INTERNATIONAL JOURNAL OF ACADEMIC RESEARCH IN PROGRESSIVE EDUCATION & DEVELOPMENT

OPEN ACCESS JOURNAL

Vol 12, Issue 3, (2023) E-ISSN: 2226-6348

First-person Shooter (FPS) Video Game Using Ray Casting Algorithm

Mohamad Hafiz Khairuddin, Nurazian Mior Dahalan, Mohd Rahmat Mohd Noordin, Anis Amilah Shari, Amar Syahmi Sulaiman

School of Computing Sciences, College of Computing, Informatics, and Mathematics, Universiti Teknologi MARA Cawangan Melaka Kampus Jasin, 77300 Merlimau, Melaka, Malaysia

Corresponding Author Email: hafizk@uitm.edu.my

To Link this Article: http://dx.doi.org/10.6007/IJARPED/v12-i3/19205 DOI:10.6007/IJARPED/v12-i3/19205

Published Online: 20 September, 2023

Abstract

A personal computer has evolved where it can be used to perform various tasks such as 3D design, video rendering, and other similar activities. Video games are not an exception, and many individuals are turning to YouTube or Twitch streaming to earn money while playing video games on their computers. Developing a video game is not a simple task since the game's developer requires specific expertise to produce them. Without this knowledge, the developer would be unable to develop the video game. This project focuses on the application of the ray-casting algorithm. Ray casting is one of the algorithms that is frequently utilised in the construction of video games, particularly in shooting games. This is because the algorithm is straightforward and does not necessitate the use of high-end technology to conduct the ray casting algorithm. It will be necessary to construct a 3D shooting game for the ray casting technique to function well since this will allow for easier analysis of the accuracy of the bullet hitbox. This project demonstrates how ray casting works based on all the results made while developing the game prototype. This project focuses on becoming an example for other video game developers to develop video games using the ray casting algorithm.

Keywords: First-Person-Shooter (FPS), Ray Casting algorithm, Video Games, Bullet Hitbox, 3d Shooting Game

Introduction

CGI (computer-generated imagery) and other applications rely on ray casting, which is the most fundamental type of raytracing available. When compared to other forms of ray-tracing, in which rays start from a light source and bounce off objects before reaching the observer, ray casting is distinguished by the fact that rays are "cast" straight from the viewpoint during the process. In the ensuing image, each time one or more cast rays intersect an object, the colour and brightness of the object at that precise point determine the value of one pixel in the final picture. It is represented as a direction vector that extends ahead in time, which represents the observer's orientation which has been stated by Computer Hope (2019). As an additional need for almost all methods of ray casting, a "camera plane" that is perpendicular to the direction vector and represents the geometry of the final displayed image must be used in conjunction with the ray casting. Figure 1 shows the example of ray-casting scenario point-of-view in the algorithm.

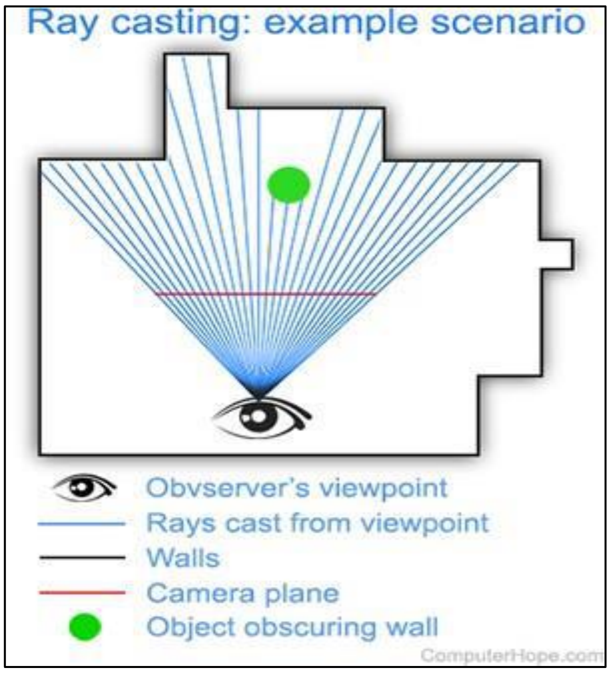


Figure 1 Ray casting scenario view
(Source: ComputerHope.com, 2019)

It is possible to create three-dimensional images from a restricted set of data by tracing rays from the viewpoint into a viewing volume, which is achieved by ray casting techniques. It is important to understand that rays can be cast and traced in groups based on geometric limitations, which is the fundamental notion of ray casting stated by Techopedia (2019). The intersection of all objects in the picture is computed by casting a ray from the pixel through the camera in the ray casting process. Following that, the pixel value from the nearest intersection is acquired and used as the foundation for the projection. When compared to ray tracing, ray casting is a rendering method that never recursively traces secondary rays, whereas ray tracing is a rendering approach that can recursively trace secondary rays. In addition, when compared to other rendering methods such as ray tracing, ray casting is extremely simple to implement and operate. It is simple and quick to use ray casting since only a single computation is required for each vertical line of the screen. Ray casting is faster than ray tracing because it is constrained by one or more geometric restrictions, as opposed to ray tracing. Another reason why ray casting was the most common rendering technology in early 3-D video games is that it allows for more realistic lighting effects by Techopedia (2019). This project studies the application of the ray casting algorithm to see how it can be developed and applied in a first-person shooter video game development. The current graphical processing technology for computers focuses on ray casting technology and therefore, it is imperative to have an academic understanding of how ray casting algorithm works. This project is important for students of multimedia discipline where they will have an insight into ray casting technology and the opportunity to further explore the application of ray casting in video game development.

Objectives

- To design a video game prototype that can be used to test the selected algorithm by using the ray casting algorithm.
- To develop the ray casting algorithm based on how the aiming hit box bullet shot on an object in the prototype game.
- To test the functionality of the proposed aiming game.

Problem Statement

An issue that arises is when choosing which aiming detection algorithms are the best ones and how we best use them. If the game developer did not concern about the accuracy of the bullet that is being shot using a certain detection algorithm, then players will not be interested in playing that game. Jung (2019) has concluded that there are too many algorithms for bullets to be detected when hitting an object. As an example, when players shoot their gun to hit another player/object in front of them, which algorithm is the best to choose to detect the bullet hitbox is still open to debate. Another obvious issue is the cost of the required hardware to implement this project since new technology in algorithms is best to run with the latest hardware technology to properly produce the best results (Anik et al., 2019).

