MULTI-DIMENSIONAL RAYCASTING

A PROJECT REPORT

Submitted by

SHADRACH GIDEON S P	**311419104071**
K SURAJ	**311419104083**
SK WASI AHAMED	**311419104097**
ZAFFER AHMED H M	**311419104100**

in partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

MEENAKSHI COLLEGE OF ENGINEERING, WEST K.K NAGAR

ANNA UNIVERSITY: CHENNAI 600 025

MAY 2023

MULTI-DIMENSIONAL RAYCASTING

A PROJECT REPORT

Submitted by

SHADRACH GIDEON S P	**311419104071**
K SURAJ	**311419104083**
SK WASI AHAMED	**311419104097**
ZAFFER AHMED H M	**311419104100**

in partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

MEENAKSHI COLLEGE OF ENGINEERING, WEST K.K NAGAR

ANNA UNIVERSITY: CHENNAI 600 025

MAY 2023

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"MULTI-DIMENSIONAL RAYCASTING"** is the bonafide work of **"K. SURAJ (311419104083), SK WASI AHAMED (311419104097), ZAFFER AHMED (311419104100), SHADRACH GIDEON SP (311419104071)"** who carried out the project work under my supervision.

SIGNATURE

Mrs.D.SUDHA, M.E.,(Ph.D).,
HEAD OF DEPARTMENT

Computer Science and Engineering,

Meenakshi College of Engineering,

West K.K, Nagar Chennai – 78.

SIGNATURE

Ms.D.REJITHA, M.E.,
SUPERVISOR
Assistant professor

Computer Science and Engineering

Meenakshi College of Engineering,

West K.K. Nagar Chennai – 78.

Submitted for the Anna University Project VIVA-VOCE Examination held on _____.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

We sincerely express our gratitude to our esteemed Managing Trustee **Tmt.R.GOMATHI RADHKRISHNAN**, Managing Director **Mrs.JAYANTHI PRABAKARAN** and the authorities of **MEENAKSHI AMMAL EDUCATIONAL TRUST** for the patronage and parental care showered on our welfare rooted in the academic career.

We express our thanks to our principal **Dr.R.GANESAN,M.E., Ph.D.,** for his support and encouragement throughout our course of study.

We express our sincere thanks to **Mrs.D.SUDHA, M.E.,(Ph.D).,** Head of the Department, Department of computer science and Engineering for giving constructive ideas and valuable criticism on our project.

We immensely oblige to our internal project Co-ordinators **Dr.S.PUSHPARANI, M.E.,Ph.D.,** Assistant Professor, Department of Computer Science and Engineering, **Ms.D.SUDHA, M.E.,(Ph.D).,** Assistant Professor, Department of Computer Science and Engineering for their valuable suggestions, guidance and sustained interest in completing the project work successfully.

We thank our internal guide **Ms.D.REJITHA, M.E.,** for her suggestions, guidance and sustained interest in completing the project work successfully.

ABSTRACT

The creation of video games involves multidisciplinary processes that are not accessible to the general public. Currently, video game development environments are very powerful tools, but they also require an advanced technical level to even start using them. This project presents a 3D game development environment to propose an alternative model to reduce the technical complexity existing in these systems, presenting a data model and a game engine that allows fulfilling this goal. In order to test its capabilities, multiple games have been successfully implemented in the proposed environment. With this achievement, it can be stated that it is possible to create video games simply and adorably for the general public without giving up its potential and remarking that there is still a long way to go to reach democratization in the creation of video games and the need to continue working in this field.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

2D	TWO DIMENSIONAL
3D	THREE DIMENSIONAL
RAM	RANDOM ACCESS MEMORY
GPU	GRAPHICS PROCESSING UNIT
OS	OPERATING SYSTEM
FPS	FRAMES PER SECOND
PNG	PORTABLE NETWORK GRAPHICS
MP3	MPEG AUDIO LAYER 3
MPEG	MOVING PICTURE EXPERTS GROUP

CHAPTER 1

INTRODUCTION

1.1 GENERAL

Videogame development is a complex process requiring multidisciplinary knowledge and skills, from the artistic and narrative vision to the ability to solve the technological challenges. Games have been a part of people's lives since time immemorial. Over time, they also found themselves in the field of information technology. Nowadays, we can find many companies that develop video games for profit. The project is a platformer genre game, where the aim is to reach endpoint through obstacles and enemies in each stage. It was made using Python and it is a Windows Operating System game. Pygame provided tools were more than enough to develop a simple 3D game.

3D games are more attractive and visually more realistic, especially with smartphones and tablets gaining popularity each year increasingly. With the modules Pygame has to offer, developing 3D games is much easier now, compared to earlier. In this report, we will take a closer look at the new features of Pygame and explain how a game can be developed even with a basic knowledge about programming.

While working on the game, we were able to acquire a deep understanding of Python language. Although there were some syntaxes that were new, it was not hard to comprehend them. We created many different modules that control different aspects of our game.

The game that was developed was solely for study purposes and is used as an example. This report can also be used as a tutorial to create a basic 3D game from start to finish.

1.2 OBJECTIVE

The main objective of this project is developing a 3D shooting game with Pseudo 3D concept along with advance A.I. enemies, and create a working game engine for future ray-casted games. We also hope to bring positive effects to users like,

- Better memory
- Improving vision for users
- Ability to responds to any situation
- Improves social skills
- Increases curiosity and motivates learning
- Stress relief

1.3 SUMMARY

This game will allow the user to have a satisfying experience while enjoying the gameplay in the process.

CHAPTER 2

LITERATURE SURVEY

2.1 GENERAL

Literature survey gives the overall description of the reference papers that has been referred to design the application, using which the problems of the existing applications and technologies is identified. The methods to overcome such limitations are also recognized.

2.2 LITERATURE REVIEW

2.2.1 Classfication of Humans and Bots in Two Typical Two-player Computer Games

The aim of this paper is to explore whether or not the artificial intelligence (AI) embedded in traditional and typical computer games can be easily identified by players. We pay particular attention on AI simulation in simple games. Therefore, a 2D shooting game and a car racing game are developed by using different game AIs (game bots). Meanwhile, a Turing Test for computer game bots is implemented to test the abilities of above two computer games playing agents (bots) to simulate human game playing behaviour. For the 2D shooting game three types of game AIs, random moving, finite state machine (FSM) and neural networks (NN), are used as agents to test. The results demonstrate that more humanized AI shows better human game playing behaviour then others. For the race game a waypoints AI are developed.

The experimental results show that although the paths design have a variety of changes, only 15% players be cheated. Careful review of the player's controls shows that players are more often fine-tune the location of the car to center it in the

track. The research results can be seen as the based line of simple game AI development.

2.2.2 Research on of 3D game design and development technology

According to the great significance of 3D game technology, this paper will do detailed research on main technologies of 3D. Firstly, this paper puts forward to design concept, designs the main frame of 3D game. Secondly, through researching collision detection technologies, the improved collision detection method is proposed based on Aligned Axis Bounding Box according to some disadvantages of collision detection methods. The improved collision detection method effectively reduces operation time, advanced game speed. Thirdly, through researching artificial intelligence for 3D game, path search method is improved according to the needs of system. The improved idea uses restricting search range of A* search arithmetic with the shortest path. It can get intelligent path search efficiently and factually. Finally, the 3D game system is realized based on 3D engine technologies. The system is rendered smoothly, authentically, manipulated easily.

2.2.3 Virtual World of Video Games

This article is dedicated to the phenomenon of video games virtual world. We have overviewed specific literature that let us formulate the generalized definition of virtual world: computer- based three- or two-dimensional environment or space that can simulate real world where users represented by avatars are able to communicate or interact simultaneously or synchronically. This definition is quite limited when it comes to virtual non-ICT worlds. To understand the essence of virtual world we referred to postmodern philosophy that implies the possibility of multiple full-fledged worlds. We have formulated one thesis stating that virtual world is the universe of simulacra and the second thesis stating that virtual world is

fiction, imaginary and illusory, that becomes real and actual in the process of interaction between the Architect of a virtual world and the Beholders. The article proposes a new view on virtual world of video games.

2.2.4 Evaluation of Individualized HRTFs in a 3D Shooter Game

Previous research stresses the importance of Head-Related Transfer Function (HRTF) individualization approaches for accurately locating sound sources in virtual 3D spaces. However, in the realm of interactive experiences, methods for assessing whether individualized HRTFs bring a benefit to the player experience are rarely investigated. Methods to improve spatial audio rendering are needed now than ever since Virtual Reality (VR) is becoming a mainstream technology for interactive experiences. This paper proposes a method of using in-game metrics to test the hypothesis that individualized HRTFs improve the experience of both expert and novice players in a First-Person Shooter (FPS) game on a desktop environment. The FPS game provides players with a localization task across three different audio renderings using the same acoustic spaces: stereo panning (control condition), generic binaural rendering, and individualized binaural rendering.

2.2.5 A Classification of Visual Style for 3D Games

Graphic styles for 3D games can be designed in any direction as the designer desires. Graphic styles range from an abstract composing with a degree of independence from visual references to a realistic rendition of the real world. Cel shaded, voxel graphics, and photo-realism are terms that widely used to characterize the graphic style. Nevertheless, these designates are not systematically set and well prescribed since development techniques change rapidly due to hardware and software innovations. In order to visualize a chosen graphic style, numerous compositions, materials, and execution are required. The purpose of a game, as

defined by developers, can be game experience goal, player enjoyment, market capitalization, or specific objectives such as cognitive improvement.

