International Journal of Engineering Research and Technology. ISSN 0974-3154, Volume 13, Number 1 (2020), pp. 58-73

Design and Implementation of an Intelligent Gaming Agent Using a* Algorithm and Finite State Machines

Adekanmi Adeyinka Adegun[1], Roseline Oluwaseun Ogundokun[2, *], Samuel Ogbonyomi[3]

[1, 2, 3]Department of Computer Science, Landmark University Omu Aran, Kwara State
[*]Corresponding Author's Detail: Department of Computer Science, Landmark University Omu Aran, Kwara State,
E-mail: ogundokun.roseline@lmu.edu.ng

ORCID: 0000-0002-2592-2824 (Ogundokun Roseline)

Abstract

The last decade has seen Artificial Intelligence (AI) seep into the game development industry, mainly for the purpose of developing more human-like non-player characters (NPCs) to improve player experience. This study focuses on solving the problem of player experience with respect to AI controlled gaming agents. The main aim of the study is to develop an intelligent gaming agent by implementing A* Pathfinding and Finite State Machines. For the implementation, Mixamo/Adobe Fuse was used to model the 3D characters (PC and NPCs), the A* Pathfinding component was implemented using a Unity 3D plugin known as "Arongranberg's A* Pathfinding Project". The FSM component on the other hand was implemented via C# scripts. Unity IDE was used to compile scripts and simulate the game. The end product of this project is a 3rd Person survival shooter 3D game which fully implements the concepts of A* Pathfinding and Finite State Machines. Potentially, this project could be used as a guide to developing games of the same or different genre. With little improvements, the end product of this project (the game) could be made retail ready.

Keywords: A*, Pathfinding, Artificial Intelligent, Game Agent, Finite State Machine, 3D, Mixamo/Adobe Fuse, C#, Unity IDE, FSM

INTRODUCTION

Game evolution/design is the procedure of producing a video game. Game evolution has evolved in recent years. No longer does making a game involve writing simple lines of code by an average programmer. Now it requires a team of specialists in disciplines such as Fine art, Graphics Modelling and Design, Software Programming, Music, Network Programming, Artificial Intelligence Programming and so on. Games have protracted happened to be an accepted field of Artificial Intelligence (AI) research, which is for a respectable motive. They are problematic nevertheless stress-free to validate, hence making it feasible to establish novel AI techniques, compute in what way they are functioning, and displays that machineries are qualified of extraordinary conduct usually alleged to necessitate cleverness exclusive of placing person breathes or property at jeopardy (*Miikkulainen et al*, 2006).

Artificial Intelligence (AI) is the field within Computer Science that seeks to explain and to emulate some or all aspects of human intelligence through mechanical or computational processes. Included among these aspects of intelligence are the ability to interact with the environment through sensory means and the ability to make decisions in unforeseen circumstances without human intervention. Typical areas of research in AI include game playing, natural language understanding and synthesis, computer vision, problem solving, learning, and robotics (Restu, 2015). Over the years, there has been an increase in the need for Artificial Intelligence in Game Development. Implementing AI in a game will give the users the illusion that they are playing intelligent agents. From the definition of Intelligent Agents in Artificial Intelligence, we can say an intelligent agent is anything that can perceive/observing its immediate environment and take action with respect to its observation, hence we can say an intelligent gaming agent is capable of learning/observing what goes on in the gaming environment and also act on its observation.

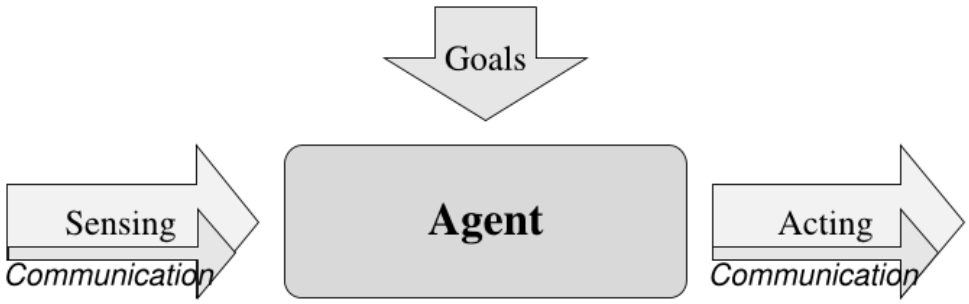
This research study focuses majorly on how Artificial Intelligence is implemented in Game Development, taking into consideration the A* Path-finding Algorithm and Finite State Machines and it was implemented in C#.

LITERATURE REVIEW

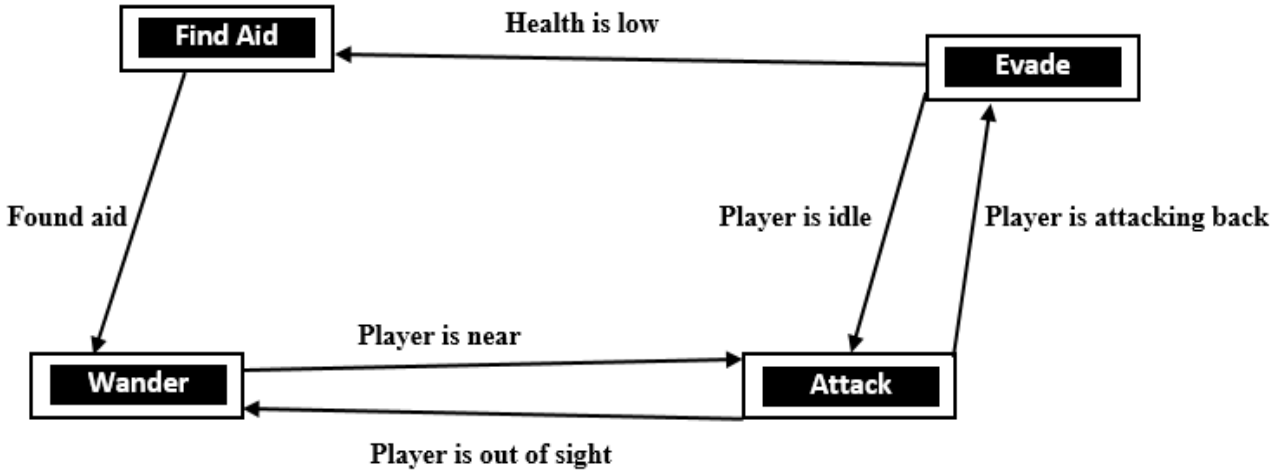
INTELLIGENT GAME AGENTS

In most games, the purpose of AI is to create an intelligent agent sometimes referred to as Non-player Character (NPC). This agent may act as an opponent, an ally, or as a neutral entity in the game world. Since the majority of game AI focuses around the agent, it is very helpful to study game AI from this perspective.

An agent has three key steps through which it continually loops. The steps are commonly known as the sense-think-act cycle. In addition to these three steps, there is an optional learning or remembering step that may also take place during this loop. In practice, most game agents do not take this extra step, but this is slowly changing because of the added challenge and playability that is leveraged as a result. The game agent must have information about the current state of the world to make good decisions and to act on those decisions. Since the game world is represented entirely inside the program, perfect information about the state of the world is always available



Agent interaction with the Environment (Nareyek, 2002).



FSM representing the brain of a game agent (Bevilaqua, 2013).

A* PATHFINDING

Pathfinding normally means finding the shortest path amid two termination outlets. Early clarifications to the problematic pathfinding in computer games, such as depth first search, iterative deepening, breadth first search, Dijkstra's algorithm, best first search, A* algorithm, and iterative deepening A*, remained overwhelmed by the sheer exponential advance in the intricacy of the game. Extra competent results are necessitated so as to be competent to elucidate pathfinding difficulties on an additional complex setting with restricted time and resources (Cui, and Shi, 2011).

A* path-finding is referred to as a system applied for discovery of inexpensive route through a location. Specifically, it is a search algorithm that is directed to exploit knowledge about a destination to guide the search intelligently and engaging in this minimized the process requires to find a solution. A* is regarded as the fastest algorithm used in finding the absolute cheapest path when compared to other search algorithms (Rabin, 2009).

A* can also be regarded as a generic search algorithm, which can be utilized to discover solutions for numerous glitches and pathfinding just one of them. For pathfinding, A* algorithm continually assesses the greatest promising unfamiliar position it has realized. After a position is sightseen, the algorithm is completed if that position foreseen is the goal; else, it makes note of all that position's neighbors for advance consideration (Cui, and Shi, 2011).