Methodology

Unlike Software Development products, which are always developed in response to a problem that has already been identified and for the purpose of providing an answer, Game Development products are developed for entertainment purposes to engage people in having fun, learning, and spending quality time with one another. This will demand the development of an original concept, a storyline, and writing ability in addition to technical competence to complete the product. Since developers must contend with a variety of obstacles during the development of the project, including artwork, visuals, animations, character interactions, collisions, physics, and sounds, among other things, we must use a specialized approach known as the Game Development Life Cycle (GDLC) (Jain, 2017).
There are 6 stages in the game development life cycle (GDLC): Idea & story, Conceptualize & Design, Technical Analysis, Development, Testing and Deployment. These stages are crucial to follow when game developers want to develop a new game in their workspace. The GDLC is being applied in a waterfall model paradigm. As (Game Development Lifecycle Models, 2021) stated, this paradigm was particularly popular during the early years of video game production (from the 1980s to the mid-1990s), when the requirements remained unchanged throughout the development process. It is suitable for usage in modest game development projects.

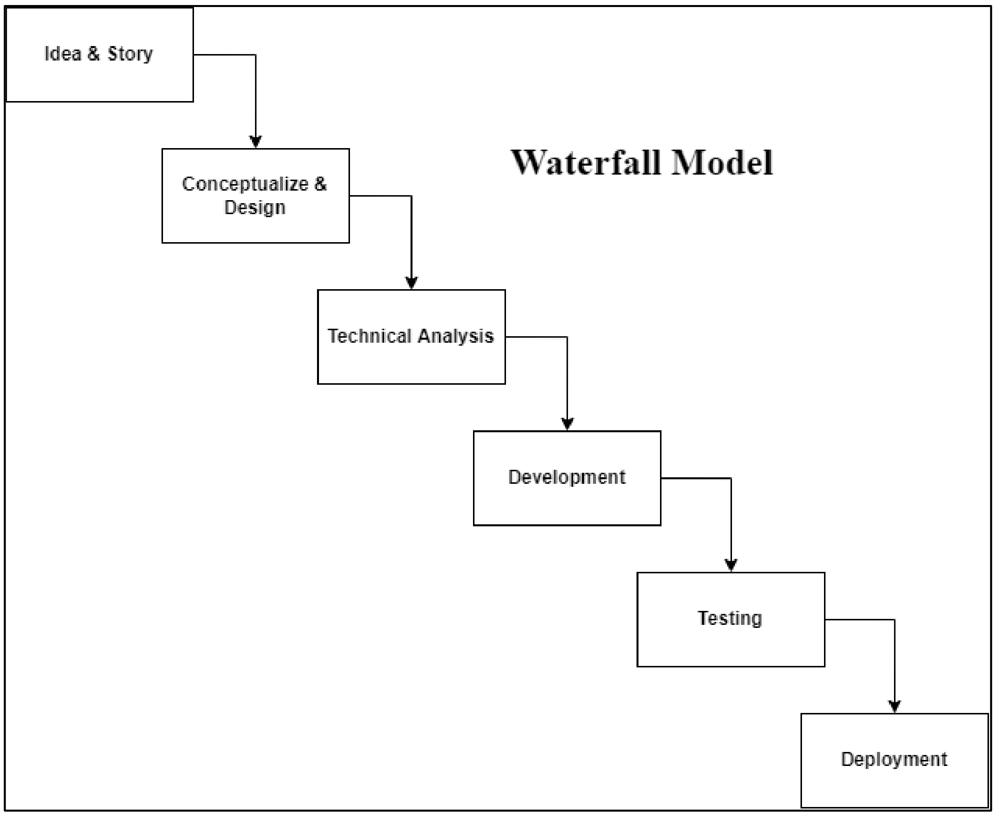


Figure 2 Game Development Life Cycle on Waterfall Model
(Source: *Game Development Lifecycle Models*, 2021)

Prototype Conceptualization and Realization

To conceptualize and realize the prototype for this project, the system design and the physical design will be combined to provide a more comprehensive picture of how the proposed application would function. This step is critical because it allows for the incorporation of information gained from a collision detection algorithm study into the design of the physical system. There are three main system design activities:

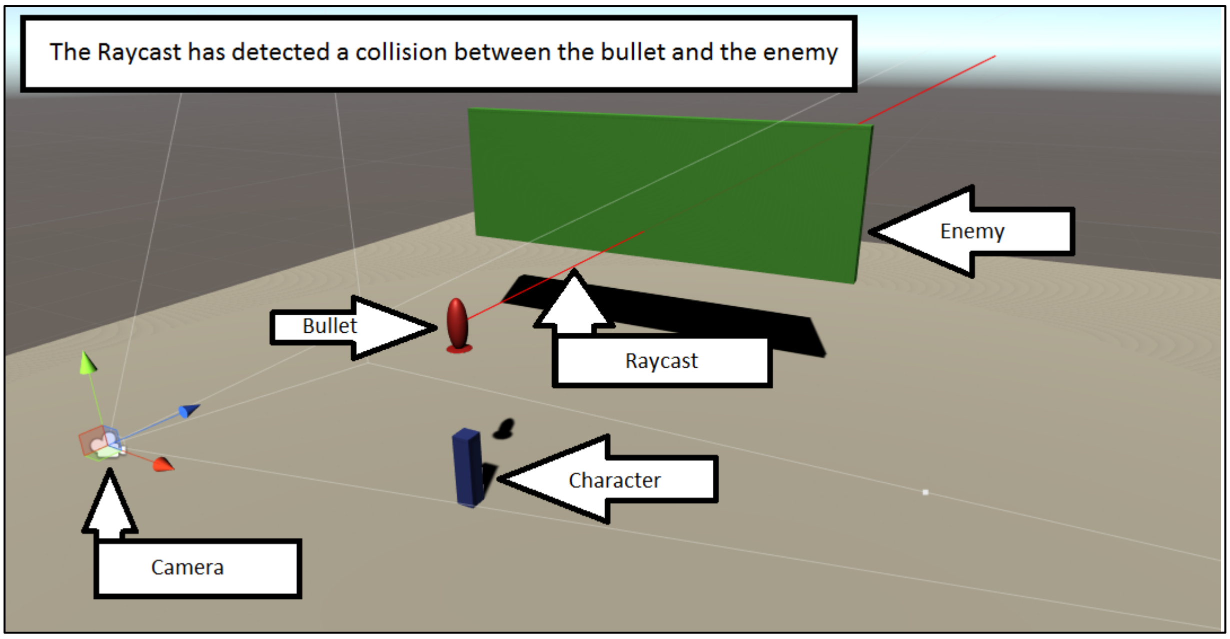
a) FPS Game Asset Requirement Acquired

It is necessary to download the asset to be used from the Unity Asset Store, which is completely free before the video game can be developed. This is necessary because the project's goal is to test the efficiency of bullet hitbox detection by using the gyroscope, rather than the graphical design, of the video game. One FPS Asset on Unity Asset Store can be downloaded, regarding which pack is used (Stenfors, 2019).

