;
	public static final String title = "My First game";

	private Thread t;
	private show sh;
	private G1 game;
```

```
        private BufferedImage img,img1,img2,img3,img4,img5;

        private boolean running = false;

        private int[] pixels;
        private InputHandler input;


        public Display()
        {
                sh=new show(WIDTH,HEIGHT);
                game =new G1();

                img=new
BufferedImage(WIDTH,HEIGHT,BufferedImage.TYPE_INT_RGB);
                try {
                        img1 = ImageIO.read(new File("res/Play1.gif"));
                        img2 = ImageIO.read(new File("res/1.jpeg"));
                        img3 = ImageIO.read(new File("res/2.jpeg"));
                        img4 = ImageIO.read(new File("res/3.jpeg"));
                        img5 = ImageIO.read(new File("res/4.jpeg"));

                } catch (IOException e) {
                        e.printStackTrace();
                }

                pixels=((DataBufferInt)img.getRaster().getDataBuffer()).getData();
          input =new InputHandler();
          addKeyListener(input);
          addMouseListener(input);
          addFocusListener(input);
```

```
    }

    public synchronized  void start() {
        if (running)
            return;
        running = true;
        t = new Thread(this);
        t.start();

    }

    public synchronized void stop()
    {
        if(!running)
            return;
        running=false;
        try
        {
            t.join();
        }
        catch(Exception e)
        {
            e.printStackTrace();
            System.exit(0);
        }



    }
public static void main(String[] args) {
```

```
        new Launch();

}


public void run() {
        int frame =0;
        double ups=0;
        long pt=System.nanoTime();
        double spt=1/40.0;
        int tc=0;
        boolean ticked=false;

        while(running)
        {
                long ct=System.nanoTime();
                long pat=ct-pt;
                pt=ct;
                ups+=pat/1000000000.0;
                while(ups>spt)
                {
                tick();
                ups-=spt;
                ticked=true;
                tc++;
                if(tc%60==0)
                {
                        System.out.println(frame+"fps");
                        frame=0;
                        }
```

```
                }
                if(ticked)
                {
                render();

                frame++;

        }


}
}


        private void render() {
        BufferStrategy br=this.getBufferStrategy();
        if(br==null)
        {
                createBufferStrategy(3);
                return;



        }
        sh.render(game);
        for(int i=0;i<WIDTH*HEIGHT;i++)
        {
                pixels[i]=sh.pixels[i];

        }
        Graphics g=br.getDrawGraphics();
        g.drawImage(img, 0, 0, WIDTH, HEIGHT, null);
```

```
                g.drawImage(img1, 0, 400, 200,200, null);

                g.drawImage(img2, 0, 0, 100,100, null);
                g.drawImage(img3, 100, 0, 100,100, null);
                g.drawImage(img4, 200, 0, 100,100, null);
                g.drawImage(img5, 300, 0, 100,100, null);



                g.dispose();
                br.show();


        private void tick() {
                game.tick(input.key);

        }
```

Render3d.java

```
package com.mine.minegame.graphics;
import com.mine.minegame.input.Controller;
import com.mine.minegame.level.Block;
import com.mine.minegame.level.Level;

import java.util.Random;

import com.mine.minegame.G1;


public class Render3d extends Render{
 public double[] zBuffer;
 Random random=new Random(100);
```

```
 private double renderdistance=5000.0;
 private double forward,right,cosine,sine;
        public Render3d(int width, int height) {
                super(width, height);
                zBuffer =new double[width*height];

        }

        public void renderWall(double xLeft, double xRight, double zDistanceLeft,double
zDistanceRight, double yHeight) {
                G1 game=new G1();
                double up=game.controls.y;
                double xcLeft = ((xLeft) - right) * 2;
                double zcLeft = ((zDistanceLeft) - forward) * 2;

                double rotLeftSideX = xcLeft * cosine - zcLeft * sine;
                double yCornerTL = ((-yHeight) - up) * 2;
                double yCornerBL = ((+0.5 - yHeight) - up) * 2;
                double rotLeftSideZ = zcLeft * cosine + xcLeft * sine;

                double xcRight = ((xRight) - right) * 2;
                double zcRight = ((zDistanceRight) - forward) * 2;

                double rotRightSideX = xcRight * cosine - zcRight * sine;
                double yCornerTR = ((-yHeight) - up) * 2;
                double yCornerBR = ((+0.5 - yHeight) - up) * 2;
                double rotRightSideZ = zcRight * cosine + xcRight * sine;

                double xPixelLeft = (rotLeftSideX / rotLeftSideZ * height + width / 2);
                double xPixelRight = (rotRightSideX / rotRightSideZ * height + width /
2);
```