2.3 SUMMARY

This regard, a survey of the above-mentioned literature works has been done and it gives the various techniques for Game design.

CHAPTER 3

SYSTEM ANALYSIS

3.1 GENERAL

System Analysis is important because it provides an avenue for solutions in the system through the various tasks involved in doing the analysis. This chapter explains in detail about the purposed system, in-depth along with some diagrams.

3.2 PSEUDO 3D

2.5D, 3/4 perspective and pseudo-3D are informal terms used to describe graphical projections and techniques that try to "fake" three-dimensionality, typically by using some form of parallel projection, wherein the point of view is from a fixed perspective, but also reveals multiple facets of an object.

Examples of pseudo-3D techniques include isometric projection, oblique projection, orthographic projection, billboarding, parallax scrolling, scaling, skyboxes, and sky domes.

In addition, 3D graphical techniques such as bump mapping and parallax mapping are often used to extend the illusion of three-dimensionality without substantially increasing the resulting computational overhead introduced by larger numbers of polygons. These terms sometimes possess a second meaning, wherein the gameplay in an otherwise 3D game is forcibly restricted to a two-dimensional plane.

3.3 EXISTING SYSTEM

Ray casting is the methodological basis for 3D CAD/CAM solid modelling and image rendering. It is essentially the same as ray tracing for computer graphics where virtual light rays are "cast" or "traced" on their path from the focal point of a camera through each pixel in the camera sensor to determine what is visible along the ray in the 3D scene.

The term "Ray Casting" was introduced by Scott Roth while at the General Motors Research Labs from 1978–1980. His paper, "Ray Casting for Modelling Solids", describes modelled solid objects by combining primitive solids, such as blocks and cylinders, using the set operator's union (+), intersection (&), and difference (-).

The general idea of using these binary operators for solid modelling is largely due to Voelcker and Requicha's geometric modelling group at the University of Rochester. See solid modelling for a broad overview of solid modelling methods. This figure on the right shows a U-Joint modelled from cylinders and blocks in a binary tree using Roth's ray casting system in 1979.

Before ray casting (and ray tracing), computer graphics algorithms projected surfaces or edges (e.g., lines) from the 3D world to the image plane where visibility logic had to be applied. The world-to-image plane projection is a 3D homogeneous coordinate system transformation (aka: 3D projection, affine transformation, or projective transform). Rendering an image in that way is difficult to achieve with hidden surface/edge removal. Plus, silhouettes of curved surfaces have to be explicitly solved for whereas it is an implicit by-product of ray casting, so there is no need to explicitly solve for it whenever the view changes.

3.3.1 EXISTING SYSTEM ARCHITECTURE

The Existing System Architecture describes the overall processing of how the developer used complex coding and scripts to create a Game engine. This game engine was vastly used in old 3D games which was developed in the late 90s.

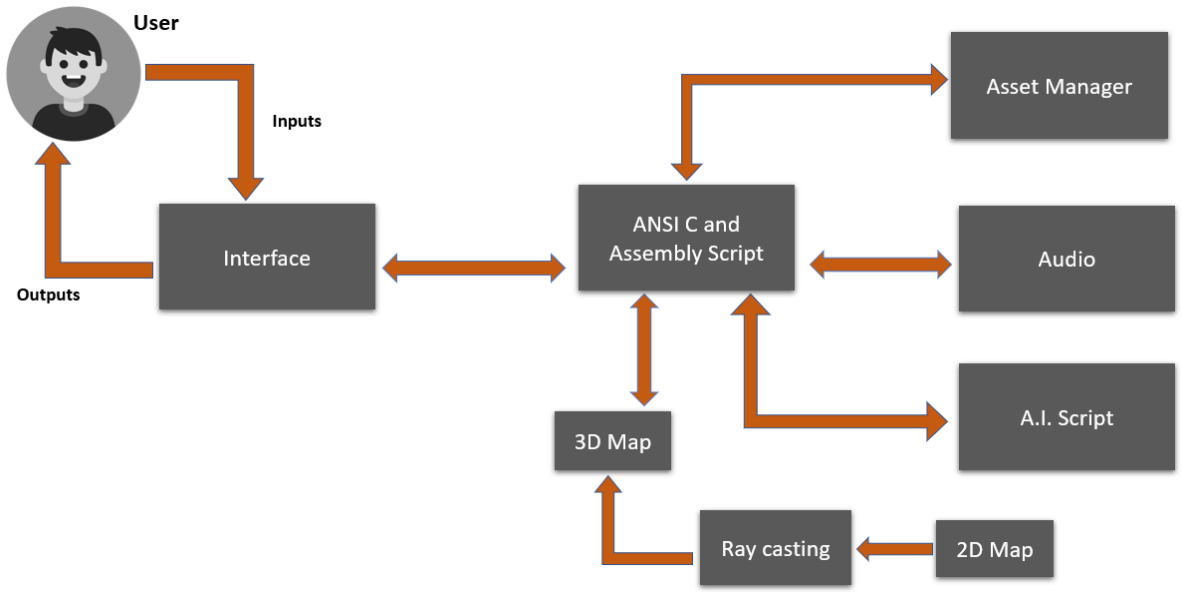

Figure 3.1 Existing system architecture

3.3.2 DISADVANTAGES

- This kind of game engine was developed in the late 90s, thus proving it's not up to trend.
- Game engine is not properly optimized.
- Pixel graphics in the old games are choppy and low quality.
- Many glitches and bugs are discoverable in these old games.

3.4 PROPOSED SYSTEM

We Proposed a system to create and improve the old ray casting game engine which was developed in the late 90s.

In order to frame a proposal to make the game development accessible to the general public, a game development environment is presented based on the definition of its data model, its visual programming elements along with its behaviour rules system, and the design of its editor

This is a 3D shooting game we have developed using Python, game ends after defeating all the enemies present in the map. The A.I. enemies target the player, when they come into their view distance.

3.4.1 PROPOSED SYSTEM ARCHITECTURE

System Architecture describes the overall processing of how the input data is pre-processed and actions are taken within the game. The game ends after defeating all the enemies present on the map or dying.

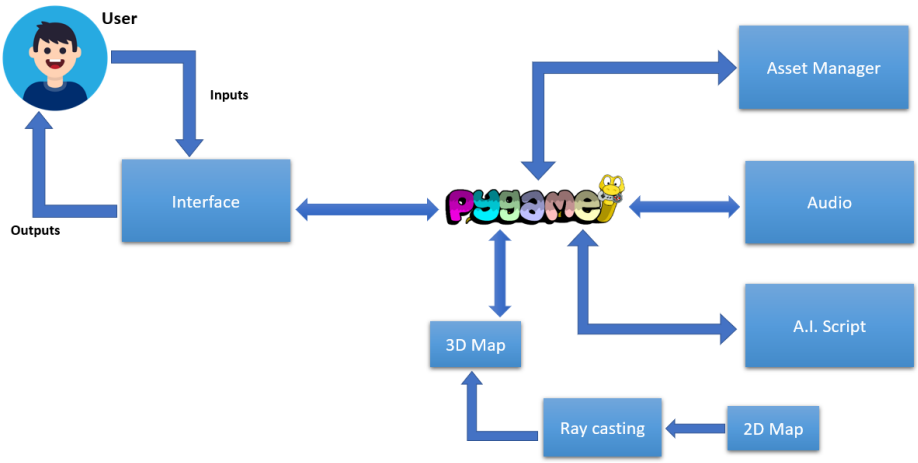

Figure 3.2 Proposed system architecture

The user sent inputs to the code via keyboard or mouse, the inputs are then processed in pygame package. The Pygame package handles almost every process happening in the game, rendering of the images, loading of Audio, managing assets (sprite-sheets), and displaying the output with smoother framerate.

3.4.2 ADVANTAGES

- Numerous amounts of software for image manipulation are available for creating 4K high resolution pixel images.
- With the package available in the coding language python, we are able to create the game without any bugs. If any is found, the bug fixes are easy than older times.
- By recreating the game engine of the old times, we can improve it into a better version with the newly available coding software.
- Realism – hardcore gamers who devote hours to playing every day prefer the more immersive experience offered by 3D games.
- High quality of graphics – some 3D HD games are real works of art where the players can spend hours just exploring the nuances of the world.
- 3D can be used to create games of different genres, and most of those games will give players some freedom of action. Compared to 2D games they have more complicated missions, and just more options in general.

CLASS DIAGRAM FOR PROPOSED SYSTEM

Figure 3.2 depicts the Class Diagram which describes the user's details and service requests as opposed to how the System responds to these requests with the help of the database.