Figure 3 FPS Pack at Unity Assets Store
(Source: Stenfors, 2019)

Based on Figure 3, there are many FPS packs that can be downloaded free in Unity Assets Store, regarding which platform you want to use, it got all platforms available. From FPS pack to 3D character can also be got from that store.

b) Character Customization
After the FPS Pack Asset has been downloaded, the move set for the character is defined. This is critical since it has a direct impact on the bullet path and hitboxes that will be used to offer effects to the activities that are being performed (Unity, 2019). For example, if the player starts shooting, the hitbox detection will only apply to the bullet that travels through the target and not the rest of the projectile. Another example is that when a player throws an explosive, the dynamics of the explosive dropping will be implemented. Looking at Figure 4, we can see that when a character fires a gun at the target, the following happens: It is planned to use the ray cast technique to fire the bullet, which will hit the designated target.


The Raycast has detected a collision between the bullet and the enemy
Enemy
Bullet
Raycast
Character
Camera
Figure 4 Overview of the character shooting a bullet at the target
(Source: Unity, 2019)

c) Attack Move Frames and Recovery
The term "frame and recovery" refers to the amount of time that passes before the hitbox appears, the amount of time that it remains as a hitbox, and the amount of time that it takes to return to the idle state. Factors like this also have an impact on how well the fluency of the bullet hits the target and can have an impact on how much fun you have while playing the game. Based on Figure 5 shows the flowchart of the attack animation that's going to be implemented in this project. A flowchart is an important thing to check whether the animation or the attack animation of the character in the game is a success or a failure. The ray casting algorithm was used in the development of the game, and it was put through its paces numerous times to ensure that the ray cast functioned as intended.

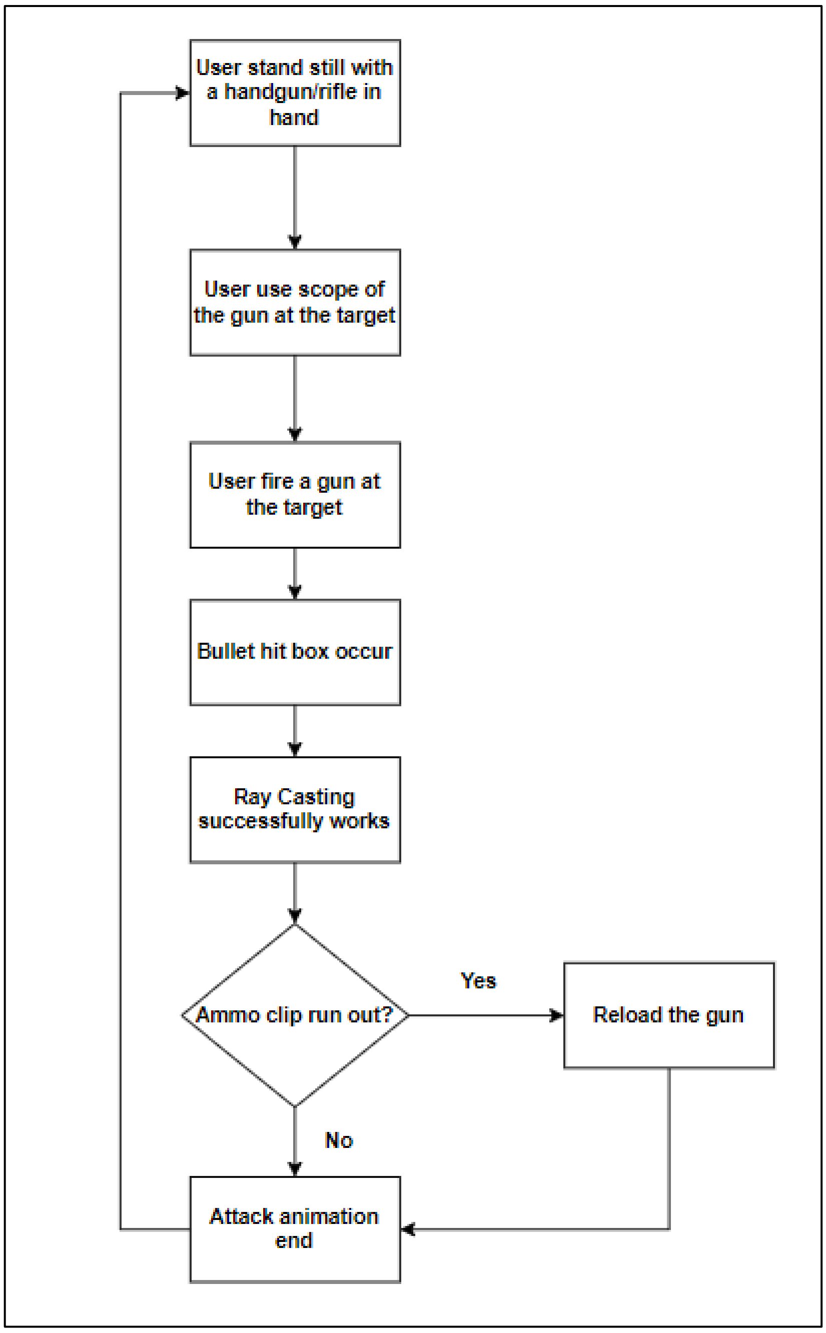

User stand still with a handgun/rifle in hand
User use scope of the gun at the target
User fire a gun at the target
Bullet hit box occur
Ray Casting successfully works
Ammo clip run out?
Yes
Reload the gun
No
Attack animation end

Figure 5 The cycle of attack animation on First Person Shooter games

Results and Discussions

This section discussed the system's design, development and testing in designing and developing a "First-Person Shooter (FPS) Video Game Using Ray Casting Algorithm":

a) System Design

A good game design system reflects how the system will deliver the overall content of the system without letting the user interfere with any of the technical defects. This subsection will further be explained the details of the game design starting with the game flow. Figure 6 below shows the overall flow chart of the game flow for "First-Person Shooter (FPS) Video Game Using Ray Casting Algorithm". The flowchart also tells how the gameplay starts from the beginning until the game is ended.