A* Pseudo-code:

1. Create the rootNode.
   - Set its x and y according to the startPoint
   - Set its parent to NULL
   - Set its finalCost to givenCost + heuristicCost
2. Push the rootNode onto the open list
3. While the open list is not empty

A) Pop the node with the lowest finalCost from open and assign it to currentNode

B) If currentNode's x and y correspond to the goalPoint then

-Break from step 3

C) For every nearbyPoint around the currentNode

a) If this nearbyPoint is in a spot that is impassable then

- Skip to the next nearbyPoint

b) Create the successorNode

- Set its x and y according to nearbyPoint

- Set its parent to currentNode

- set its finalCost to givenCost + heuristicCost

c) If a node for this nearbyPoint has been created before then

- if successorNode is better than oldNode then

    - pop the oldNode and delete it

- else

    - skip to next nearbyPoint

d) Push the successorNode onto the open list

D) Push the currentNode onto the closed list

4. If the while loop exists without finding the goal, goalPoint must be unreachable (Rabin, 2009).

THE GAME

The game in which the AI techniques will be implemented will be of the following genre;

1. Orthogonal (Camera View)
2. Single-player
3. 3rd Dimensional (3D)
4. Survival Shooter

1. **Orthographic (Camera View):** The orthographic camera view is of the 3rd Person camera view form. What this means is that, players can see all of the PC during gameplay.
2. **Single-player:** Only one player can control the PC and play the game at a time.
3. **3rd Dimensional:** The game will be rendered and played with respect to all three coordinates (X, Y and Z axis).
4. **Survival Shooter:** Players will have unlimited ammo. The gamed is scored with respect to the number of NPCs killed. The game is over when the PC dies (that is PC Health=0).

Basically, the game involves the PC (wielding a gun) moving around the game environment shooting and evading the NPCs. The PC will be controlled by whomever is playing the game using the computer's keyboard and mouse. The NPCs on the other hand will be controlled by the AI components which include A* Pathfinding and FSMs.

1. Unity IDE: Unity is a cross-platform game engine developed by Unity Technologies and used to develop video games for PC, consoles, mobile devices and websites. First announced only for OS X, at Apple's Worldwide Developers Conference in 2005, it has since been extended to target 27 platforms.

Unity allows specification of texture compression and resolution settings for each platform that the game engine supports and provides support for bump mapping, reflection mapping, parallax mapping, screen space ambient occlusion (SSAO), dynamic shadows using shadow maps, render-to-texture and full-screen post-processing effects.

Unity is notable for its ability to target games to multiple platforms. Within a project, developers have control over delivery to mobile devices, web browsers, desktops, and consoles. Supported platforms include Android, Apple TV, BlackBerry 10, iOS, Linux, Nintendo 3DS line, macOS, PlayStation 4, PlayStation Vita, Unity Web Player (including Facebook), Wii, Wii U, Windows Phone 8, Windows, Xbox 360, and Xbox One (Wikipedia, 2017).

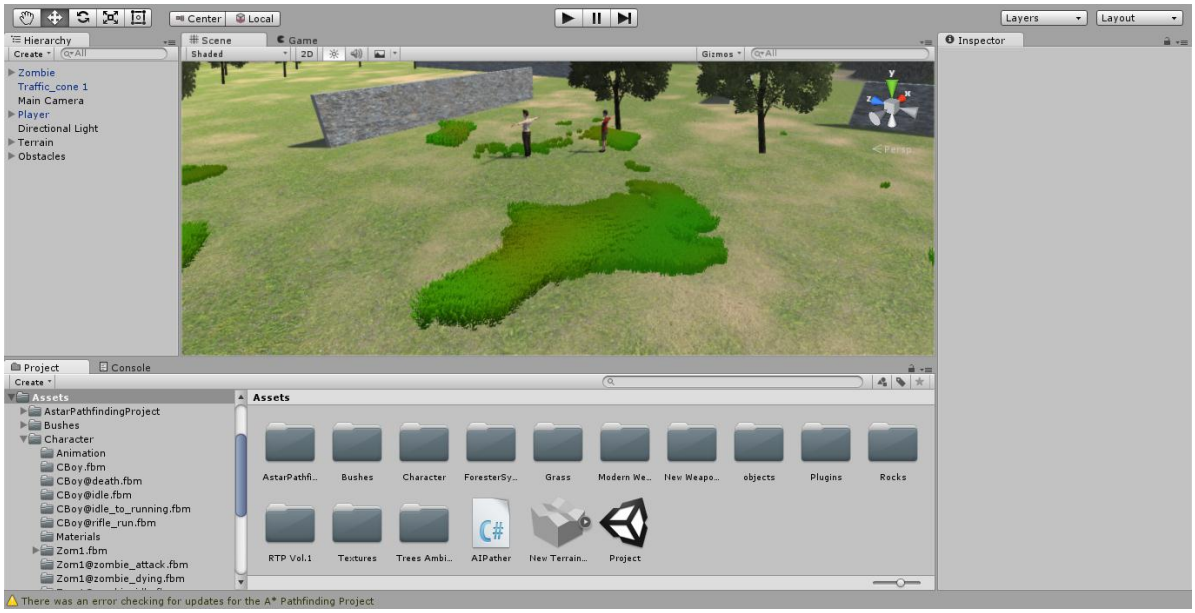


Figure 1. Image of the Unity IDE.

2. Xamarin Studio/MonoDevelop: MonoDevelop (also known as Xamarin Studio) is an open source integrated development environment for Linux, macOS, and Windows. Its primary focus is development of projects that use Mono and .NET frameworks. MonoDevelop integrates features similar to those of NetBeans and Microsoft Visual Studio, such as automatic code completion, source control, a graphical user interface (GUI) and Web designer. MonoDevelop integrates a Gtk# GUI designer called Stetic. It supports Boo, C, C++, C#, CIL, D, F#, Java, Oxygene, Vala, and Visual Basic.NET.

A customized version of MonoDevelop ships with Unity, the game engine by Unity Technologies. It enables advanced C# scripting, which is used to compile cross-platform video games by the Unity compiler.

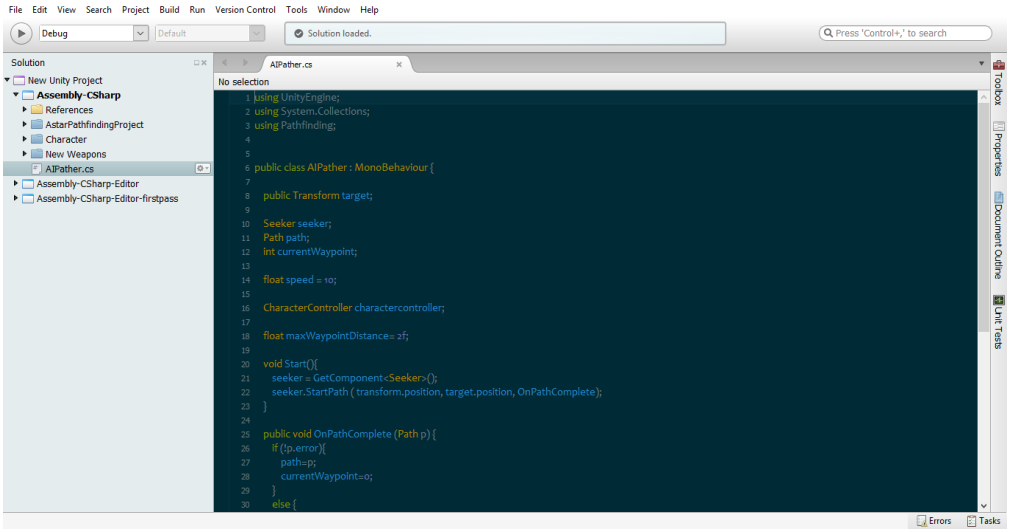


Figure 2 Image of Xamarin Studio/Mono Develop.

3. Adobe/Mixamo Fuse: Adobe Fuse CC (formerly Fuse Character Creator) is a 3D computer graphics software developed by Mixamo that enables users to create 3D characters. Its main novelty is the ability to import and integrate user generated content into the character creator. Fuse is part of Mixamo's product suite and it is aimed at video game developers, video game modders, and 3D enthusiasts.

Fuse is a client based product that lets users choose and modify character components such as body parts in real-time. Users can also customize their characters with clothing and texture options provided by Allegorithmic. Fuse's main novelty is the ability for users to import and automatically integrate their own content into the character creation system, leveraging all the features of pre-loaded content. Fuse characters are rigged through Mixamo online service. Characters have a bone driven rig and a blend shape based facial rig for facial animation.

Figure 3. Image of Adobe/Mixamo Fuse.

METHODOLOGY

In Game Development, there are numerous Game AI methods/techniques through which intelligent agents can be created some of which include; Navigation, Hunting, Path-finding, Finite State Machines and so on.

This study will make use of Finite State Machines (for agents to perceive their environment) and A* Path-finding (for agents to act on their environment) to create the intelligent agents. These Game AI techniques will be implemented using C# Scripts on the Unity IDE. Because of the complexity of the implementation of the A* pathfinding with respect to programming know-how, a Unity 3D plugin known as "Arongranberg's A* Pathfinding Project" will be used for implementing A* pathfinding.