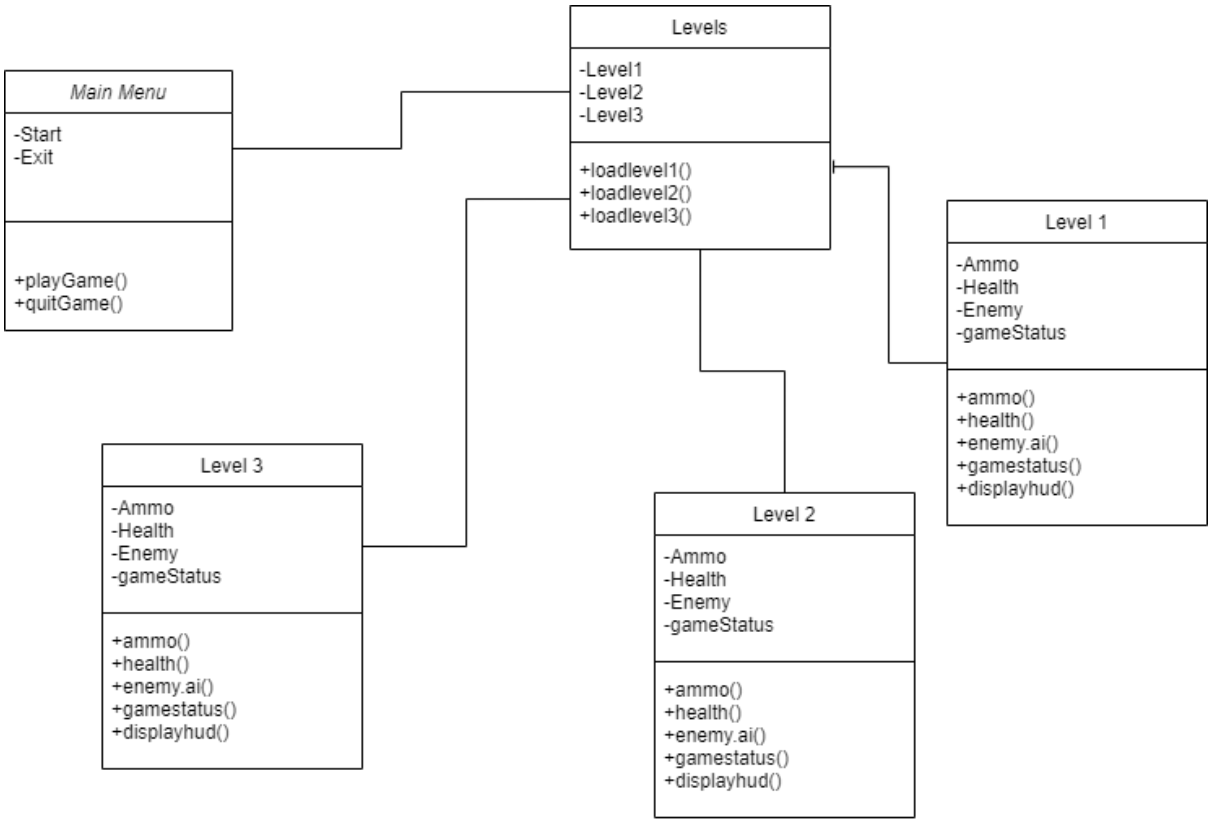

Figure 3.3 Class diagram for Proposed system

USE CASE DIAGRAM FOR PROPOSED SYSTEM

Describes how the user performs various functions from start to finish in the System.

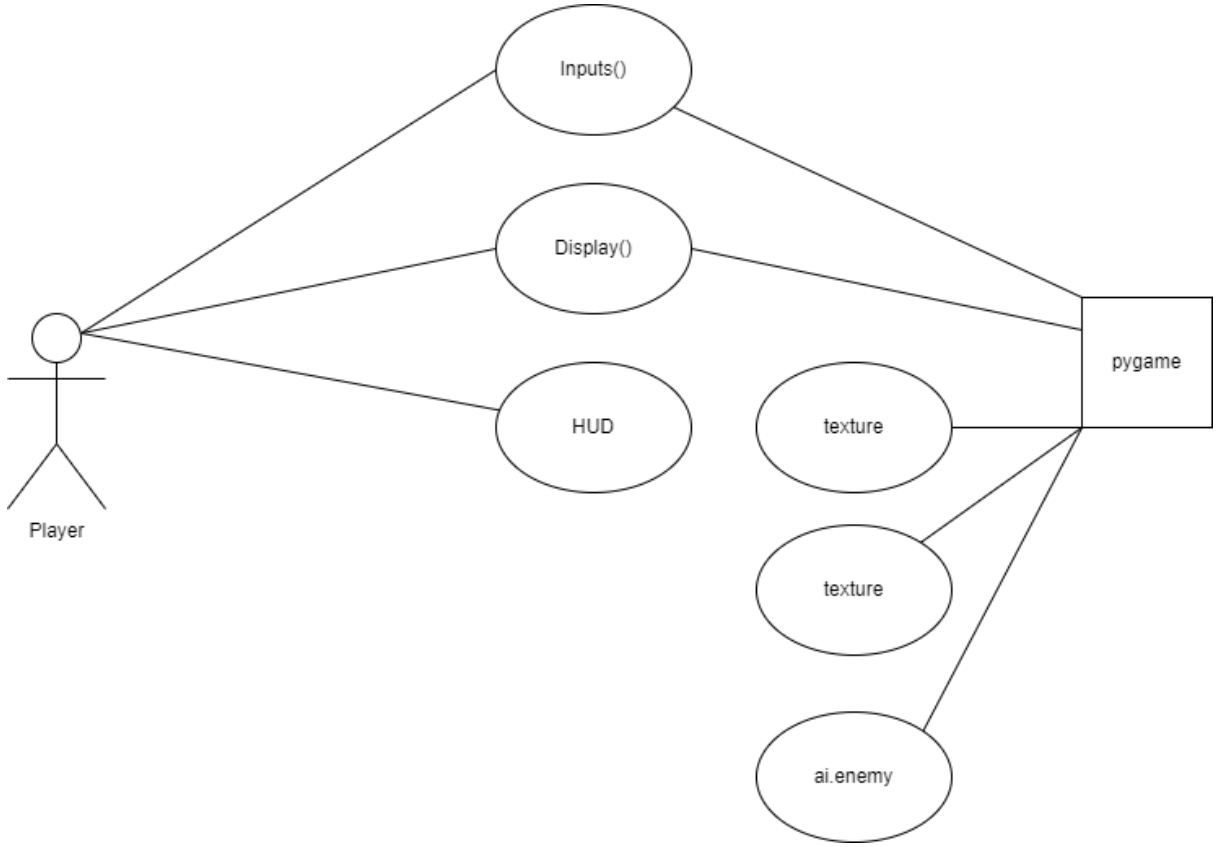

Figure 3.4 Use Case diagram for Proposed system

SEQUENCE DIAGRAM FOR PROPOSED SYSTEM

Figure 3.4 depicts the Sequence Diagram which describes the sequence of events occurring within the System with details as to how the System utilizes the database to deliver the various functionalities to the user.

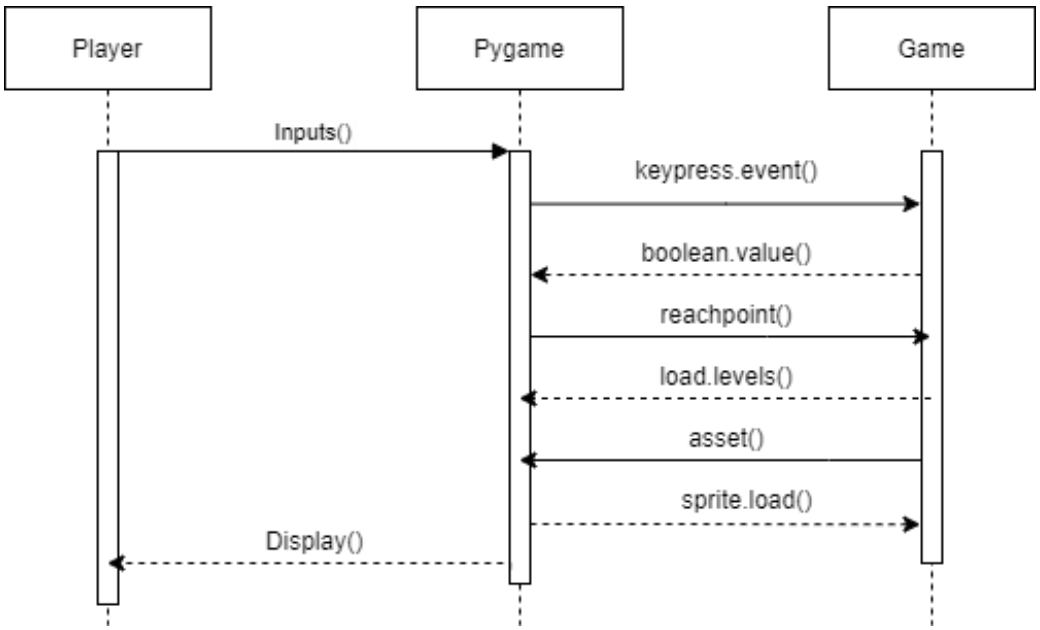

Figure 3.5 Sequence diagram for Proposed system

COLLABRATION DIAGRAM FOR PROPOSED SYSTEM

Figure 3.5 depicts the Collaboration Diagram which describes the basic way in which the user interacts with the System and the System in turn utilizes the database to store and retrieve information.