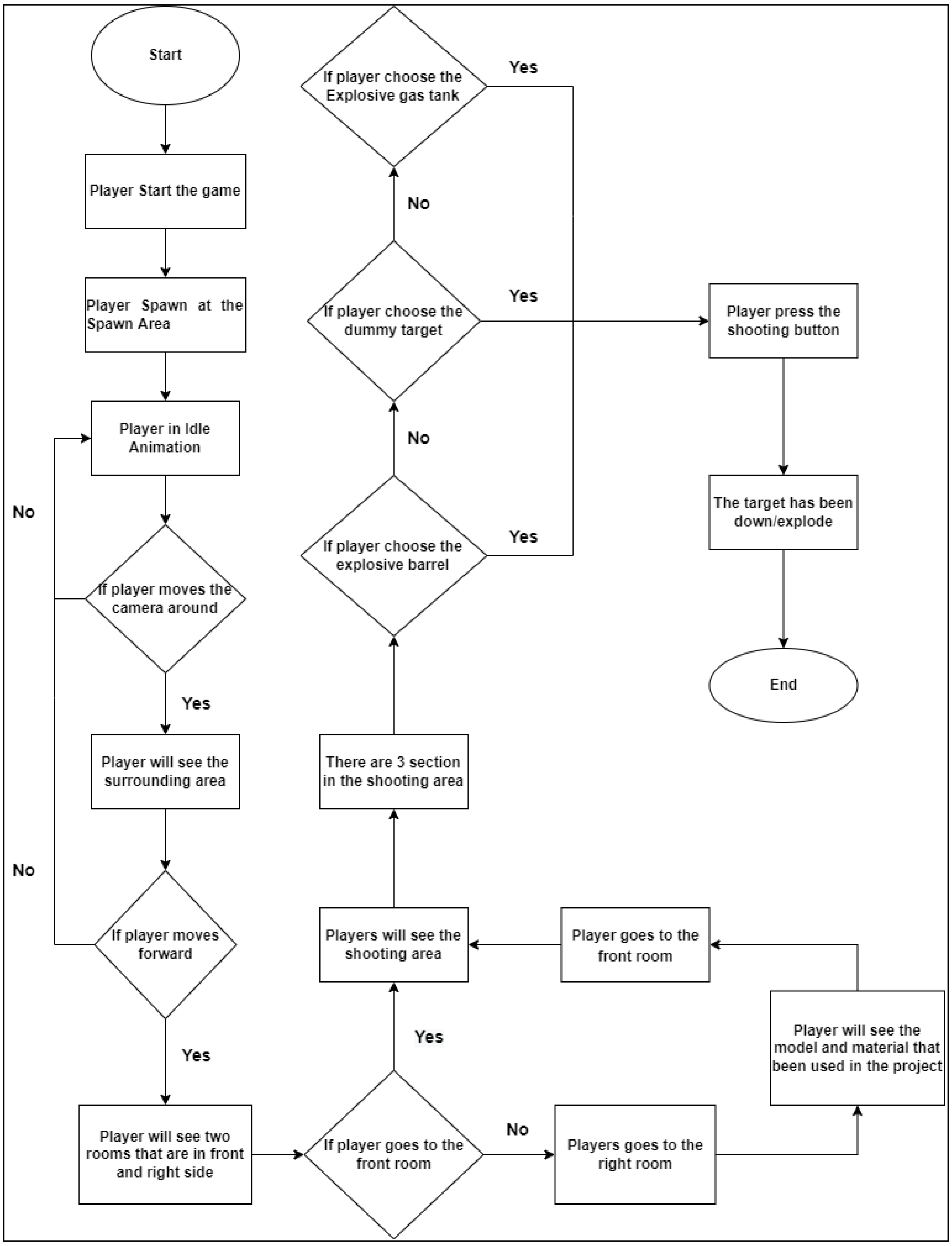
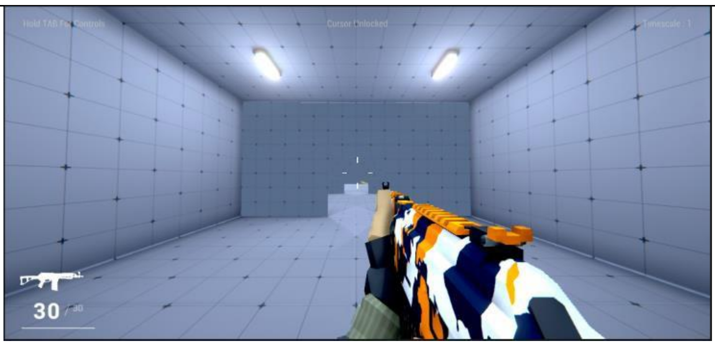
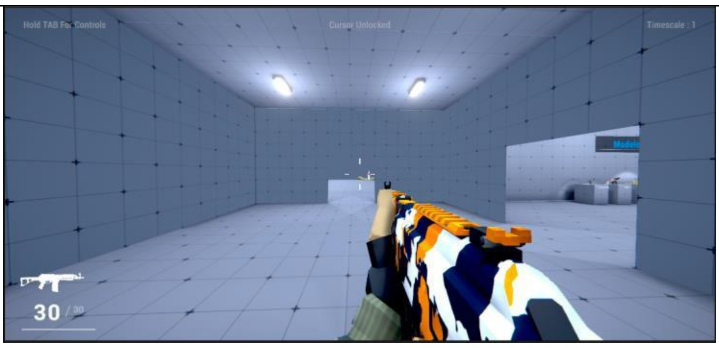
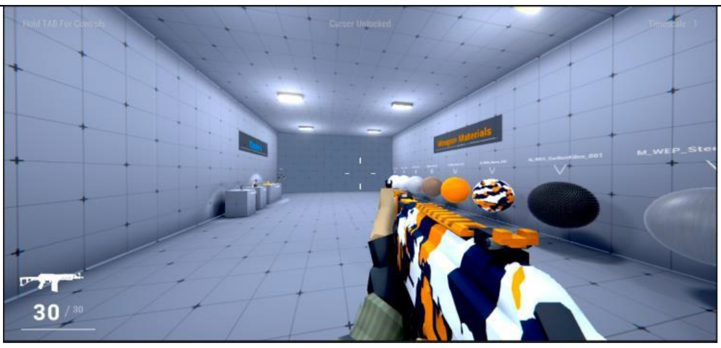
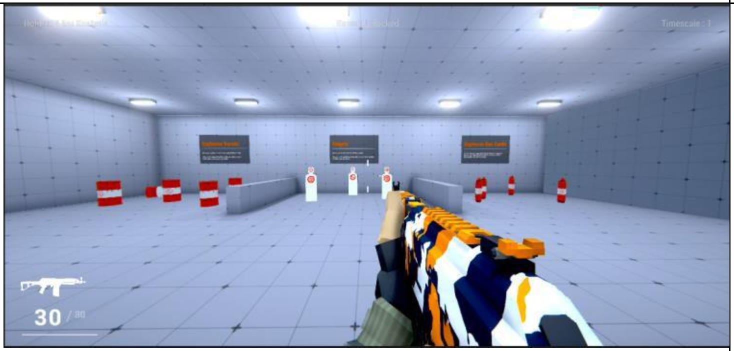