CONCEPTUAL DEVELOPMENT PROCESS

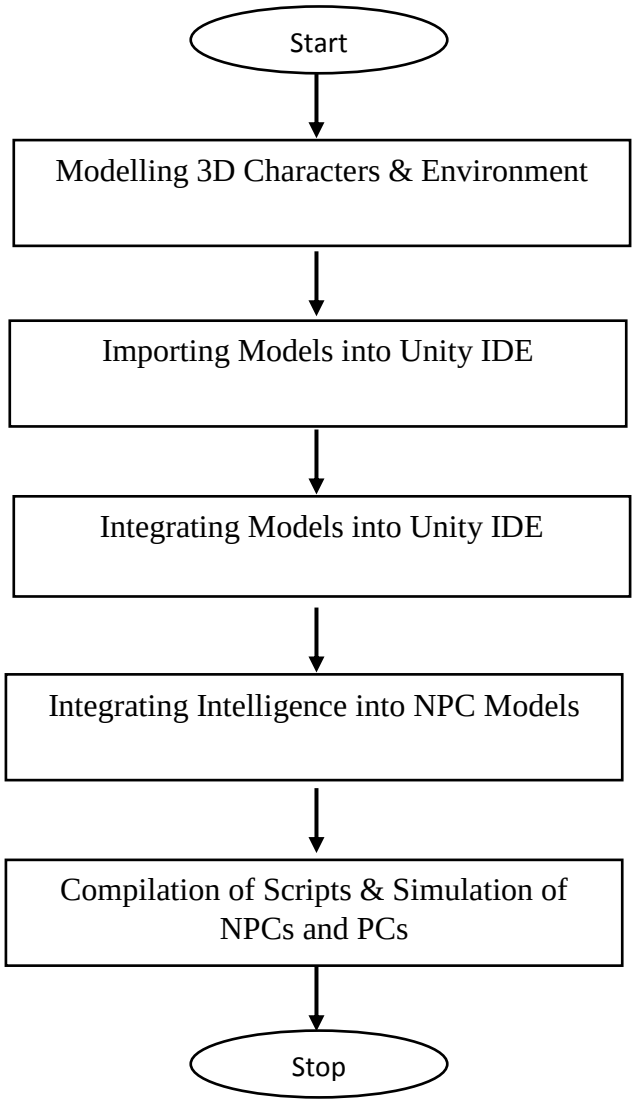


Figure 4. Flowchart for the development process

In relation with the above flow chart, the first phase involves the modelling of the 3D characters and the 3D environment. The characters will be modelled using Adobe/Maximo Fuse and the environment will be modelled on the Unity IDE.

The second phase is concerned with importing the already made 3D characters alongside other downloaded 3D models into the Unity IDE.

The third phase which is about Integration is concerned with creating animation flows, writing scripts for PC movement, specifying PC and NPC attributes and so on.

The fourth phase focuses mainly on the NPC. This is where the NPCs will be made intelligent agents by applying A* Pathfinding and FSMs respectively.

And finally, the fifth phase will involve combining all the components and compiling all the scripts on the Unity IDE.

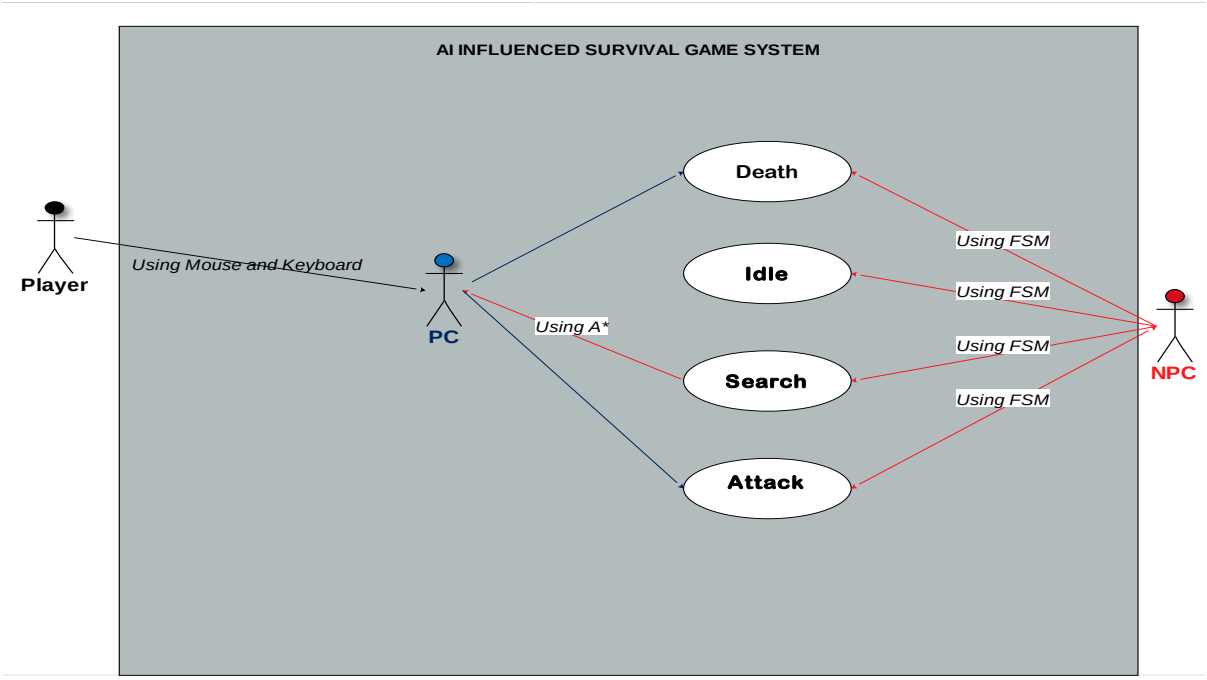


Figure 5. Image of System Use Case Diagram.

The above image is a simple description of the system using Use Case. From the diagram we can deduce the following:

1. Player has one state which is taking control of the PC via Mouse and Keyboard.
2. The PC also has two states; to attack as many NPCs as possible and also die.

NPCs have four states; Idle, Death, Search and Attack. The Idle, Death and Attack states are influenced only by the FSM component while the Search state is influenced by both the FSM component and the A* Pathfinding component.

NPC Behavior Pseudo-code:

1. START
2. If NPC health > 0
3. Go To 5
4. Else enter action state DEATH
5. Initialize PC Health to 100 (PC Health = 100)
6. Initialize Attack_Rate to 5 (Attack_Rate=5)
7. Enter action state IDLE
8. If PC is Visible
9. Enter action state SEARCH
10. Else
11. Go To 5
12. If PC is found and close enough to attack
13. Enter action state ATTACK
14. PC Health = PC Health – Attack_rate
15. Else
16. Go To 5
17. If PC Health != 0
18. Go To 11
19. Else
20. Go To 5 || STOP

NPC Behavior Flow Chart

The above flowchart was adopted from a typical A* Pathfinding algorithm. The algorithm is a complex one, as a result assistance will be needed for its implementation.

As stated earlier the implementation will be carried out using Arongranberg's A* Pathfinding Project. This software comes as a plugin for the Unity IDE with two prominent versions (Free and Pro). The Free version comes free of charge as well as with constraints, one of which is being limited to the use of Grid Graphs. The Pro version on the other hand requires payment and gives full access to the plugin's features.

Euclidean Distance Algorithm:

```
1 p: position(s) of seeker (Array)
2 q: position(s) of target (Array)
3 n: space/range of values
4 d: distance between seeker(p) and target(q)
5 a: difference of p and q
6 i: Index counter
7 p{n}
8 q{n}
9   for i = 0;i <n;i ++
10        a = (p[i] - q[i])
11           a-square = a * a
12        a-square = a-square + a-square
13   end for
14 d = square-root(a-square)
```

Manhattan Distance Algorithm:

```
1 p: position(s) of seeker (Array)
2 q: position(s) of target (Array)
3 n: space/range of values
4 d: distance between seeker(p) and target(q)
5 a: absolute difference of p and q
6 i: Index counter
7 p{n}
8 q{n}
9         for i = 0;i < n;i ++
10          a = abs(p[i] - q[i])
11          a = a + a
12        end for
13 d = a
```