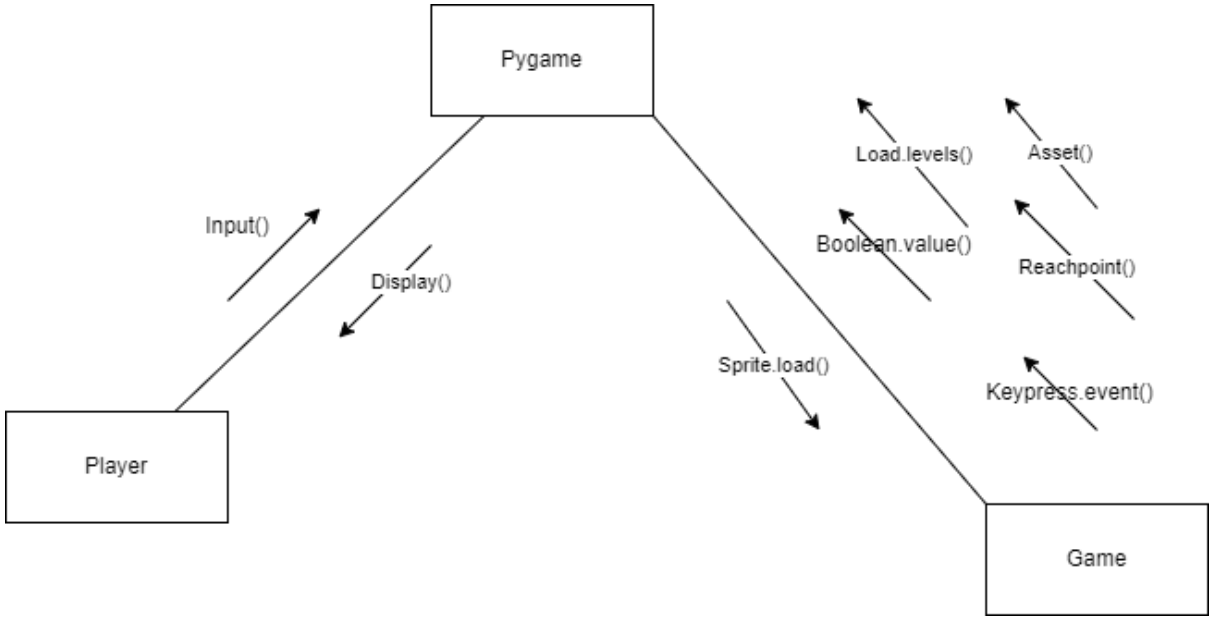

Figure 3.6 Collaboration diagram for Proposed system

3.5 SUMMARY

The UML Diagrams are an ideal representation of the design work that describes the working of the System in a general perspective. In that, the various categories of UML Diagrams describe the System in a unique manner such that all the aspects of the System are described, analysed, rectified and finalized before it is translated in the form of working code. The purpose of representing the requirements in the form of such diagrams is to make sure that all the necessary changes can be made before the implementation process as doing so here will be effective in terms of cost and time.

CHAPTER 4

SYSTEM DESIGN AND IMPLEMENTATION

4.1 GENERAL

System Design describes the overall processing of the entire working of the game, this includes how the A.I would behave when player approaches, and how each interaction will take place with items placed in the level. The python package pygame, does all the work including the entire process of loading and changing values of Boolean variables present in the game. Finally, after all the process the game is executed, with fun interactions and problem-solving situations.

4.2 LIST OF MODUES

1. Implementing cross-platform package “Pygame”
2. Assigning Textures for In-game models
3. Adding key binds for player control
4. Designing Game engine (ray casting)
5. Advanced A.I. implementation

4.3 MODULE DESCRIPTION

The modules are performed using different techniques and they are described below with the help of illustrations and screenshots of the game and the application used to design the game.

4.3.1 IMPLEMENTING CROSS-PLATFORM PACKAGE PYGAME

Pygame is a Python wrapper for the SDL library, which stands for Simple DirectMedia Layer. SDL provides cross-platform access to your system's underlying multimedia hardware components, such as sound, video, mouse, keyboard, and joystick.

Pygame started life as a replacement for the stalled PySDL project. The cross-platform nature of both SDL and pygame means you can write games and rich multimedia Python programs for every platform that supports them.

4.3.1.1 INITIALIZATION AND MODULES

```
$ pip install pygame
```

Figure 4.1 Command for installing pygame

The pygame library is composed of a number of Python constructs, which include several different modules. These modules provide abstract access to specific hardware on your system, as well as uniform methods to work with that hardware. For example, display allows uniform access to your video display, while joystick allows abstract control of your joystick.

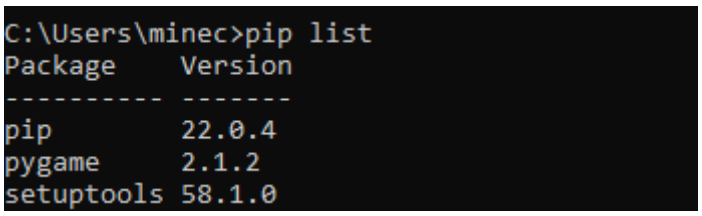

Figure 4.2 The Pygame package present in the system

After importing the pygame library in the example above, the first thing you did was initialize Pygame using pygame.init(). This function calls the separate init() functions of all the included pygame modules.

Since these modules are abstractions for specific hardware, this initialization step is required so that you can work with the same code on Linux, Windows, and Mac.

4.3.1.2 DISPLAYS AND SURFACES

In addition to the modules, pygame also includes several Python classes, which encapsulate non-hardware dependent concepts. One of these is the Surface which, at its most basic, defines a rectangular area on which you can draw. Surface objects are used in many contexts in pygame. Later you'll see how to load an image into a Surface and display it on the screen.

In pygame, everything is viewed on a single user-created display, which can be a window or a full screen. The display is created using set_mode(), which returns a Surface representing the visible part of the window. It is this Surface that you pass into drawing functions like pygame.draw.circle(), and the contents of that Surface are pushed to the display when you call pygame.display.flip().

4.3.1.3 IMAGES AND RECT

Our basic pygame program drew a player model directly onto the display's Surface, but you can also work with images on the disk. The image module allows you to load and save images in a variety of popular formats. Images are loaded into Surface objects, which can then be manipulated and displayed in numerous ways.

As mentioned above, Surface objects are represented by rectangles, as are many other objects in pygame, such as images and windows. Rectangles are so

heavily used that there is a special Rect class just to handle them. You'll be using Rect objects and images in your game to draw players and enemies, and to manage collisions between them.

4.3.2 ASSIGNING TEXTURES FOR IN-GAME MODELS

The benefit of applying a texture or bump / displacement map directly onto a CAD model is that you can utilize existing graphics and images, making dramatic changes to the surface geometry without having to model any complex surfaces by hand.

4.3.2.1 RESOURCE LOADING

The images and sounds objects can be used to load images and sounds from files stored in the images and sounds subdirectories respectively. Pygame Zero will handle loading of these resources on demand and will cache them to avoid reloading them. You generally need to ensure that your images are named with lowercase letters, numbers and underscores only. They also have to start with a letter.

```
alien.png
alien_hurt.png
alien_run_7.png
```

Figure 4.3 Files names that will work

The files which starts with numbers, or have symbols will not work as shown in figure 4.4

```
3.png
3degrees.png
my-cat.png
sam's dog.png
```

Figure 4.4 File names that will not work

4.3.2.2 IMAGES

We here called Images as Sprite-sheets, these sprite sheets can be used for multiple models at the same time, while saving storage space. Some sprites have many frames, in order to load them in quickly enough to make it seem like an animation, “looping” is used.

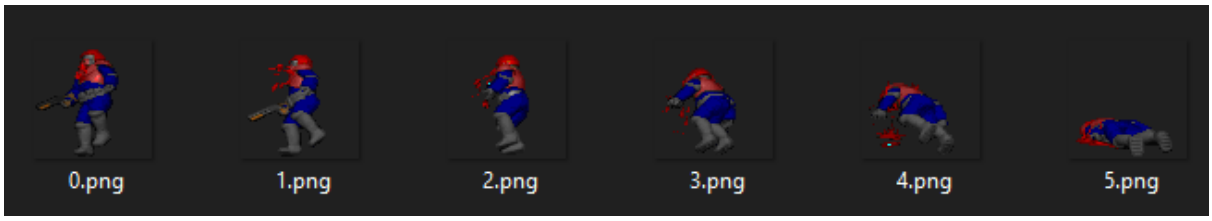

Figure 4.5 Set of Sprite Models

Pygame can Load images in .png, .gif, and .jpg formats. PNG is recommended: it will allow high quality images with transparency. We need to ensure an images directory is set up.

4.3.2.3 IMAGE SURFACES

You can also load images from the images directory using the images object. This allows you to work with the image data itself, query its dimensions and so on

```
self.attack_images = self.get_images(self.path + '/attack')
self.death_images = self.get_images(self.path + '/death')
self.idle_images = self.get_images(self.path + '/idle')
self.pain_images = self.get_images(self.path + '/pain')
self.walk_images = self.get_images(self.path + '/walk')
```

Figure 4.6 Loading image data, for implementing animation

Each loaded image is a Pygame Surface. You will typically use screen.blit(...) to draw this to the screen. It also provides handy methods to query the size of the image in pixels.

