Robot Shooter Game

Group - 14

Ajay Krishna Vadlavalli (11554489)

Tejaswini Aranagu (11556194)

Sushmitha Reddy Chintalapani (11554373)

INFO 5900 - Special Problems (Section 021)

Dr. Sharad Sharma

Abstract:

This project intends to create a virtual reality (VR) environment with the purpose of simulating a particular situation, such as the behavior of evacuees, the occurrence of potentially dangerous occurrences, or the renovation of a structure. Users who are interested in experiencing and interacting with a 3D setting are the focus of the virtual reality implementation. Unity 3D and the programming framework C# were used throughout the development of the task. 3ds Max or Google Sketchup were used to produce the various 3D models and graphics. A variety of elements, including geometry, textures, animations, functionality, and interaction, are included in the virtual reality experience. The inclusion of music, moving objects, and artificial intelligence entities all contribute to an interaction that is more realistic. The program has the potential to be useful in a variety of real-world situations, including pleasure, instruction, and learning.

The Robot Shooter Game places users in a fantastical universe where they must defend themselves from hostile robots. Unity, a well-known game engine, was used to bring the game's breathtaking visuals and genuine gameplay elements to life.

The ever-increasing need for interactive video games that make full use of the potential offered by virtual reality equipment is the impetus behind the need for the development of this endeavor. The goal of this endeavor is to deliver a one-of-a-kind gaming experience that immerses the player in a simulated environment and, at the same time, presents them with a game that is both exciting and difficult to play.

The scene of the augmented reality experience depicts a future metropolis with towering skyscrapers, moving idols, and sophisticated robotic foes. The adventure is split up into many stages, each of which has its own unique challenges. The player has access to a wide variety of weapons and other items with which to do battle with the hostile robots.

Players that like immersive, first-person shooting games, especially those who are appreciative of the use of augmented reality (AR) technology, are the intended users of this virtual reality (VR) application. Because the game's mechanics are straightforward and easy to understand, it is accessible to players of a wide range of skill levels.

The primary objective of carrying out this project was to create a virtual reality gaming experience that was not only challenging but also engaging and engrossing. The achievement of the project can be seen in the widespread appeal of the game as well as the positive reception it received from players.

The video game acts as a kind of entertainment by putting the player in a scenario that is set in the future and giving them the opportunity to do combat against mechanical foes. In addition, the game's rapid pace and tough gameplay make it an ideal tool for improving coordination of the hands and eyes, reaction speed, and the ability to make split-second judgments.

In a nutshell, the Robot Shooter Game is an augmented reality video game that offers a thrilling and authentic game play that makes use of several virtual reality technology. The combination of its stunning

visuals, straightforward playing concepts, and taxing gameplay makes it an excellent tool for both amusement and the enhancement of one's abilities.

Introduction:

We have created a "ROBOT SHOOTER GAME" that is a 3D gaming application that is centered on first person controller gameplay. Including the creation of the game's scenario, playing technique, and approved circumstance, as well as the development of the enemy's artificial intelligence sophisticated logic and resource system. In between, the standard Unity 3d engine is used to provide the primary implementation technologies of the game.

For the development of this game, we made use of the scripting languages C# and JavaScript, which makes up the server side of the Unity game engine. The Unity game engine has an additional object-based programming framework. The implementation of real-time graphics, a physics engine, network support, and sound effects will all be covered in the dissertation.

The objective of the project is to create an exciting and engaging robot shooting game by making use of the Unity game engine. This project provides an overview of the proposed environment, including its many components and the features that have been successfully implemented.

The action of the video game takes place in a future metropolis, which has lofty skyscrapers, powerful robotic foes, and a wide selection of weapons and upgrades for the player to choose from. The newest virtual reality tech is used in the production of this game in order to provide the player with an experience that is both spot-on realistic and interactive.

The purpose of this project is to develop a virtual reality (VR) setting that can imitate a particular event, such as the burning of a structure. The surroundings is comprised of a variety of components, including geometry, textures, animations, and interactive elements. The virtual reality (VR) application may be utilized for educational and training reasons, giving users with a feeling that is both deeper yet realistic. The planned world may be explored using either a first-person or third-person controller, and user input is used to activate interactions inside the scene.

Implementation:

The game was developed using the game engine known as Unity, which enabled high-quality graphics and authentic gameplay elements. Additionally, 3D models and textures were created using either 3ds Max or Google Sketchup. The following are some of the characteristics included in the game:

Player Movement:

The player can maneuver across the world by making use of either the computer keyboard or the gamepad controller.

Weapons:

A variety of firearms, such as a pistol or a rifle, are available for the player to choose from.

Enemies:

The artificial intelligence (AI) technologies that have been programmed into adversary robots give them the ability to navigate their environment and pursue the player.

Health System:

When the player takes damage because of being shot at by other foes, their health bar will decrease.

Power-Ups:

The player can pick up power-ups that give them an increase in either their assault power or their lifetime.

Level Design:

The game is broken up into many stages, each of which has its own unique set of challenges and difficulties.

Sound Effects:

The video game has a diverse selection of sound effects, including gunshots, explosions, and robot growls.

AI Implementation:

By doing the use of a user choices, the setting allows for the implementation of various conduct, such as self-centered or selfless conduct.

Interface elements:

The user may initiate conversations with the help of the interface by using menu items like icons.

Users can have an enhanced and full immersion thanks to the virtual reality environment, which is excellent for educational and training reasons. Because it enables users to experience and interact with a 3D environment, augmented reality is the technology that should be used for this endeavor. Conventional 2D techniques provide a less interactive encounter than the use of virtual reality, which is why virtual reality is the technology that should be used. The project ran into a few issues and shortfalls, such as restricted hardware compatibility or limited execution of AI behavior. These issues and weaknesses might potentially be rectified in further work on the project.