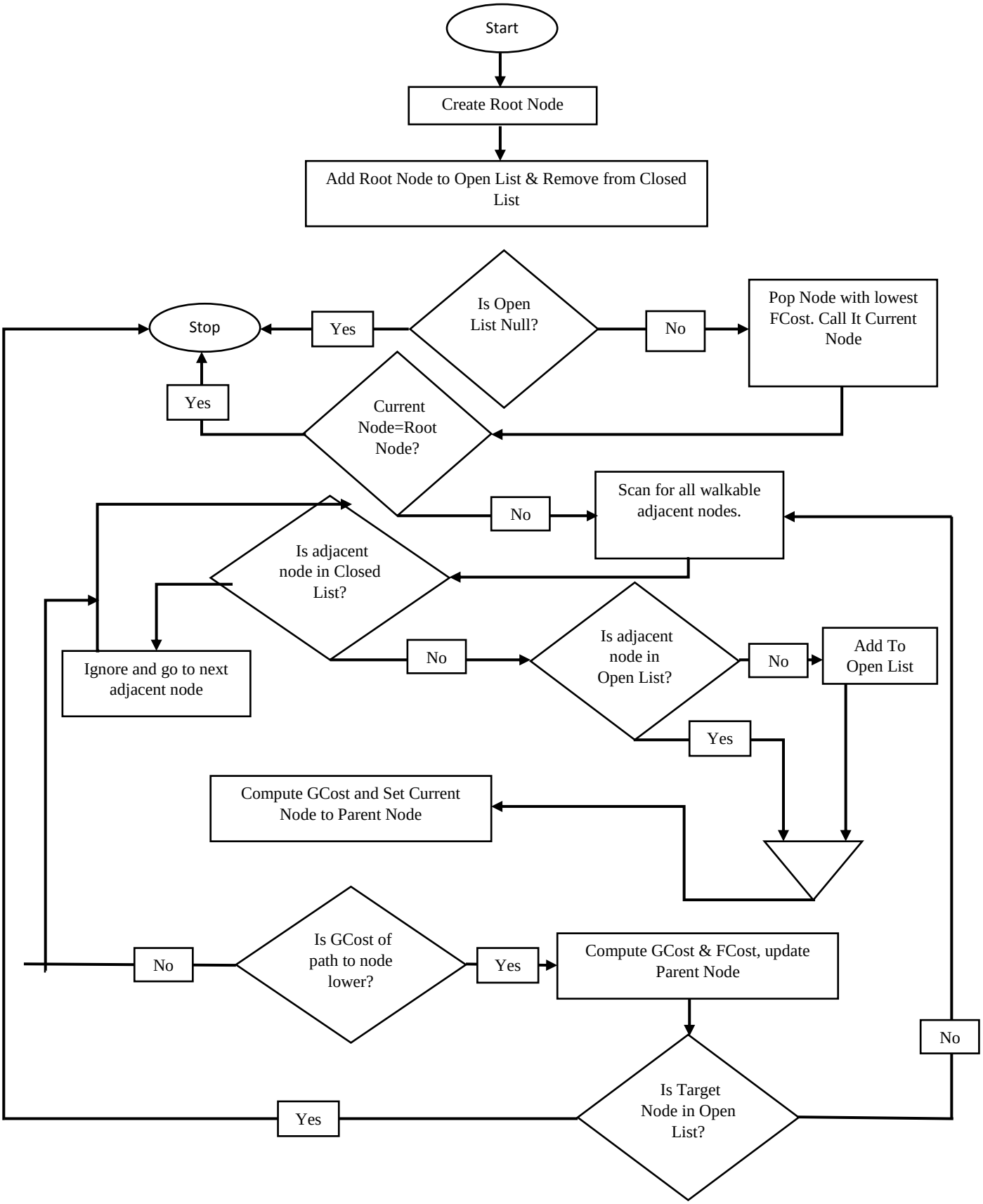


Figure 6. Flowchart for A* Pathfinding Heuristics Algorithms

RESULTS AND DISCUSSION

CHARACTER AND ENVIRONMENT MODELLING

This phase was done with Mixamo/Adobe Fuse and the Unity IDE. The characters (that is NPCs and PCs) for the game were modelled on Mixamo/Adobe Fuse modelling software after which they (modelled characters) were rigged and animated on the Mixamo animation website while the game environment which the characters will interact in was modelled on the Unity IDE. After animating the modelled characters on the Mixamo animation website, the characters were downloaded in (.fbx) format for the Unity IDE.

Figure 7. NPC Modelled on Mixamo/Adobe Fuse.

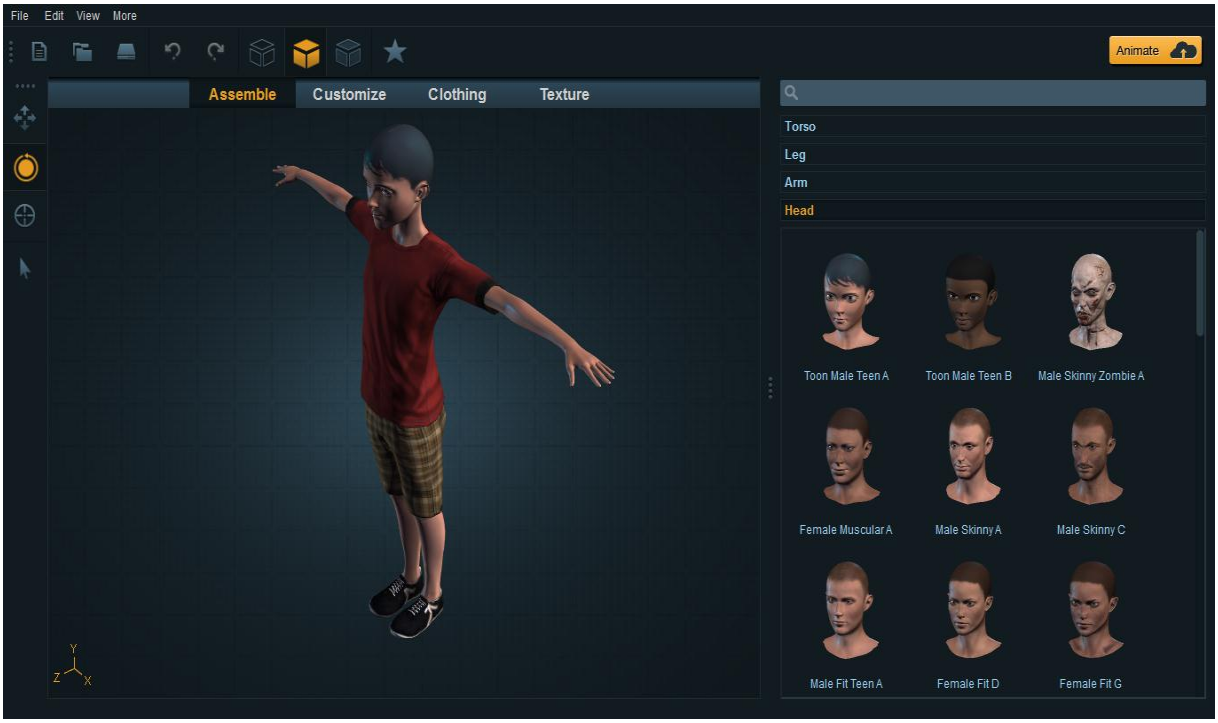


Figure 8. PC Modelled on Mixamo/Adobe Fuse.

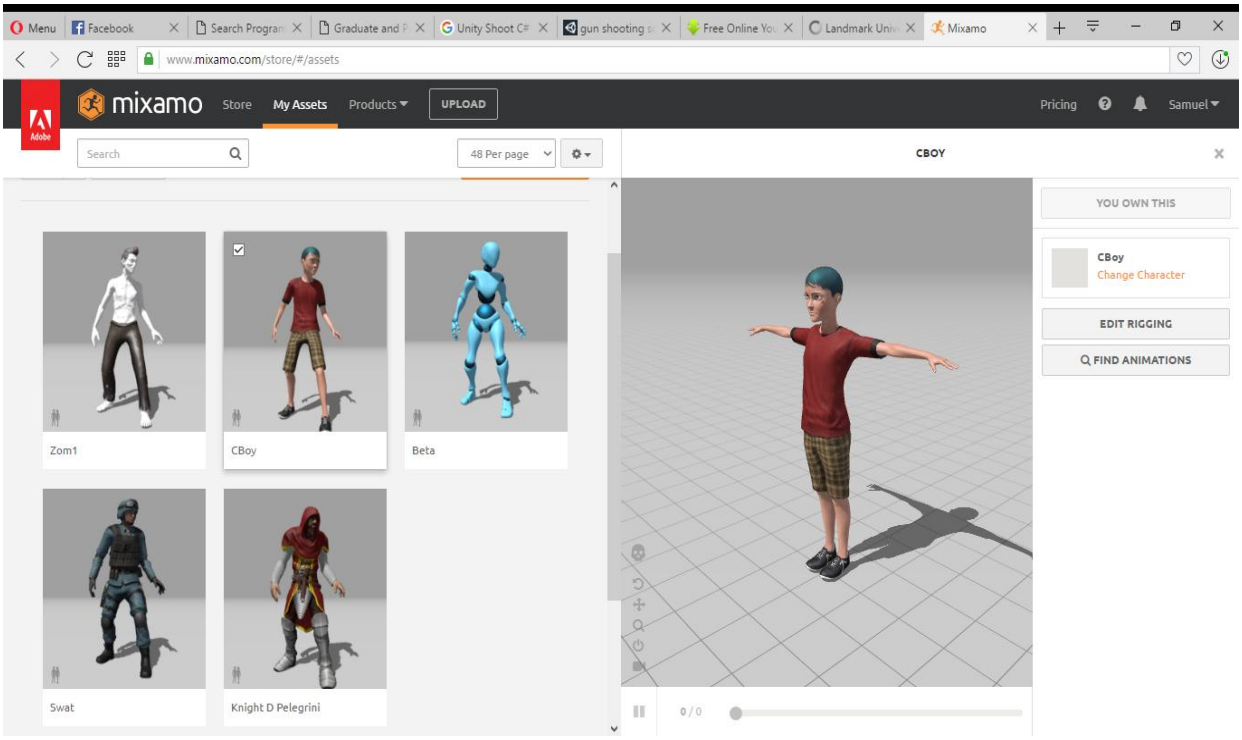


Figure 9. Mixamo/Adobe Fuse online rigging and animation webpage.

As for the environment modelling, environment prefabs were downloaded from the internet, after which they were imported into the Unity IDE. The environment was created using Unity's Terrain Component.