4.3.3 ADDING KEYBINDS FOR PLAYER CONTROLS

You'll use Pygame to add keyboard controls so you can direct your character's movement. There are functions in Pygame to add other kinds of controls (such as a mouse or game controller), but since you certainly have a keyboard if you're typing out Python code, that's what this topic covers.

Once you understand keyboard controls, you can explore other options on your own.

4.3.3.1 PLAYER MOVEMENT FUNCTION

To make your sprite move, you must create a property for your sprite that represents movement. When your sprite is not moving, this variable is set to 0. If you are animating your sprite, or should you decide to animate it in the future, you also must track frames so the walk cycle stays on track. The player sprite doesn't need to respond to control all the time because sometimes it isn't being told to move. The code that controls the sprite, therefore, is only one small part of all the things the player sprite can do.

```
def movement(self):
    sin_a = math.sin(self.angle)
    cos_a = math.cos(self.angle)
    dx,dy = 0, 0
    speed = PLAYER_SPEED * self.game.delta_time
    speed_sin = speed * sin_a
    speed_cos = speed * cos_a
```

Figure 4.7 setting variables for moving XY axis

When you want to make an object in Python do something independent of the rest of its code, you place your new code in a function.

To move a sprite in Pygame, you must tell Python to redraw the sprite in its new location—and where that new location is. Since the Player sprite isn't always moving, make these updates a dedicated function within the Player class.

Adding Boolean variable can make the player move left and right as well, this is possible by adding key.events(). These events will check if a specific key is being pressed or not.

```
keys = pg.key.get_pressed()
if keys[pg.K_w]:
    dx += speed_cos
    dy += speed_sin
if keys[pg.K_s]:
    dx += -speed_cos
    dy += -speed_sin
if keys[pg.K_a]:
    dx += speed_sin
    dy += -speed_cos
if keys[pg.K_d]:
    dx += -speed_sin
    dy += speed_cos
```

Figure 4.8 Variable set for moving

A Variable “Speed” is declared to let the model know how quickly it would need to move in that direction. A function named shoot is declared to allow the player to shoot with a key-event.

```
def single_fire_event(self, event):
    if event.type == pg.MOUSEBUTTONDOWN:
        if event.button == 1 and not self.shot and not self.game.weapon.reloading:
            self.game.sound.pistol.play()
            self.shot = True
            self.game.weapon.reloading = True
```

Figure 4.9 Key press events

4.3.4 DESIGNING GAME ENGINE

Ray Casting is a technique to create a 3D projection based on 2D plane. This technique was used for old games when computers didn't have a good performance like today computers. You can find this rendering method in Wolfstein 3D that is considered to be the first 3D game ever. The game DOOM uses a similar technique known as binary space partitioning (BSP).

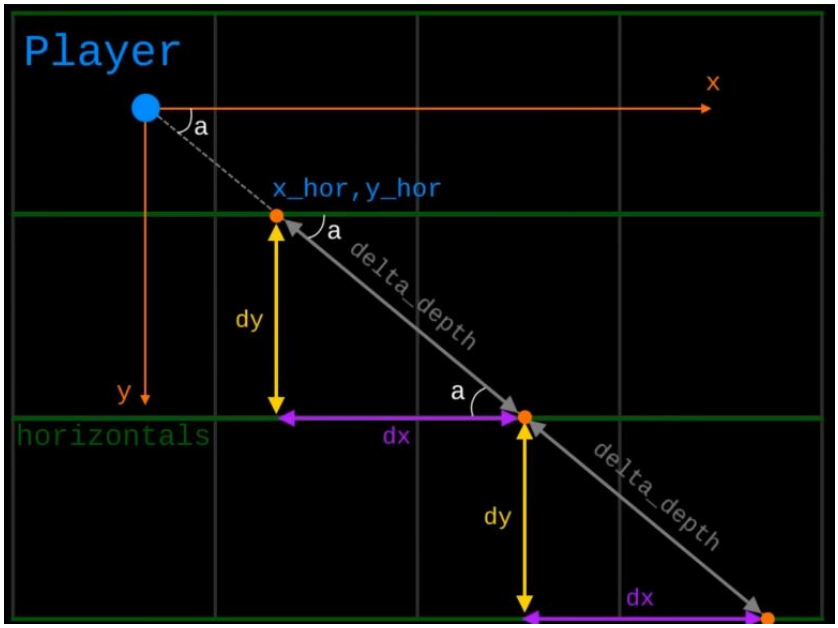

Figure 4.10 Formula used for ray casting

Working of Ray casting is based on Trigonometry formulas. These rays will have a point of intersection between each vertices. These vertices are the different matrix values assigned on the map to act as a wall or floor. Each and every straight line which corresponds to the vertical and horizontal vertices are measured using depth.

As in figure 4.11, the yellow ray coming from the green point (player) is a combination of multiple rays interacting with the world. This interaction is purely used for the next step in Ray casting which is 3D projection.

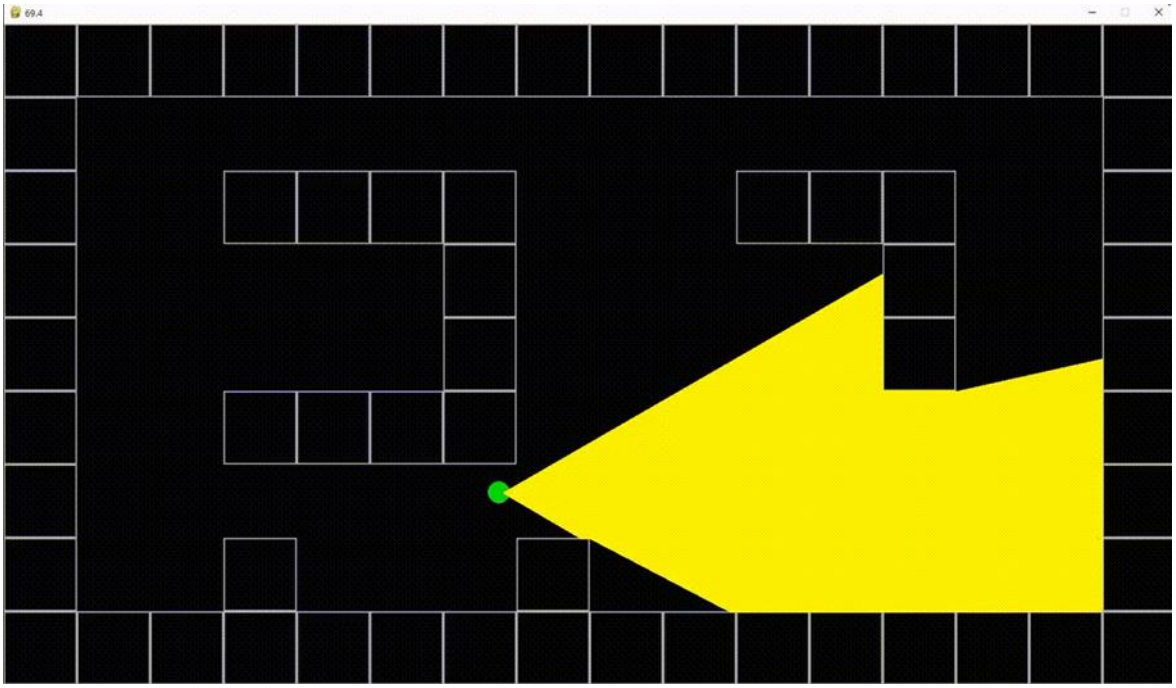

Figure 4.11 ray casting in 2D plane

In 3D projection we use the rays to cast a 3D projection on the screen, thus the name Ray casting. When the rays hit a certain object on the map it creates a point, we use that point to project a 3D projection of the game using texture and that point of contact.

4.3.5 ADVANCED A.I. IMPLEMENTATION

In video games, artificial intelligence (A.I) is used to generate responsive, adaptive or intelligent behaviours primarily in non-player characters (NPCs) similar to human-like intelligence. Artificial intelligence has been an integral part of video games since their inception in the 1950s. Artificial intelligence in video games is a distinct subfield and differs from academic AI. It serves to improve the game-player experience rather than machine learning or decision making.

During the golden age of arcade video games, the idea of AI opponents was largely popularized in the form of graduated difficulty levels, distinct movement patterns, and in-game events dependent on the player's input.

Modern games often implement existing techniques such as pathfinding and decision trees to guide the actions of NPCs. AI is often used in mechanisms which are not immediately visible to the user, such as data mining and procedural-content generation.

```
def run_logic(self):
    if self.alive:
        self.ray_cast_value = self.ray_cast_player_npc()
        self.check_hit_in_npc()

        if self.pain:
            self.animate_pain()

        elif self.ray_cast_value:
            self.player_search_trigger = True

            if self.dist < self.attack_dist:
                self.animate(self.attack_images)
                self.attack()
            else:
                self.animate(self.walk_images)
                self.movement()

        elif self.player_search_trigger:
            self.animate(self.walk_images)
            self.movement()

        else:
            self.animate(self.idle_images)
    else:
        self.animate_death()
```

Figure 4.12 A.I script

A separate script is written for A.I. enemies, this script is activated once the A.I has visual of the player in its line of sight. The sprite is also updated this way to make the enemy look animated. Once the player is in A.I. visual, the A.I will proceed to seek out the player, in addition the A.I. also damages the player at a certain distance, thus resulting in a getting damaged feel. The damage given/taken by the A.I is customizable in the script.