```
                if (xPixelLeft >= xPixelRight) {
                        return;
                }

                int xPixelLeftInt = (int) (xPixelLeft);
                int xPixelRightInt = (int) (xPixelRight);

                if (xPixelLeftInt < 0) {
                        xPixelLeftInt = 0;
                }
                if (xPixelRightInt > width) {
                        xPixelRightInt = width;
                }
                double yPixelLeftTop = (int) (yCornerTL / rotLeftSideZ * height + height
/ 2);
                double yPixelLeftBottom = (int) (yCornerBL / rotLeftSideZ * height +
height / 2);
                double yPixelRightTop = (int) (yCornerTR / rotRightSideZ * height +
height / 2);
                double yPixelRightBottom = (int) (yCornerBR / rotRightSideZ * height +
height / 2);


                for (int x = xPixelLeftInt; x < xPixelRightInt; x++) {
                        double pixelRotation = (x - xPixelLeft) / (xPixelRight -
xPixelLeft);

                        double yPixelTop = yPixelLeftTop + (yPixelRightTop -
yPixelLeftTop) * pixelRotation;
                        double yPixelBottom = yPixelLeftBottom + (yPixelRightBottom -
```

```
yPixelLeftBottom) * pixelRotation;



                        int yPixelTopInt = (int) (yPixelTop);
                        int yPixelBottomInt = (int) (yPixelBottom);

                        if (yPixelTopInt < 0) {
                                yPixelTopInt = 0;
                        }
                        if (yPixelBottomInt > height) {
                                yPixelBottomInt = height;
                        }

                        for (int y = yPixelTopInt; y < yPixelBottomInt; y++) {


                                        pixels[x + y * width] =0x43C6DB;


                                zBuffer[x + y * width] = 0;
                        }

                }
        }
           public void floor(G1 game) {

                double fp = 8;
                double cp = 8;
                forward = game.controls.z;
```

```
right = game.controls.x;
double up = game.controls.y;
double walking = Math.sin(game.time / 6.0) * 0.5;

double rotation =game.controls.rotation;
cosine = Math.cos(rotation);
sine = Math.sin(rotation);

for (int y = 0; y < height; y++) {
        double ceiling = (y + -height / 2.0) / height;

        double z = (fp + up) / ceiling;
        if (Controller.walk) {
                z = (fp + up + walking) / ceiling;
        }
        if (ceiling < 0) {
                z = (cp - up) / -ceiling;
                if (Controller.walk) {
                        z = (cp - up - walking) / -ceiling;

                }
        }

        For (int x = 0; x < width; x++) {
                double depth = (x - width / 2.0) / height;
                depth *= z;
                double xx = depth * cosine + z * sine;
                double yy = z * cosine - depth * sine;
                int xPix = (int) (xx + right);
                int yPix = (int) (yy + forward);
                zBuffer[x + y * width] = z;
```

```
                    pixels[x+y*width]=((xPix& 15)*16)|((yPix& 15)*16)<< 8;
                    if (z > 500) {
                        pixels[x + y * width] = 0;
                    }

                }
            }
             Level level = game.level;
        int size = 50;
        for(int xB = -size; xB <= size; xB++){
            for (int zB = -size; zB <= size; zB++){
                Block block = level.create(xB, zB);
                Block east = level.create(xB + 1, zB);
                Block south = level.create(xB, zB +1);

                if(block.solid ){
                    if(!east.solid){
                        renderWall(xB + 1, xB + 1, zB, zB +1, 0);

                    }

                    if(!south.solid ){
                        renderWall(xB + 1, xB, zB + 1, zB +1, 0);

                                                                    }

                } else {
                    if(east.solid){
                        renderWall(xB + 1, xB + 1, zB + 1, zB, 0);

                    }
```

```
                if(south.solid ){
                    renderWall(xB, xB + 1, zB + 1, zB +1, 0);

                }
    }

            }
        }


    for(int xB = -size; xB <= size; xB++){
for (int zB = -size; zB <= size; zB++){
    Block block = level.create(xB, zB);
    Block east = level.create(xB + 1, zB);
    Block south = level.create(xB, zB +1);

    if(block.solid ){
        if(!east.solid){
            renderWall(xB + 1, xB + 1, zB, zB +1, 0.5);
        }

        if(!south.solid ){
            renderWall(xB + 1, xB, zB + 1, zB +1, 0.5);
        }

    } else {
        if(east.solid){
            renderWall(xB + 1, xB + 1, zB + 1, zB, 0.5);
        }
```

```
            if(south.solid ){
                renderWall(xB, xB + 1, zB + 1, zB +1, 0.5);
            }
        }
    }
}}

                public void rdl()
        {
                for(int i=0;i<width*height;i++)
                {
                        int colour=pixels[i];
                        int brightness=(int)(renderdistance/(zBuffer[i]));
                        if(brightness<0)
                        {
                                brightness=0;
                        }
                        if(brightness>255)
                        {
                                brightness=255;
                        }
                        int r=(colour>>16)&0xff;
                        int g=(colour>>8)&0xff;
                        int b=(colour)&0xff;
                        r=r*brightness/255;
                        g=g*brightness/255;
                        b=b*brightness/255;
                        pixels[i]=r<<16|g<<8|b;


                }
```