Start
Player Start the game
Player Spawn at the Spawn Area
Player in Idle Animation
If player moves the camera around
Yes
No
Player will see the surrounding area
If player moves forward
Yes
No
Player will see two rooms that are in front and right side
If player goes to the front room
Yes
No
Players goes to the right room
Player will see the model and material that been used in the project
Player goes to the front room
Players will see the shooting area
There are 3 section in the shooting area
If player choose the explosive barrel
No
Yes
If player choose the dummy target
No
Yes
If player choose the Explosive gas tank
Yes
Player press the shooting button
The target has been down/explode
End

Figure 6 Overall game flow chart for First-Person Shooter (FPS) Video Game Using Ray Casting Algorithm

b) System Development

The system development can only be started after the design works were completed. During this phase, a step-by-step development and implementation would be performed on the system. The activities involved are developing an application system. Table 1 shows the description of the user interface design of the proposed game.

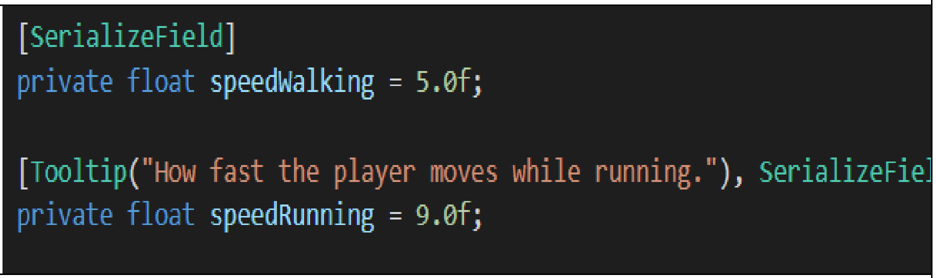
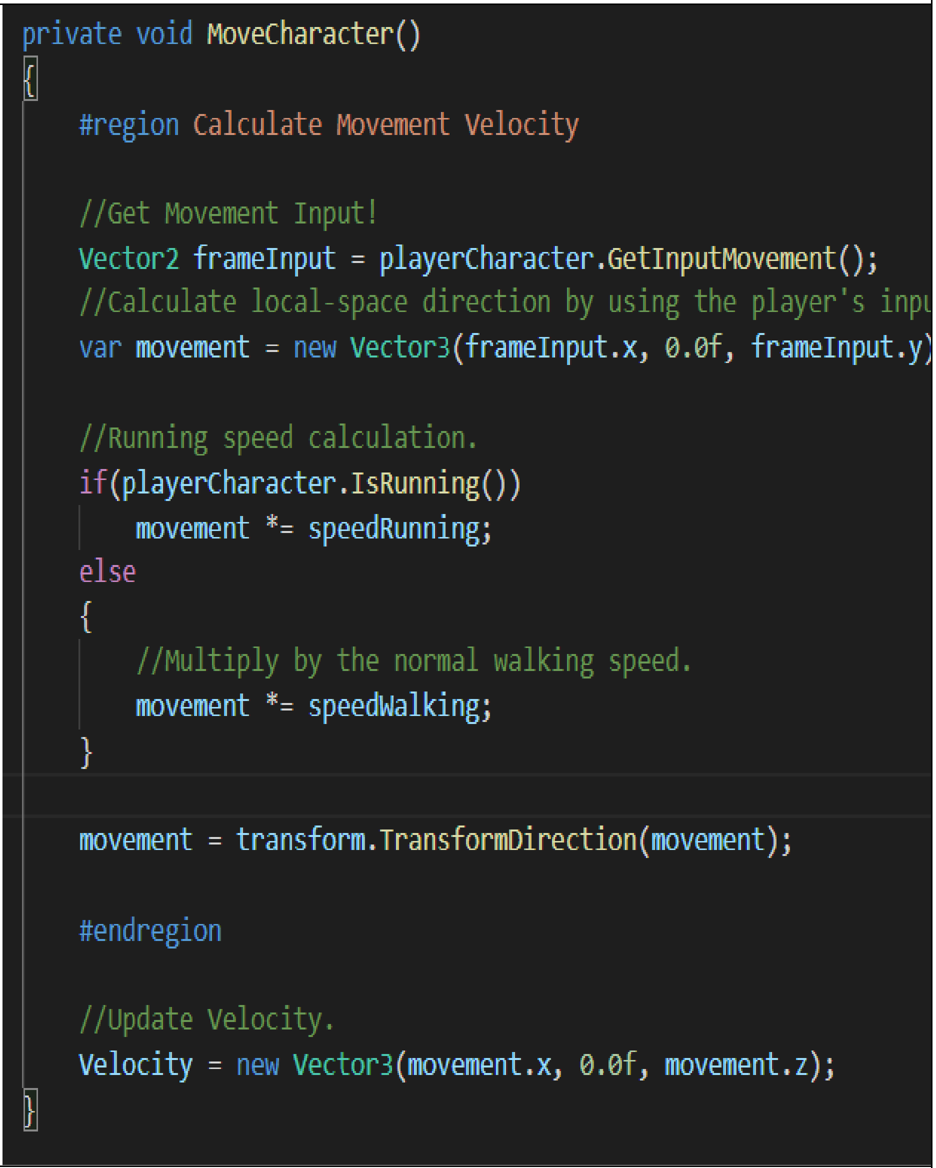
Table 1
Final version of the user interface description