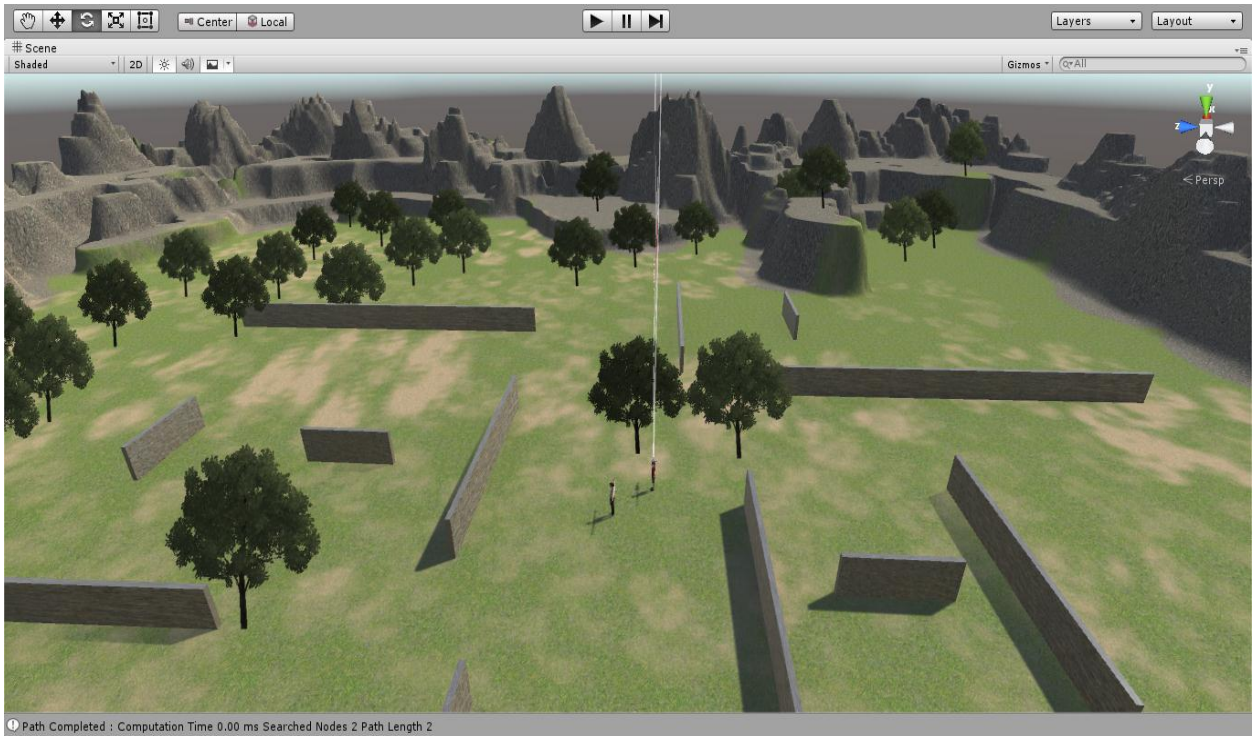


Figure 10. Environment Modelled on Unity.

IMPORTING MODELS

In this phase of implementation, the previously modelled characters were imported into the Unity IDE. Importing in Unity is based on whether the assets (models) of Unity origin or of Third party origin, as such, the process of importing is different based on that (origin). Third party assets can be imported simply by dragging the item into the "Project" tab on the Unity IDE. Unity assets on the other hand can be imported by clicking the "Assets" button by the top-left corner of the Unity IDE. The Character models to be imported are of third party origin so they were simply dragged into the "Project" tab.

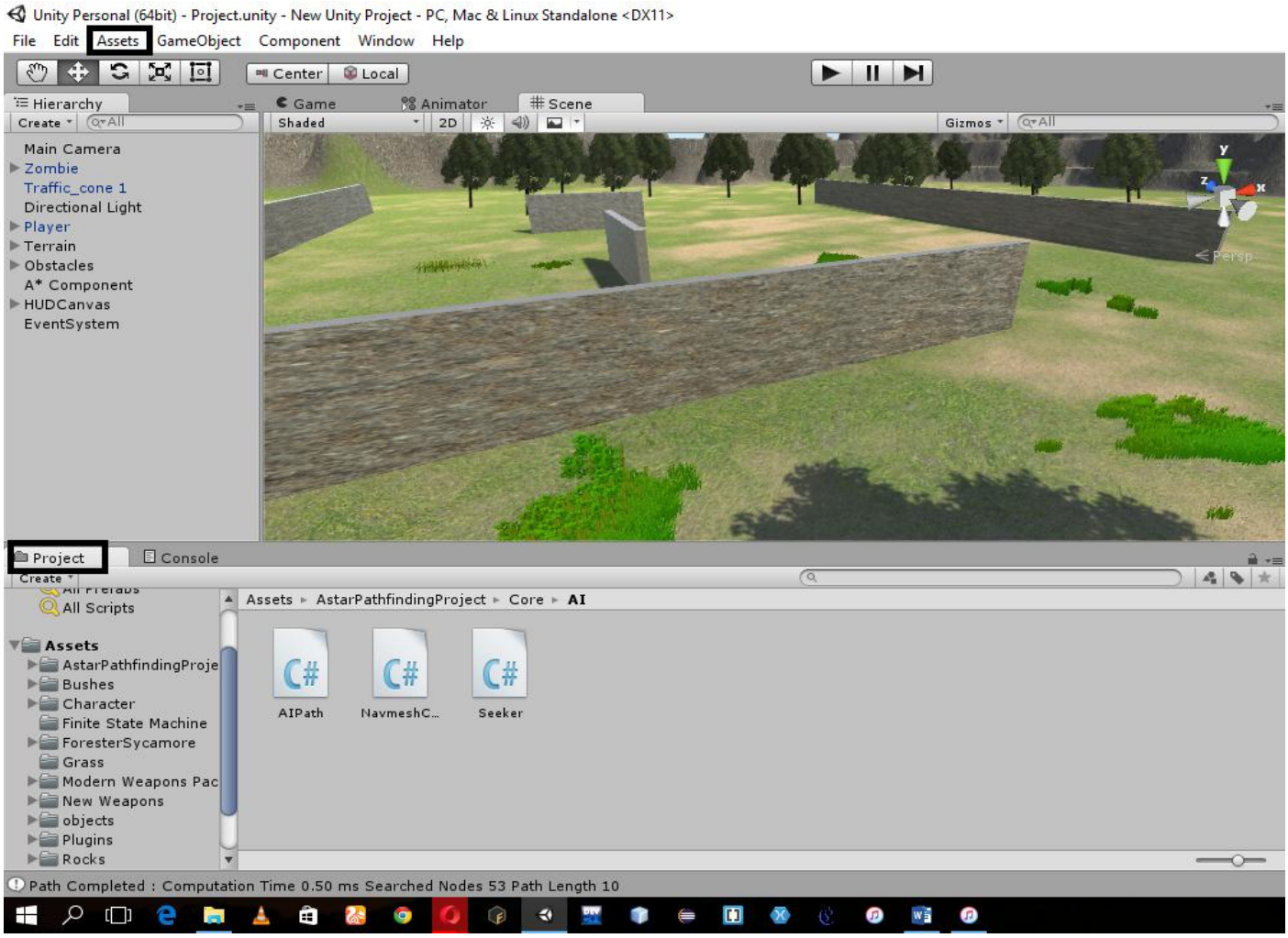


Figure 11. Image highlighting the "Assets" Button and "Project" Tab on Unity.

INTEGRATING MODELS AND COMPONENTS

This phase is concerned integrating the previously imported characters into Unity based on what kind of game is to be implemented. Necessary Unity components such as Animator, Rigidbody, Capsule Collider, Sphere Collider, Character Controller, Scripts and so on were integrated into the NPC and PC.

For the PC, the following Unity components and scripts were integrated;

1. Animator
2. Rigidbody
3. Capsule Collider
4. Player Movement (Script)
5. Weapon Attachment (Script)
6. Player Health (Script)
7. Mecanin Animator (Animation Controller)

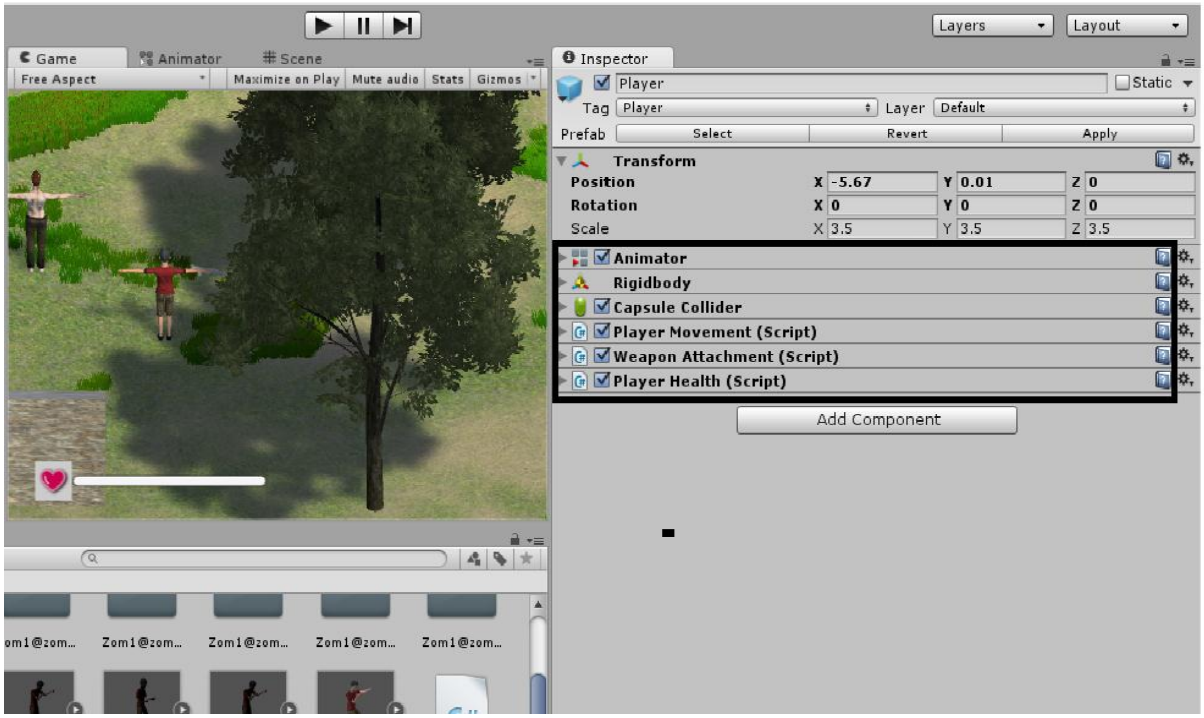


Figure 12. Image highlighting PC Components.