4.4 SUMMARY

These are the various modules needed to design, develop and implement the proposed system which was explained before.

CHAPTER 5

SYSTEM REQUIREMENTS

5.1 GENERAL

This chapter gives an overall description about the functional and non-functional requirements for the system. Functional requirements should include functions performed by specific screens, outlines of work-flows performed by the system, and other business or compliance requirements. Functional requirements of a system can relate to hardware, software or both, in terms of calculations, technical details, data manipulation and processing or other specific functionality that defines what a system is supposed to accomplish. A non-functional requirement is a requirement, which specifies how the system performs a certain function. Non-functional requirements are often called "quality attributes" of a system.

5.2 SYSTEM REQUIREMENTS

5.2.1 HARDWARE REQUIREMENTS

The hardware components are used to develop the systems and to achieve the objectives of the system.

- System : Pentium IV 2.4 GHz.
- Hard Disk : 40 GB.
- Monitor : 15 VGA Colour.
- Ram : 2 GB.
- Video card : GTX 910 or higher

5.2.2 SOFTWARE REQUIREMENTS

The software requirements of the system can also be enlisted in terms of that is used to achieve the objectives of the System and those that will assist the former.

- Operating system : Windows 10
- Programming Language : Python 3.10.4
- Tool : Sublime text

5.3 TECHNICAL SPECIFICATION

5.3.1 SUBLIME

Sublime Text is a shareware text and source code editor available for Windows, macOS, and Linux. It natively supports many programming languages and markup languages. Users can customize it with themes and expand its functionality with plugins, typically community-built and maintained under free-software licenses. To facilitate plugins, Sublime Text features a Python API. The editor utilizes minimal interface and contains features for programmers including configurable syntax highlighting, code folding, search-and-replace supporting regular-expressions, terminal output window, and more. It is proprietary software, but a free evaluation version is available.

5.3.2 PYTHON

Python is commonly used for developing websites and software, task automation, data analysis, and data visualization. Since it's relatively easy to learn, Python has been adopted by many non-programmers such as accountants and scientists, for a variety of everyday tasks, like organizing finances. There are many features in Python, some of which are discussed below as follows:

1. Easy to code: Python is a high-level programming language. Python is very easy to learn the language as compared to other languages like C, C#, Javascript, Java, etc. It is very easy to code in python language and anybody can learn python basics in a few hours or days. It is also a developer-friendly language.

2. Free and Open Source: Python language is freely available at the official website and you can download it from the given download link below click on the Download Python keyword. Download Python Since it is open-source, this means that source code is also available to the public. So you can download it as, use it as well as share it.

3. Object-Oriented Language: One of the key features of python is Object-Oriented programming. Python supports object-oriented language and concepts of classes, objects encapsulation, etc.

4. GUI Programming Support: Graphical User interfaces can be made using a module such as PyQt5, PyQt4, wxPython, or Tk in python. PyQt5 is the most popular option for creating graphical apps with Python.

5. High-Level Language: Python is a high-level language. When we write programs in python, we do not need to remember the system architecture, nor do we need to manage the memory.

6. Extensible feature: Python is a Extensible language. We can write some Python code into C or C++ language and also we can compile that code in C/C++ language.

7. Python is Portable language: Python language is also a portable language. For example, if we have python code for windows and if we want to run this code on other platforms such as Linux, Unix, and Mac then we do not need to change it, we can run this code on any platform.

8. Python is Integrated language: Python is also an Integrated language because we can easily integrated python with other languages like c, c++, etc.

9. Interpreted Language: Python is an Interpreted Language because Python code is executed line by line at a time. like other languages C, C++, Java, etc. there is no need to compile python code this makes it easier to debug our code. The source code of python is converted into an immediate form called bytecode.

10. Large Standard Library: Python has a large standard library that provides a rich set of modules and functions so you do not have to write your own code for every single thing. There are many libraries present in python for such as regular expressions, unit-testing, web browsers, etc.

11. Dynamically Typed Language: Python is a dynamically-typed language. That means the type (for example- int, double, long, etc.) for a variable is decided at run time not in advance because of this feature we don't need to specify the type of variable.

12. Frontend and backend development: With a new project pyscript you can run and write python codes in html with the help of some simple tags <py-script>, <py-env>, etc. This will help you do frontend development work in python like javascript. Backend is the strong forte of python it's extensively used for this work cause of its framework like django and flask.

5.4 SUMMARY

PyCharm is a dedicated Python Integrated Development Environment (IDE) providing a wide range of essential tools for Python developers, tightly integrated to create a convenient environment for productive Python, web, and data science development.

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENTS

The purpose of our project is to develop a improved version of the old game engine with ray casting concept. While making a 3D shooter game which the user can enjoy and have fun solving problems and performing tasks, while enjoying the beauty of the game texture.

In future, we plan on adding more additional features such as a Scoreboard, Pause menu and a Boss health bar.

APPENDIX 1

SCREENSHOTS

Game Starting Area

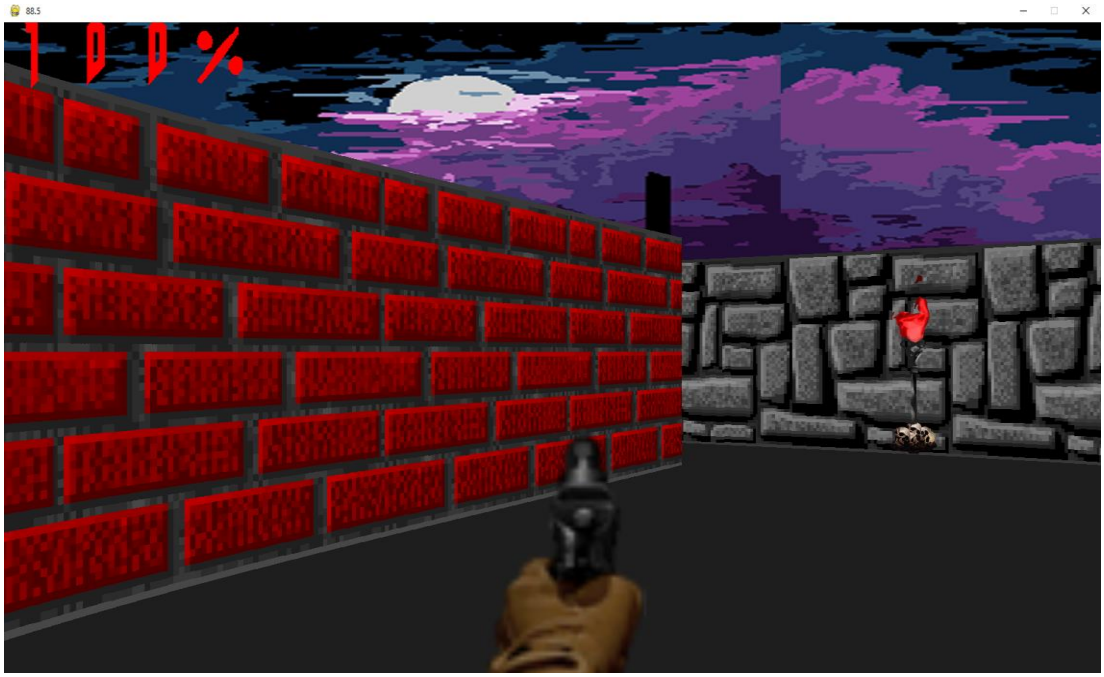

Figure A.1 Starting Area

Player Health Bar

Figure A.2 Health Bar

Enemy's Aggression Towards Player

Figure A.3 A.I. Detection

Game Over Screen

Figure A.4 Game Over Screen

APPENDIX 2

SAMPLE CODE

```
import pygame as pg
import sys
from settings import *
from map import *
from player import *
from raycasting import *
from object_renderer import *
from sprite_object import *
from object_handler import *
from weapon import *
from sound import *
from pathfinding import *
class Game:
    def __init__(self):
        pg.init()
        pg.mouse.set_visible(False)
        self.screen = pg.display.set_mode(RES)
        self.clock = pg.time.Clock()
```

```
        self.delta_time = 1
        self.global_trigger = False
        self.global_event = pg.USEREVENT + 0
        pg.time.set_timer(self.global_event, 40)
        self.new_game()
    def new_game(self):
        self.map = Map(self)
        self.player = Player(self)
        self.object_renderer = ObjectRenderer(self)
        self.raycasting = RayCasting(self)
        self.object_handler = ObjectHandler(self)
        self.weapon = Weapon(self)
        self.sound = Sound(self)
        self.pathfinding = PathFinding(self)
    def update(self):
        self.player.update()
        self.raycasting.update()
        self.object_handler.update()
        self.weapon.update()
        pg.display.flip()
```

```
            self.delta_time = self.clock.tick(FPS)
            pg.display.set_caption(f'{self.clock.get_fps() : .1f}')

    def draw(self):
            #self.screen.fill('black')       # turn on for 2d top down format
            self.object_renderer.draw()
            self.weapon.draw()
            #self.map.draw()                 # turn on for 2d top down format
            #self.player.draw()              # turn on for 2d top down format
    def check_events(self):
            self.global_trigger = False
            for event in pg.event.get():
                    if event.type == pg.QUIT or (event.type == pg.KEYDOWN
and event.key == pg.K_ESCAPE):
                            pg.quit()
                            sys.exit()
                    elif event.type == self.global_event:
                            self.global_trigger = True
                    self.player.single_fire_event(event)
    def run(self):
```

```
            while True:
                self.check_events()
                self.update()
                self.draw()
```