Inter face	Description
	After pressing play, the player will respawn at this area, which is where they started from when they pressed play.
	This is where the player leaves the spawn room and begins their trip. The player must pick whether or not to go forward or to the right room in the building in this section.
	This is the point at which players stop progressing and instead visit the model and material room where they may view the models and materials used in the project.
	This is the point at which the players move forward to the front room. The shooting room is divided into three sections: an explosive barrel, a dummy target, and an explosive gas tank.

c) Functionality Testing

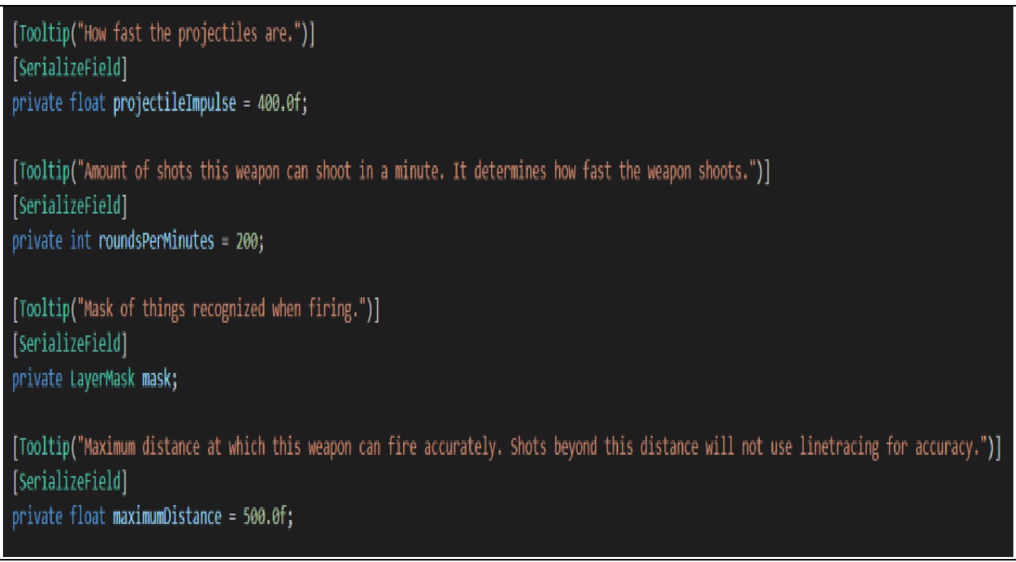
Functionality testing is performed against test cases to validate the requirements of the system and to examine the outcome based on the input that has been specified. Every test case will be tested and explained in each subtopic below:

i. Character Movement

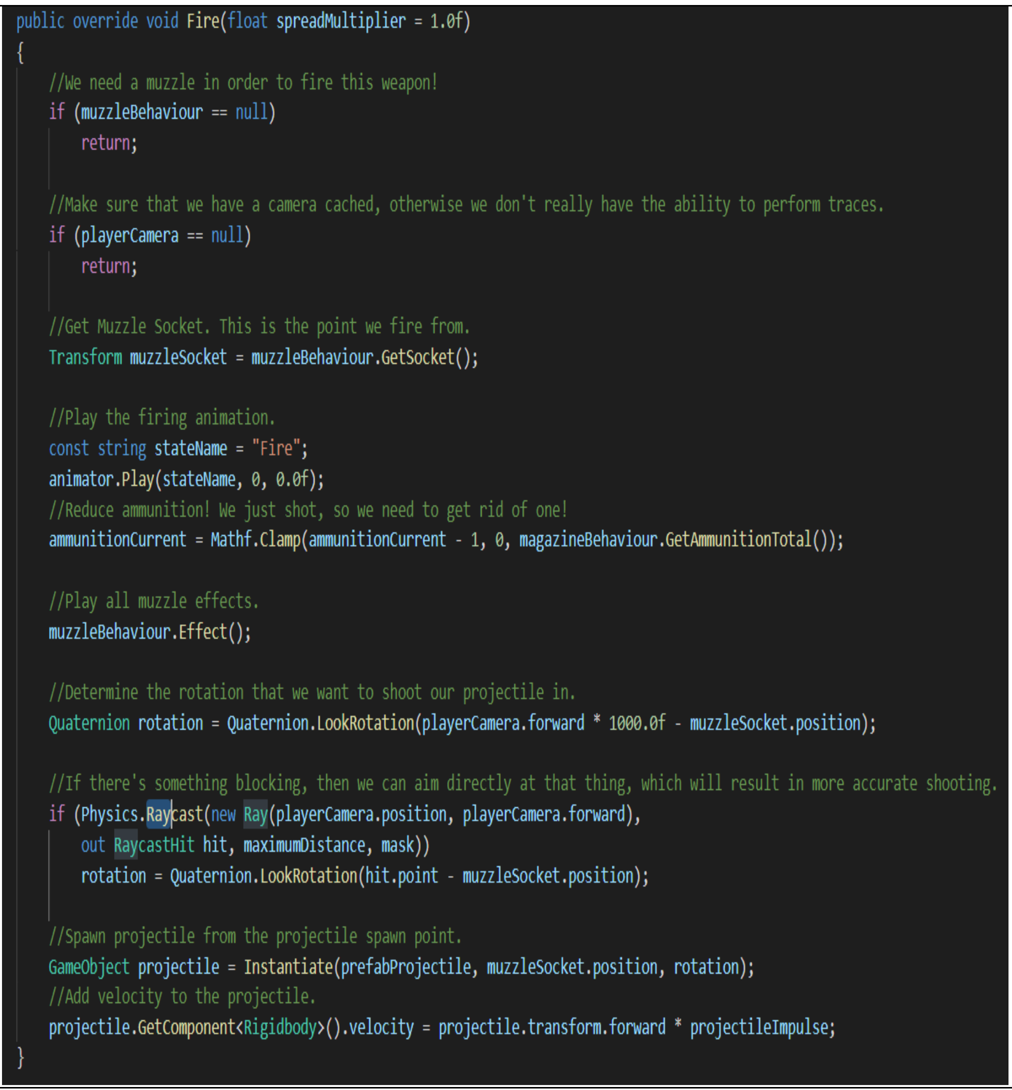
Test Objective	To test the Movement of the character's fluidness.
Potential Test Input	 `[SerializeField]` `private float speedWalking = 5.0f;` `[Tooltip("How fast the player moves while running."), SerializeFiel` `private float speedRunning = 9.0f;`
Expected Test Output	Character speed movement is function as well.
Test Procedure	 `private void MoveCharacter()` `{` `#region Calculate Movement Velocity` `//Get Movement Input!` `Vector2 frameInput = playerCharacter.GetInputMovement();` `//Calculate local-space direction by using the player's inpu` `var movement = new Vector3(frameInput.x, 0.0f, frameInput.y)` `//Running speed calculation.` `if(playerCharacter.IsRunning())` `movement *= speedRunning;` `else` `{` `//Multiply by the normal walking speed.` `movement *= speedWalking;` `}` `movement = transform.TransformDirection(movement);` `#endregion` `//Update Velocity.` `Velocity = new Vector3(movement.x, 0.0f, movement.z);` `}`
Actual Test Result	There are some bug in the process with the making using the current code but have been fixed with movement variable
Testing Result	Movement successfully works