For the NPCs, the following Unity components and scripts were integrated;

1. Animator
2. Rigidbody
3. Capsule Collider
4. Sphere Collider
5. Character Controller
6. Enemy Health (Script)
7. A* Pathfinding Seeker (Script)
8. A* Pathfinding AI Path (Script)
9. Enemy Attack (Script)

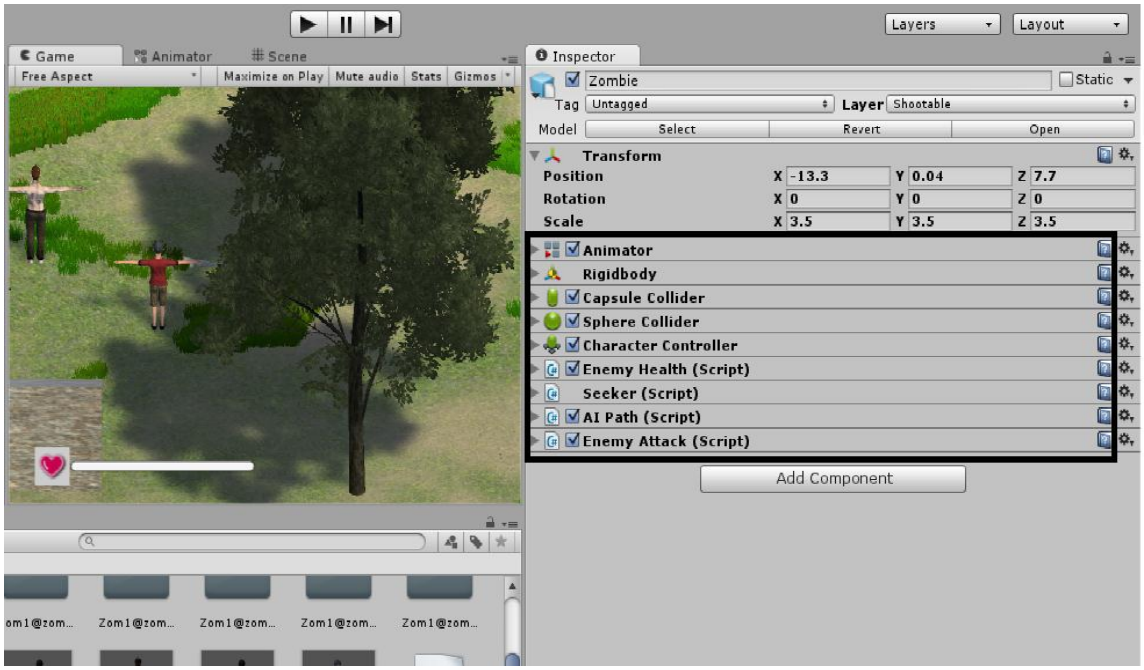


Figure 13. Image highlighting NPC Components.

For the Environment, two Unity components were created in the "Hierarchy" tab;

1. Main Camera
2. Terrain
3. HUD Canvas

Main Camera: This Unity component is responsible for the camera settings and the following sub components were integrated into the Main Camera;

1. Camera
2. GUI Layer
3. Flare Layer
4. Camera Follow (Script)

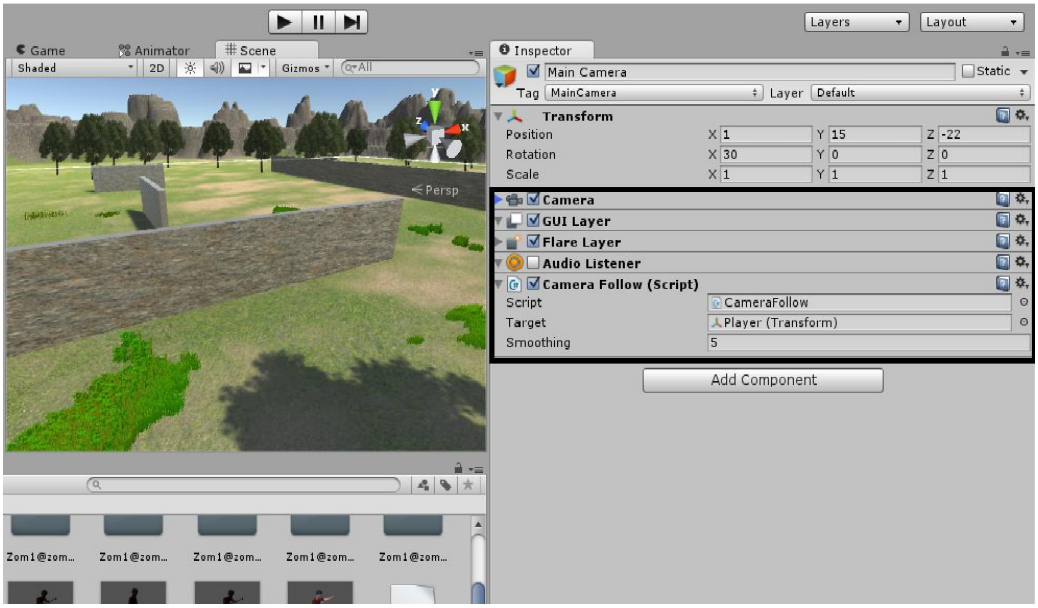


Figure 14. Image highlighting Main Camera components.

Terrain: This Unity component is responsible for creating and styling the environment. The following sub component was integrated into the Terrain

1. Terrain (Editor)
2. Terrain Collider

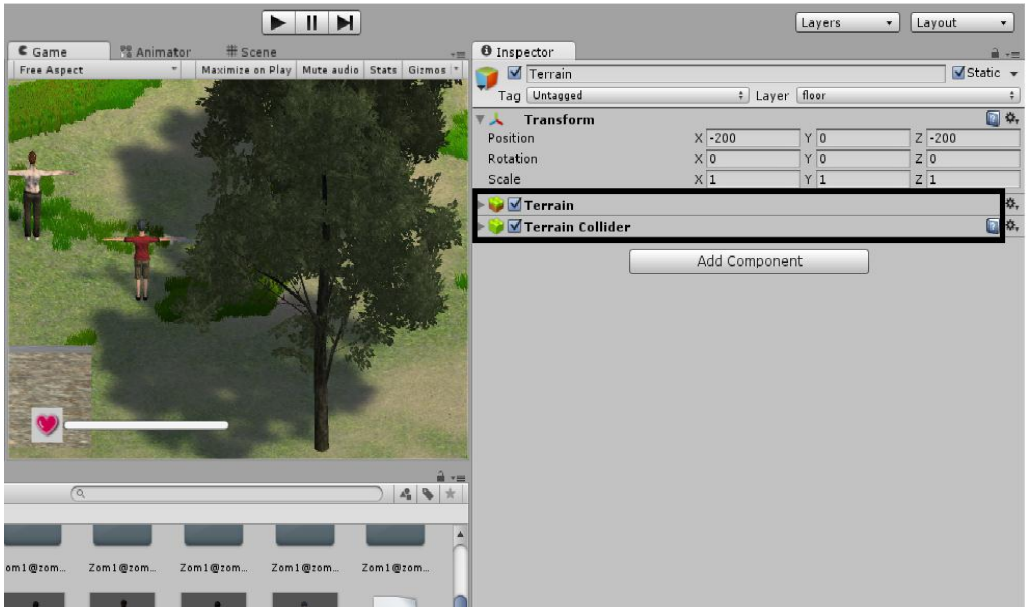


Figure 15. Image highlighting Terrain components.