RAYCASTING.PY

```
import pygame as pg
import math
from settings import *
class RayCasting:
    def __init__(self, game):
        self.game = game
        self.ray_casting_result = []
        self.objects_to_render = []
        self.textures = self.game.object_renderer.wall_textures
    def get_objects_to_render(self):
        self.objects_to_render = []
        for ray, values in enumerate(self.ray_casting_result):
            depth, proj_height, texture, offset = values
            if proj_height < HEIGHT:
                wall_column = self.textures[texture].subsurface(
```

```
                                offset * (TEXTURE_SIZE - SCALE), 0, SCALE,
TEXTURE_SIZE
                            )
                            wall_column = pg.transform.scale(wall_column,
(SCALE, proj_height))
                            wall_pos = (ray * SCALE, HALF_HEIGHT -
proj_height // 2)
                        else:
                            texture_height = TEXTURE_SIZE * HEIGHT /
proj_height
                            wall_column = self.textures[texture].subsurface(
                                offset * (TEXTURE_SIZE - SCALE),
HALF_TEXTURE_SIZE - texture_height //2,
                                SCALE, texture_height
                            )
                            wall_column = pg.transform.scale(wall_column,
(SCALE, HEIGHT))
                            wall_pos = (ray * SCALE, 0)
                        self.objects_to_render.append((depth, wall_column, wall_pos))
    def ray_cast(self):
        self.ray_casting_result = []
        ox, oy = self.game.player.pos
```

```
x_map, y_map = self.game.player.map_pos
texture_vert, texture_hor = 1, 1
ray_angle = self.game.player.angle - HALF_FOV + 0.0001
for ray in range(NUM_RAYS):
        sin_a = math.sin(ray_angle)
        cos_a = math.cos(ray_angle)
        #horizontals
        y_hor, dy = (y_map + 1, 1) if sin_a >0 else (y_map - 1e-6, -1)
        depth_hor = (y_hor -oy) / sin_a
        x_hor = ox + depth_hor * cos_a
        delta_depth = dy / sin_a
        dx = delta_depth * cos_a
        for i in range (MAX_DEPTH):
                tile_hor = int(x_hor), int(y_hor)
                if tile_hor in self.game.map.world_map:
                        texture_hor =.game.map.world_map[tile_hor]
                        break
                x_hor += dx
                y_hor += dy
                depth_hor += delta_depth
```

```
#verticals
x_vert, dx = (x_map + 1, 1) if cos_a > 0 else (x_map - 1e-6, -1)
depth_vert = (x_vert - ox) / cos_a
y_vert = oy + depth_vert * sin_a
delta_depth = dx / cos_a
dy = delta_depth * sin_a
for i in range(MAX_DEPTH):
        tile_vert = int(x_vert), int(y_vert)
        if tile_vert in self.game.map.world_map:
                texture_vert = game.map.world_map[tile_vert]
                break
        x_vert += dx
        y_vert += dy
        depth_vert += delta_depth
#depth, texture offset
if depth_vert < depth_hor:
        depth, texture = depth_vert, texture_vert
        y_vert %= 1
        offset = y_vert if cos_a > 0 else (1 - y_vert)
else:
```

```
                    depth, texture = depth_hor, texture_hor
                    x_hor %= 1
                    offset = (1 - x_hor) if sin_a > 0 else x_hor
                #removing fishbowl effect
                depth *= math.cos(self.game.player.angle - ray_angle)
                # projection
                proj_height = SCREEN_DIST / (depth + 0.0001)

# ray casting result
            self.ray_casting_result.append((depth, proj_height, texture, offset))
                ray_angle += DELTA_ANGLE
        def update(self):
            self.ray_cast()
            self.get_objects_to_render()
```

PLAYER.PY

```
from settings import *
import pygame as pg
import math
class Player:
    def __init__(self,game):
```

```
        self.game = game
        self.x, self.y = PLAYER_POS
        self.angle = PLAYER_ANGLE
        self.shot = False
        self.health = PLAYER_MAX_HEALTH
        self.rel = 0
        self.health_recovery_delay = 700
        self.time_prev = pg.time.get_ticks()
    def recover_health(self):
        if self.check_health_recovery_delay() and self.health <
PLAYER_MAX_HEALTH:
            self.health += 1
    def check_health_recovery_delay(self):
        time_now = pg.time.get_ticks()
        if time_now - self.time_prev > self.health_recovery_delay:
            self.time_prev = time_now
            return True
    def check_game_over(self):
        if self.health < 1:
            self.game.object_renderer.game_over()
```

```
                pg.display.flip()
                pg.time.delay(1500)
                self.game.new_game()

    def get_damage(self, damage):
        self.health -= damage
        self.game.object_renderer.player_damage()
        self.game.sound.player_pain.play()
        self.check_game_over()

    def single_fire_event(self, event):
        if event.type == pg.MOUSEBUTTONDOWN:
            if event.button == 1 and not self.shot and not
self.game.weapon.reloading:
                self.game.sound.pistol.play()
                self.shot = True
                self.game.weapon.reloading = True

    def movement(self):
        sin_a = math.sin(self.angle)
        cos_a = math.cos(self.angle)
        dx,dy = 0, 0
        speed = PLAYER_SPEED * self.game.delta_time
```

```
speed_sin = speed * sin_a
speed_cos = speed * cos_a
keys = pg.key.get_pressed()
if keys[pg.K_w]:
        dx += speed_cos
        dy += speed_sin
if keys[pg.K_s]:
        dx += -speed_cos
        dy += -speed_sin
if keys[pg.K_a]:
        dx += speed_sin
        dy += -speed_cos
if keys[pg.K_d]:
        dx += -speed_sin
        dy += speed_cos
self.check_wall_collision(dx, dy)
#if keys[pg.K_LEFT]:
#       self.angle -= PLAYER_ROT_SPEED * self.game.delta_time
#if keys[pg.K_RIGHT]:
#       self.angle += PLAYER_ROT_SPEED * self.game.delta_time
```

```
            self.angle %= math.tau
        def check_wall(self, x, y):
            return (x,y) not in self.game.map.world_map
        def check_wall_collision(self, dx, dy):
            scale = PLAYER_SIZE_SCALE / self.game.delta_time
            if self.check_wall(int(self.x + dx * scale), int(self.y)):
                self.x += dx
            if self.check_wall(int(self.x), int(self.y + dy * scale)):
                self.y += dy
        def draw(self):
            pg.draw.line(self.game.screen, 'yellow', (self.x * 100, self.y * 100),
                                    (self.x * 100 + WIDTH * math.cos(self.angle),
                                self.y * 100 + WIDTH * math.sin(self.angle)),2)
        pg.draw.circle(self.game.screen, 'green', (self.x * 100, self.y * 100),15)
        def mouse_control(self):
            mx, my = pg.mouse.get_pos()
            if mx < MOUSE_BORDER_LEFT or mx >
MOUSE_BORDER_RIGHT:
                pg.mouse.set_pos([HALF_WIDTH, HALF_HEIGHT])
            self.rel = pg.mouse.get_rel()[0]
```

```
            self.rel = max(-MOUSE_MAX_REL, min(MOUSE_MAX_REL,
self.rel))
            self.angle += self.rel * MOUSE_SENSITIVITY *
self.game.delta_time
    def update(self):
            self.movement()
            self.mouse_control()
            self.recover_health()
```

WEAPON.PY

```
from sprite_object import *
class Weapon(AnimatedSprite):
    def __init__(self, game, path='resources/sprites/weapon/1.png', scale=6,
animation_time=90):
            super().__init__(game=game, path=path, scale=scale,
animation_time=animation_time)
            self.images = deque(
                [pg.transform.smoothscale(img,(self.image.get_width() * scale,
self.image.get_height() * scale))
                 for img in self.images])
            self.weapon_pos = (HALF_WIDTH - self.images[0].get_width() // 2,
HEIGHT - self.images[0].get_height())
            self.reloading = False
```

```
            self.num_images = len(self.images)
            self.frame_counter = 0
            self.damage = 50
        def animate_shot(self):
            if self.reloading:
                self.game.player.shot = False
                if self.animation_trigger:
                    self.images.rotate(-1)
                    self.image = self.images[0]
                    self.frame_counter += 1
                    if self.frame_counter == self.num_images:
                        self.reloading = False
                        self.frame_counter = 0
        def draw(self):
            self.game.screen.blit(self.images[0], self.weapon_pos)


def update(self):
            self.check_animation_time()
            self.animate_shot()
```