ii. Testing Ray Cast Algorithm

Test Objective	To test the algorithm when player fire the weapon

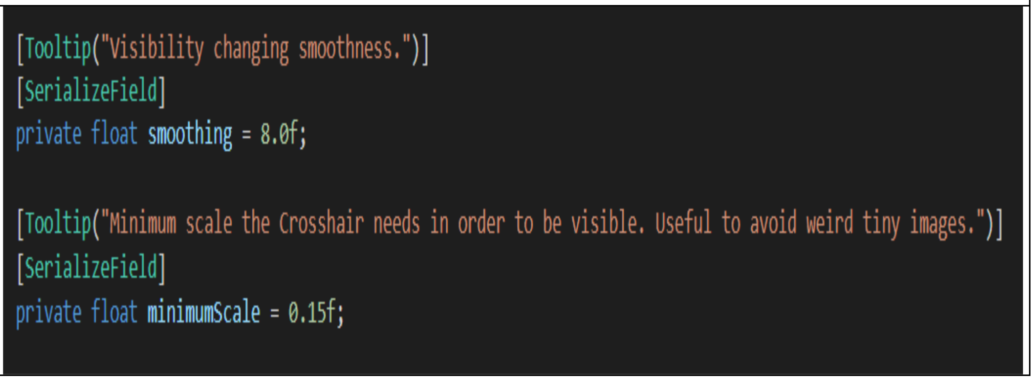
Potential Test Input	
	`[Tooltip("How fast the projectiles are.")]`
	`[SerializeField]`
	`private float projectileImpulse = 400.0f;`
	
	`[Tooltip("Amount of shots this weapon can shoot in a minute. It determines how fast the weapon shoots.")]`
	`[SerializeField]`
	`private int roundsPerMinutes = 200;`
	
	`[Tooltip("Mask of things recognized when firing.")]`
	`[SerializeField]`
	`private LayerMask mask;`
	
	`[Tooltip("Maximum distance at which this weapon can fire accurately. Shots beyond this distance will not use linetracing for accuracy.")]`
	`[SerializeField]`
	`private float maximumDistance = 500.0f;`

Expected Test Output	Data successfully input into the code

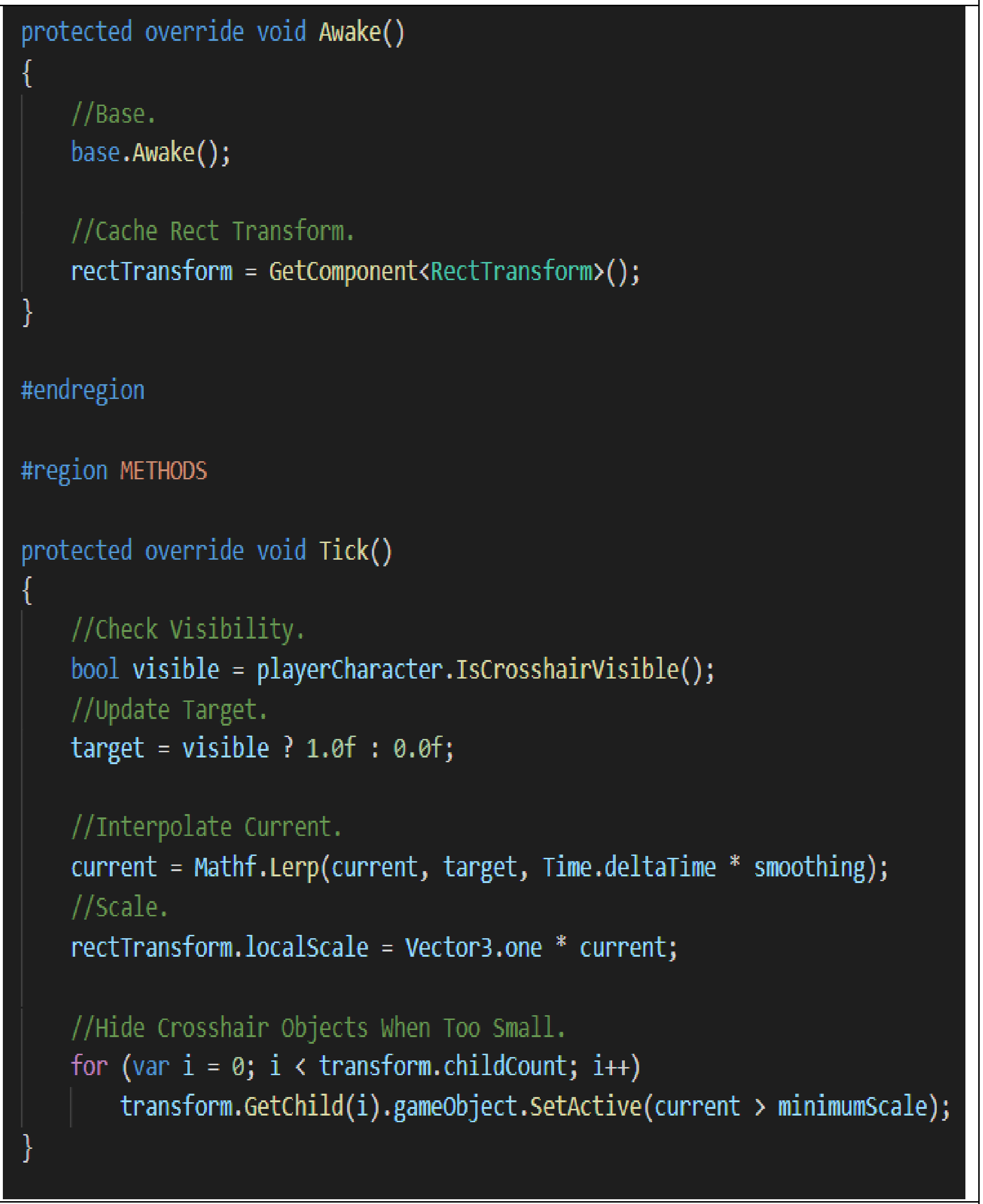
Test Procedure	
Actual Test Result	First when data has been input, the ray casting cannot be detected but with tweak and using physics function, Ray cast detect normally
Testing Result	Ray casting of the bullet successfully function

```
public override void Fire(float spreadMultiplier = 1.0f)
{
    //We need a muzzle in order to fire this weapon!
    if (muzzleBehaviour == null)
        return;

    //Make sure that we have a camera cached, otherwise we don't really have the ability to perform traces.
    if (playerCamera == null)
        return;

    //Get Muzzle Socket. This is the point we fire from.
    Transform muzzleSocket = muzzleBehaviour.GetSocket();

    //Play the firing animation.
    const string stateName = "Fire";
    animator.Play(stateName, 0, 0.0f);
    //Reduce ammunition! We just shot, so we need to get rid of one!
    ammunitionCurrent = Mathf.Clamp(ammunitionCurrent - 1, 0, magazineBehaviour.GetAmmunitionTotal());

    //Play all muzzle effects.
    muzzleBehaviour.Effect();

    //Determine the rotation that we want to shoot our projectile in.
    Quaternion rotation = Quaternion.LookRotation(playerCamera.forward * 1000.0f - muzzleSocket.position);

    //If there's something blocking, then we can aim directly at that thing, which will result in more accurate shooting.
    if (Physics.Raycast(new Ray(playerCamera.position, playerCamera.forward),
        out RaycastHit hit, maximumDistance, mask))
        rotation = Quaternion.LookRotation(hit.point - muzzleSocket.position);

    //Spawn projectile from the projectile spawn point.
    GameObject projectile = Instantiate(prefabProjectile, muzzleSocket.position, rotation);
    //Add velocity to the projectile.
    projectile.GetComponent<Rigidbody>().velocity = projectile.transform.forward * projectileImpulse;
}
```