HUD Canvas: This Unity component is responsible for the Health UI and Damage Image. The following sub components were integrated into the HUD Canvas;

1. Rect Transform
2. Canvas
3. Canvas Scaler (Unity Script)
4. Graphic Raycaster (Script)
5. Canvas Group

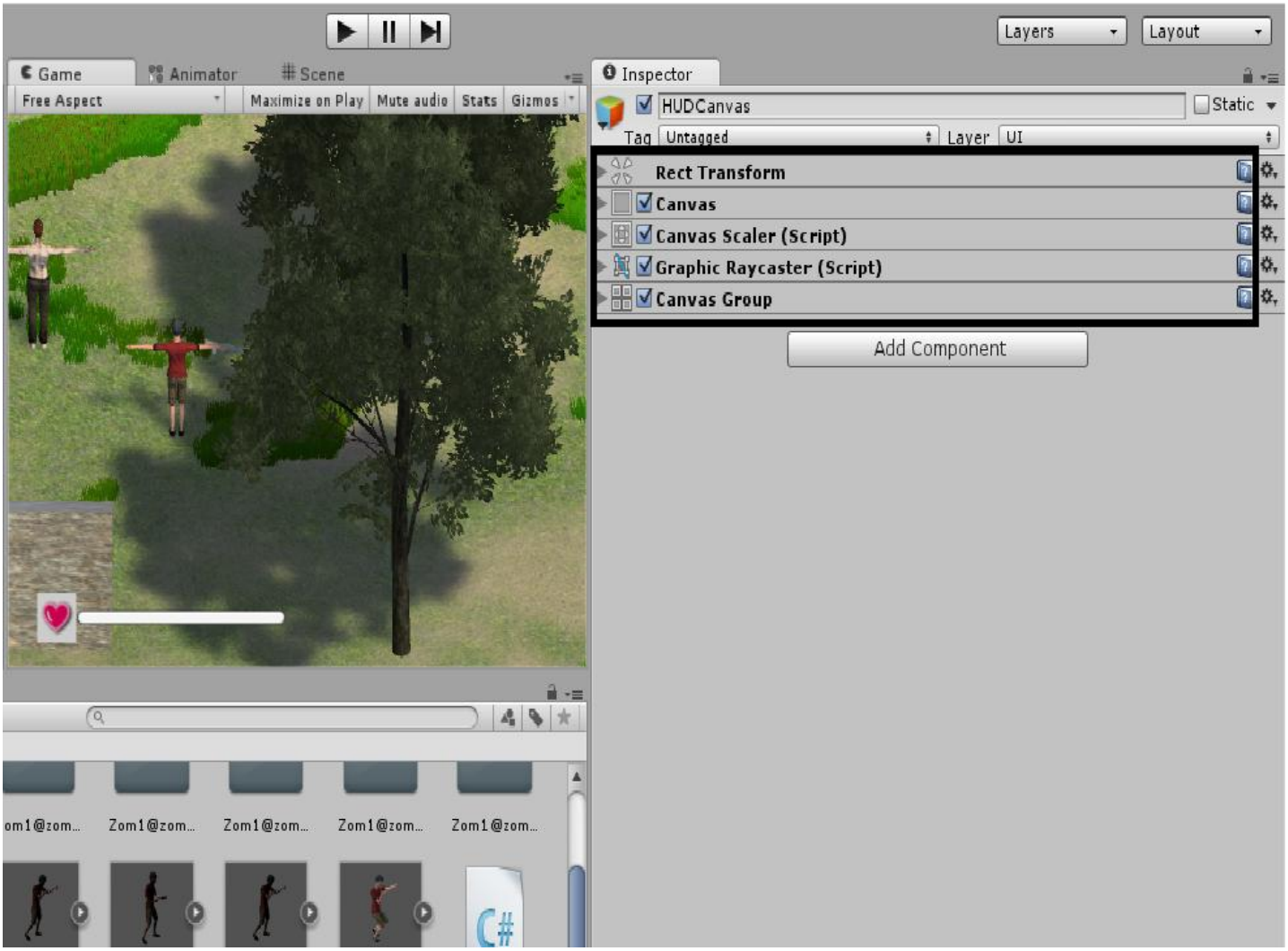


Figure 15. Image highlighting HUD Canvas components.

INTEGRATING INTELLIGENCE INTO NPCs

This phase is where the NPCs are made intelligent by integrating A* Pathfinding and Finite States Machines into them.

As mentioned earlier, a Unity plugin in the name of "Arongranberg's A* Pathfinding Project" was used to implement the A* Pathfinding. FSMs on the other hand was implemented by creating an Animation Controller (Mecanim Animator) for the NPC and Scripts in C#

A* PATHFINDING INTEGRATION

The following steps should be and were taken to integrate the A* Pathfinding plugin into Unity and into the NPCs;

1. Import the A* Pathfinding plugin by opening its Unity package file and then select "Import" on the Unity window that pops up.
2. Create an empty game object and named it.
3. Add the Pathfinding component by selecting "Component" (Top-left corner of Unity UI), then "Pathfinding", and then "Pathfinder". This opens up in the "Inspector" tab.
4. Add a new graph by selecting "Graphs" and then "Grid Graph".
5. Input "Node Width", "Node Depth" and "Node Size".
6. Specify value of the "Mask" subfield under "Collision Testing".
7. Specify value of the "Ray Length" and "Mask" subfields under "Height Testing".

8. Select the "Scan" button at the bottom of the "Inspector" tab.
9. Attach "Seeker" and "AI Path" scripts to the NPC.
10. Under "AI Path" component, specify the "Target" game object as the PC object.

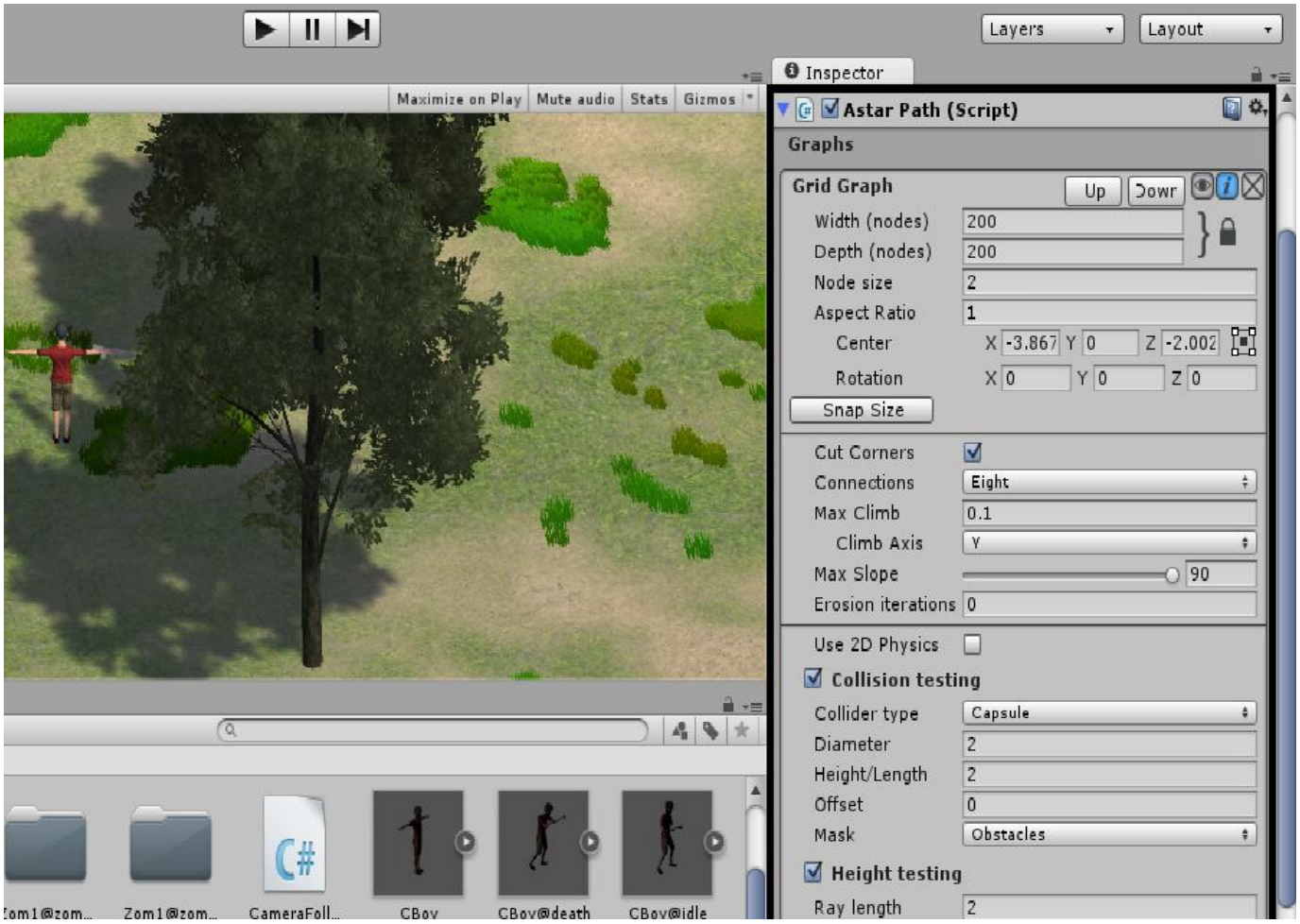


Figure 16. Image highlighting A* Plugin interface.

FINITE STATE MACHINES INTEGRATION

As mentioned earlier, to integrate the FSMs, an Animation Controller (Mecanim Animator) was created for all the available animations of the NPC as well as C# Scripts. Some of these scripts were embedded in the "AI Path" script gotten from the A* Pathfinding plugin. The scripts can be seen in the Appendix. The Mecanim Animator provided the platform to specify the possible states (which are represented by the NPC and PC animations) into which the NPCs and PC can enter. It was also used to specify all the possible transitions and their respective triggers.