ENEMY.PY

```
from sprite_object import *
from random import randint, random, choice
class NPC(AnimatedSprite):
    def __init__(self, game, path='resources/sprites/npc/soldier/0.png',
pos=(12.5, 5.5),
                 scale=1, shift=0.38, animation_time=180):
        super().__init__(game, path, pos, scale, shift, animation_time)
        self.attack_images = self.get_images(self.path + '/attack')
        self.death_images = self.get_images(self.path + '/death')
        self.idle_images = self.get_images(self.path + '/idle')
        self.pain_images = self.get_images(self.path + '/pain')
        self.walk_images = self.get_images(self.path + '/walk')
        self.attack_dist = randint(3, 6)
        self.speed = 0.03
        self.size = 7
        self.health = 100
        self.attack_damage = 15
        self.accuracy = 0.15
        self.alive = True
        self.pain = False
```

```
            self.ray_cast_value = False
            self.frame_counter = 0
            self.player_search_trigger = False
        def update(self):
            self.check_animation_time()
            self.get_sprite()
            self.run_logic()
            #self.draw_ray_cast()                    #this is used to view
enemy in 2D top-down format
        def check_wall(self, x, y):
            return (x,y) not in self.game.map.world_map
        def check_wall_collision(self, dx, dy):
            if self.check_wall(int(self.x + dx * self.size), int(self.y)):
                self.x += dx
            if self.check_wall(int(self.x), int(self.y + dy * self.size)):
                self.y += dy
        def movement(self):
            next_pos = self.game.pathfinding.get_path(self.map_pos,
self.game.player.map_pos)
            next_x, next_y = next_pos
```

```
            if next_pos not in self.game.object_handler.npc_position:
                angle = math.atan2(next_y + 0.5 - self.y, next_x + 0.5 - self.x)
                dx = math.cos(angle) * self.speed
                dy = math.sin(angle) * self.speed
                self.check_wall_collision(dx, dy)
    def attack(self):
        if self.animation_trigger:
            self.game.sound.npc_shot.play()
            if random() < self.accuracy:
                self.game.player.get_damage(self.attack_damage)
    def animate_death(self):
        if not self.alive:
            if self.game.global_trigger and self.frame_counter <
len(self.death_images) - 1:
                self.death_images.rotate(-1)
                self.image = self.death_images[0]
                self.frame_counter += 1
    def animate_pain(self):
        self.animate(self.pain_images)
        if self.animation_trigger:
```

```
                    self.pain = False
        def check_hit_in_npc(self):
            if self.ray_cast_value and self.game.player.shot:
                if HALF_WIDTH - self.sprite_half_width < self.screen_x <
HALF_WIDTH + self.sprite_half_width:
                    self.game.sound.npc_pain.play()
                    self.game.player.shot = False
                    self.pain = True
                    self.health -= self.game.weapon.damage
                    self.check_health()

        def check_health(self):
            if self.health < 1:
                self.alive = False
                self.game.sound.npc_death.play()
        def run_logic(self):
            if self.alive:
                self.ray_cast_value = self.ray_cast_player_npc()
                self.check_hit_in_npc()
                if self.pain:
```

```
                self.animate_pain()

            elif self.ray_cast_value:
                self.player_search_trigger = True

                if self.dist < self.attack_dist:
                    self.animate(self.attack_images)
                    self.attack()
                else:
                    self.animate(self.walk_images)
                    self.movement()

            elif self.player_search_trigger:
                self.animate(self.walk_images)
                self.movement()
            else:
                self.animate(self.idle_images)
        else:
            self.animate_death()
    @property
```

```
def map_pos(self):
        return int(self.x), int(self.y)
def ray_cast_player_npc(self):
        if self.game.player.map_pos == self.map_pos:
                return True
        wall_dist_v, wall_dist_h = 0, 0
        player_dist_v, player_dist_h = 0, 0
        ox, oy = self.game.player.pos
        x_map, y_map = self.game.player.map_pos
        texture_vert, texture_hor = 1, 1
        ray_angle = self.theta
        sin_a = math.sin(ray_angle)
        cos_a = math.cos(ray_angle)
        #horizontals
        y_hor, dy = (y_map + 1, 1) if sin_a >0 else (y_map - 1e-6, -1)
        depth_hor = (y_hor -oy) / sin_a
        x_hor = ox + depth_hor * cos_a
        delta_depth = dy / sin_a
        dx = delta_depth * cos_a
        for i in range (MAX_DEPTH):
```

```
            tile_hor = int(x_hor), int(y_hor)
            if tile_hor == self.map_pos:
                player_dist_h = depth_hor
                break
            if tile_hor in self.game.map.world_map:
                wall_dist_h = depth_hor
                break
            x_hor += dx
            y_hor += dy
            depth_hor += delta_depth
        #verticals
        x_vert, dx = (x_map + 1, 1) if cos_a > 0 else (x_map - 1e-6, -1)
        depth_vert = (x_vert - ox) / cos_a
        y_vert = oy + depth_vert * sin_a
        delta_depth = dx / cos_a
        dy = delta_depth * sin_a
        for i in range(MAX_DEPTH):
            tile_vert = int(x_vert), int(y_vert)
            if tile_vert == self.map_pos:
                player_dist_v = depth_vert
```

```
                    break
                if tile_vert in self.game.map.world_map:
                    wall_dist_v = depth_vert
                    break
                x_vert += dx
                y_vert += dy
                depth_vert += delta_depth
            player_dist = max(player_dist_v, player_dist_h)
            wall_dist = max(wall_dist_v, wall_dist_h)
            if 0 < player_dist < wall_dist or not wall_dist:
                return True
            return False
        def draw_ray_cast(self):
            pg.draw.circle(self.game.screen, 'red', (100 * self.x, 100 * self.y), 15)
            if self.ray_cast_player_npc():
                pg.draw.line(self.game.screen, 'orange', (100 *
self.game.player.x, 100 * self.game.player.y),
                                                    (100 * self.x, 100 * self.y), 2)
```

PATHFINDING.PY

```
from collections import deque
```

```
class PathFinding:
    def __init__(self, game):
        self.game = game
        self.map = game.map.mini_map
        self.ways = [-1, 0], [0, -1], [1, 0], [0, 1], [-1, -1], [1, -1], [1, 1], [-1, 1]
        self.graph = {}
        self.get_graph()
    def get_path(self, start, goal):
        self.visited = self.bfs(start, goal, self.graph)
        path = [goal]
        step = self.visited.get(goal, start)
        while step and step != start:
            path.append(step)
            step = self.visited[step]
        return path[-1]
    def bfs(self, start, goal, graph):
        queue = deque([start])
        visited = {start: None}
        while queue:
            cur_node = queue.popleft()
```

```
                if cur_node == goal:
                    break
                next_nodes = graph[cur_node]
                for next_node in next_nodes:
                    if next_node not in visited and next_node not in
self.game.object_handler.npc_position:
                        queue.append(next_node)
                        visited[next_node] = cur_node
        return visited
    def get_next_nodes(self, x, y):
        return [(x + dx, y + dy) for dx, dy in self.ways if (x + dx, y + dy) not
in self.game.map.world_map]
    def get_graph(self):
        for y, row in enumerate(self.map):
            for x, col in enumerate(row):
                if not col:
                    self.graph[(x,y)] = self.graph.get((x, y), []) +
self.get_next_nodes(x, y)
```

SOUND.PY

```
import pygame as pg
class Sound:
```

```
def __init__(self, game):
        self.game = game
        pg.mixer.init()
        self.path = 'resources/sound/'
        self.theme = pg.mixer.music.load(self.path + 'theme.mp3')
        self.npc_pain = pg.mixer.Sound(self.path + 'npc_pain.mp3')
        self.npc_death = pg.mixer.Sound(self.path + 'npc_death.mp3')
        self.npc_shot = pg.mixer.Sound(self.path + 'pistol.mp3')
        self.player_pain = pg.mixer.Sound(self.path + 'player_pain.mp3')
        self.pistol = pg.mixer.Sound(self.path + 'pistol.mp3')
```

REFERENCES

[1] Brent Cowan and Bill Kapralos, "A Survey of Frameworks and Game Engines for Serious Game Development", 2014 IEEE 14th International Conference on Advanced Learning Technologies.

[2] Georgios N. Yannakakis and J. Togelius, Artificial Intelligence and Games, New York, NY, USA:Springer-Verlag, 2018.

[3] Jonas Siim Andersen and Riccardo Miccini, “Evaluation of Individualized HRTFs in a 3D Shooter Game”, Immersive and 3D Audio: from Architectrue to Automotive (I3DA), 2021

[4] Matthias Labschütz et al., "Content creation for a 3D game with Maya and Unity 3D", Institute of Computer Graphics and Algorithms Vienna University of Technology, 2011.

[5] Meng Yang and Zhen Wang, “Research on of 3D game design and development technology”, 3rd Internataional Conference on Computer Science and Information Technology, 2010.

[6] Pa. Megha, L. Nachammai and T.M. Senthil Ganesan, "3D Game Development Using Unity Game Engine", SSRG International Journal of Computer Science and Engineering, 2018.

[7] Victor Travassos Sarinho and Antonio Lopes Apolinario, “A Generative Programming Approach for Game Development”, VII Brazilian Symposium on Games and Digital Entertainment, 2009

[8] Voravika Wattanasoontorn and Mathus Theppaitoon, “A Classification of Visual Style for 3D Games”, 23rd International Computer Science and Engineering Conference, 2019.