iii. Testing Crosshair Implementation

Test Objective	To test the crosshair of the weapon for ray casting development
Potential Test Input	

```
[Tooltip("Visibility changing smoothness.")]
[SerializeField]
private float smoothing = 8.0f;

[Tooltip("Minimum scale the Crosshair needs in order to be visible. Useful to avoid weird tiny images.")]
[SerializeField]
private float minimumScale = 0.15f;
```

Expected Test Output	Data of the crosshair successfully input
Test Procedure	 (code below)
Actual Test Result	The crosshair cannot work properly because it was invisible, after using IsCrosshairVariable function to make the crosshair visible
Testing Result	Crosshair functioning as well

```
protected override void Awake()
{
    //Base.
    base.Awake();

    //Cache Rect Transform.
    rectTransform = GetComponent<RectTransform>();
}

#endregion

#region METHODS

protected override void Tick()
{
    //Check Visibility.
    bool visible = playerCharacter.IsCrosshairVisible();
    //Update Target.
    target = visible ? 1.0f : 0.0f;

    //Interpolate Current.
    current = Mathf.Lerp(current, target, Time.deltaTime * smoothing);
    //Scale.
    rectTransform.localScale = Vector3.one * current;

    //Hide Crosshair Objects When Too Small.
    for (var i = 0; i < transform.childCount; i++)
        transform.GetChild(i).gameObject.SetActive(current > minimumScale);
}
```

Conclusion

In conclusion, the first goal of the project was to design a video game prototype that could be used to test the selected ray casting algorithm within the video game. This was the method by which the design for this project was created, using a variety of models and materials. The project will make use of a 3D model to ensure that the design is visually appealing and of high quality, as well as to make it more realistic for users to play the game while participating in it.

Additional materials were introduced to the model, which now appears not only in black and white but also in a variety of colours, which is intended to encourage participants in the gaming experience. To achieve the second objective of the project, which is to implement the ray casting method based on how the targeting hit box bullet shot on an object in the prototype game was satisfied, the following steps were taken: The ray casting implementation is used to ensure that the hitbox bullets are delivered to the target dummy and that the target dummy is functional. The codes for the ray casting algorithm, which is used to fire the weapon's shooting bullet, are demonstrated. The author of the project is employing ray casting because the algorithm's implementation is straightforward. From the author's perspective, many shooting games have made use of the technique because it was simple to incorporate into the game and to support the statement, and Jung (2019) agreed that ray casting is one of the more straightforward techniques to incorporate into the creation of shooting games. Finally, the third goal is to put the proposed targeting game through its paces to see how well it works. Everything else on the project has been done apart from the gyroscope implementation. Other features of the project system, such as character movement, weapon functioning, bullet hitbox using ray casting, and many others, have been successfully incorporated. In general, it appears that the project's operation is proceeding smoothly and successfully. This project can be further explored by implementing it through a mobile platform or in a learning-based game application (Dahalan et al., 2022).

References

Anik, M. A. I., Hassan, M., Mahmud, H., & Hasan, M. K. (2017). Activity recognition of a badminton game through accelerometer and gyroscope. 19th International Conference on Computer and Information Technology, ICCIT

Computer Hope. (2019). Ray Casting. Retrieved from https://www.computerhope.com/jargon/r/ray-casting.htm

Dahalan, N. M., Dahlan, A., Abdullah, Z., & Halim, W. A. F. W. A. (2022). J -Wmyh Mobile Game-Based

Learning Application. International Journal of Academic Research in Progressive Education and Development, 11(4), 29–39

Game Development Lifecycle Models. (2021). StudyTonight.

Jain, S. (2017). Game Development Life Cycle. Retrieved from https://www.linkedin.com/pulse/game-development-life-cycle-sumit-jain

Jung, T. (2019). How Do Bullets Work in Video Games? Retrieved from https://www.gamasutra.com/blogs/TristanJung/20191206/355250/How_Do_Bullets_Work_in_Video_Games.php

Stenfors, D. (2019). Low Poly FPS Pack. Retrieved from https://assetstore.unity.com/packages/3d/props/weapons/low-poly-fps-pack-free-sample-144839#releases

Techopedia. (2019). Ray Casting. Retrieved from https://www.techopedia.com/definition/21614/ray-casting

Unity (2019). 2.5D Bullet hitbox detection for distant objects. Retrieved from https://answers.unity.com/questions/1411150/25d-bullet-hitbox-detection-for-distant-objects.html