NPC FINITE STATE MACHINE

For the NPC, the following states were created via their respective Animations;

1. Zombie_run: represents SEARCH state.
2. Zombie_idle: represents IDLE state.
3. Zombie_attack: represents ATTACK state.
4. Zombie_dying: represents DEATH state.

The following Transitions were created;

1. Zombie_run ↔ Zombie_idle: NPC can move from SEARCH to IDLE and vice versa with triggers "PlayerDead" and "Running" respectively.
2. Zombie_run → Zombie_dying: NPC can move from SEARCH to DEATH with trigger "Dead".
3. Zombie_run → Zombie_attack: NPC can move form SEARCH to ATTACK with trigger "Attack".

4. Zombie_attack → Zombie_dying: NPC can move from ATTACK to DEATH with trigger "Dead".
5. Zombie_attack → Zombie_idle: NPC can move from ATTACK to IDLE with trigger "PlayerDead".

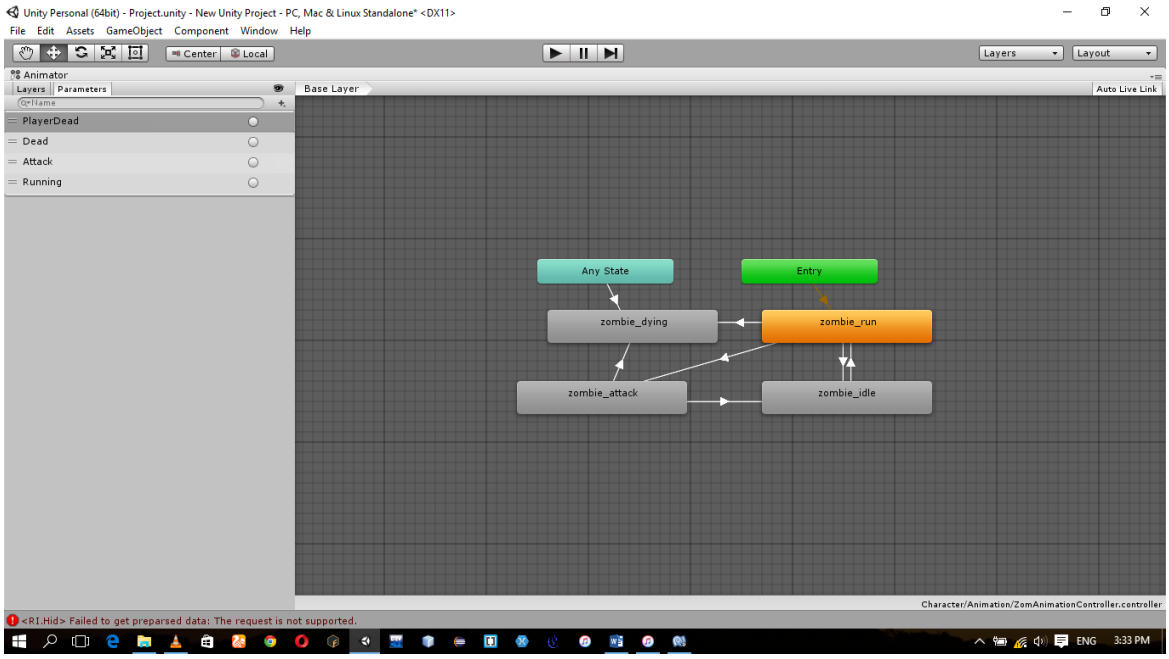


Figure 17. Mecanim Animation FSM for NPC.

SCRIPT COMPILATION AND AGENTS SIMULATION

In this phase, all the Non Unity scripts handling the various components of the game were all compiled and simulated alongside the gaming agents (NPCs and PCs) in the "Game" tab. These scripts include;

1. AI Path
2. EnemyAttack
3. EnemyHealth
4. PlayerHealth
5. PlayerMove
6. PlayerShooting
7. FSM
8. Score Manager

The process of compilation and simulation is to ensure the correctness of the scripts and their cohesion with other scripts. As such, this process was carried out more than once.

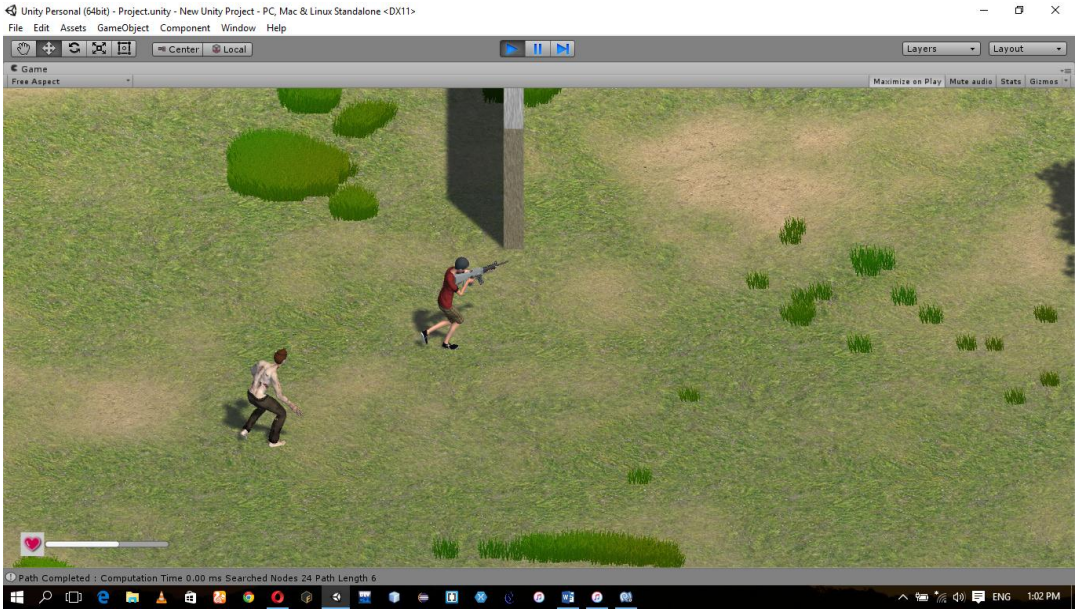


Figure 19. Image showing Agents Simulation.

CONCLUSION

With respect to the AI techniques used as well as the earlier mentioned objectives of this project, it was found that some level of agent intelligence, though not optimal could be achieved by combining A* Pathfinding (or any other equivalent) and Finite State Machines with the previously specified states and transitions.

A higher level of intelligence could also be achieved by improving the Finite State Machines component. This can be done by adding to the number of possible states, transitions and triggers (events). As for the A* Pathfinding, using a NavMesh graph instead of Grid graph would make way for Patrol waypoints which would make the agents even more intelligent.

The major usefulness of this project can be realized in the development of other survival based shooter games. Following this approach would provide any game of the same genre with the necessary level of agent intelligence required for player satisfaction.

RECOMMENDATIONS

On the game development platform, this project can act as a guide to developing games, and so, it is recommended to entry-level game developers. On the gaming platform, this project is recommended to gamers interested in the survival shooter genre. It also helps to enhance logical reasoning and strategic planning skills of players.

Finally, on the research platform, this project is recommended to researchers or even game developers who seek to design intelligent gaming agents.

CREDIT AUTHOR STATEMENT

Adekanmi A. Adegun: Supervision, Writng- Review & Editing, Conceptualization, Project Admininstration, **Roseline O. Ogundokun**: Writing- Original Draft, Visualization, Validation, Resources, **Samuel Ogbonyomi**: Conceptualization, Methodology, Software, Resources

FUTURE RESEARCH

Any further research on the aim and objectives of this project would revolve around achieving a higher or even optimal level of agent intelligence. Also, it could revolve around making the end product (the game) retail ready.

In achieving a higher level of agent intelligence, the following could be considered;

1. Different combination of AI techniques.
2. NavMesh graphs as an alternative to Grid graphs.
3. More FSM states and transitions.
4. Different game genre.
5. An alternative to A* Pathfinding and so on.

REFERENCES

[1] Bevilacqua, F. (2013, October 24). *Finite State Machines: Theory and Implementation.* Retrieved November 3, 2016, from https://gamedevelopment.tutsplus.com/tutorials/finite-state-machines-theory-and-mplementation--gamedev-11867

[2] Cui X., Shi H. (2011, January). *A*-based Pathfinding in Modern Computer Games. International Journal of Computer Science and Network Security.* VOL.11 No 1, pp. 125-130.

[3] Miikkulainen, R. et al, (2006). Computational intelligence in games. Computational Intelligence: Principles and Practice.

[4] Nareyek, A. (2002). *Intelligent Agents for Computer Games. Computers and Games, Second International Conference,* pp. 414-422.

[5] Rabin, S. (June 2009). *Introduction to Game Development.* (2nd ed.), pp. 521-558. Boston, USA: Course Technology.

[6] Restu, P. (2015, September 13). *Artificial Intelligence.* Retrieved October 7, 2016, from http://web.if.unila.ac.id/restupratiwi/2015/09/13/artificial-intelligence/

[7] Unity (game engine). (n.d). In *Wikipedia*. Retrieved February 24, 2017, from https://en.wikipedia.org/wiki/Unity_(game_engine)