The game has been researched and inspected to ensure that it will give a gaming experience that is both efficient and pleasant.

Acknowledgement:

Unity 3D, the programming language C#, 3ds Max or Google Sketchup for 3D modeling, and a variety of additional software as well as hardware components that are often used in virtual reality (VR)

applications are some of the resources that have been included into this project. The citations section includes annotations that are pertinent to the links leading to other sources.

References:

Unity 3D, C#, 3ds Max or Google Sketchup for 3D modeling, and a variety of other software and hardware elements that are often used in virtual reality apps are some of the primary tools that were utilized in the production of the game.

Goal and Objectives:

The objective of the Robot Shooter project is to develop a virtual setting in which players are tasked with defending themselves against an onslaught of dangerous robots. The goal of this project is to create and construct a game in which the player takes control of a robot and engages in combat with other robots while traversing a 3D world.

- To accomplish this aim, the following subgoals have been established:
- To give a gaming experience that is both tough and entertaining, you need create an immersive virtual world that has accurate representations of buildings, terrain, and items.
- Increasing the challenge of the game by adding survival elements such as managing one's health and ammo would give players the impression that they are engaged in a struggle to stay alive.
- Programming the antagonistic robots to act in a way that is both difficult and clever will force players to use both strategy and skill to prevail over them.
- Build the game with a user interface and controls that are easy to understand for as many people as possible, regardless of whether they have previous experience playing video games.
- Using music, sound effects, and visual feedback, creates a gaming experience that is not only fun but also rewarding for the player.
- Conceive of the game's overall concept, which includes creating the overall narrative of the game, designing the robots, and establishing the gaming mechanics.
- Construct a Three-Dimensional Environment: To plan and create a three-dimensional setting for the game. This covers the creation of the landscape, the buildings, and any other assets that will be used in the game.
- Create the playable characters for the game: The third mission is to develop the game characters, such as the player's robot and the robots that the player will face off against. Creating intricate 3D models, texturing, and animating those models are all part of this process.
- Create the game's gameplay mechanics: The fourth goal is to create the game's gameplay mechanics, which includes the controls for the player's robot, the weapons and abilities that may be employed, and the rules for fighting.
- Putting the game logic into action, the fifth goal is to put the game logic into action. This includes the artificial intelligence that controls the adversarial robots, the scoring system, and any additional game regulations.
- The last step is to put the game through its pace and make any necessary adjustments depending on the responses received from players after conducting thorough playtesting. This

includes resolving any errors that have been found, altering the game's balance, and making any necessary adjustments to the gameplay mechanics.

Modeling:

The virtual setting for the robot shooter game is envisioned to be a post-apocalyptic metropolis, complete with derelict structures, streets littered with debris, and threatening robots hiding around every turn. The setting will be created to evoke feelings of hopelessness and peril, with buildings that are falling apart, and rubbish strewn throughout the countryside.

Realistic and immersive geometry for the area will be built, and a wide range of textures and animations will be employed to bring the setting to life. The online setting will feature comprehensive models of structures, furnishings, home essential objects, and landscape aspects. The goal of these models is to generate a feeling of realism and immersion for the player inside the game world.

The program will be intended to be user-friendly and intuitive to explore and engage with. This will enable players to navigate about the area using normal gaming controls, such as the WASD keys for movement and the mouse for shooting. The difficulty of the game will be purposefully increased, and gamers will be expected to make use of both strategy and talent in order to prevail over the mechanical foes.

The 3D models for the setting will be created utilizing either 3ds Max or Google Sketchup, and then the world will be brought to life via the addition of intricate coloring and movement.

Programming:

Unity 3D and the C# programming language will be used throughout the development process for the robot shooting game. During the programming phase, we will create artificial intelligence for the antagonistic robots, implement the survival mechanisms, as well as construct the user interface and control scheme.

The artificial intelligence that controls the adversarial robots will be made to be both tough and sophisticated, using a variety of strategies to win against the player. The complexity of the game will be increased to give players the impression that they are actually engaged in a struggle for existence with the addition of survival mechanisms such as managing their health and ammo supplies.

The user interface and controls will be intended to be straightforward and simple to use, enabling players to pick up the game and become proficient at it in a short amount of time. To make the gameplay more interesting and satisfying for the player, we will be adding some music, sound effects, and visual feedback.

The health management system of the game, which can be seen in action in the images that follow, monitors the player's current health condition, reacts to the player's activities, and causes the appropriate adjustments to be made to the game's setting.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerDetector : MonoBehaviour
{
    public GameOverManager gameOverManager;

    private void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Enemy" && !other.isTrigger)
        {
            float enemyDistance = Vector3.Distance(transform.position, other.transform.position);
            gameOverManager.ShowWarning(enemyDistance);
        }
    }
}
```

```
    void Death()
    {
        isDead = true;

        playerShooting.DisableEffects ();

        //mentrigger animasi Die
        anim.SetTrigger("Die");

        //Memainkan suara ketika mati
        playerAudio.clip = deathClip;
        playerAudio.Play();

        //mematikan script player movement
        playerMovement.enabled = false;

        playerShooting.enabled = false;
    }

    public void RestartLevel()
    {
        //meload ulang scene dengan index 0 pada build setting
        SceneManager.LoadScene(0);
    }
```

```
// fungsi untuk menambah nyawa
public void Healing()
{
    //mengurangi health
    int newHealth = currentHealth + 40;
    if (newHealth >= 100)
    {
        currentHealth = 100;
    }
    else
    {
        currentHealth = newHealth;
    }

    //Merubah tampilan dari health slider
    healthSlider.value = currentHealth;
}
```

- The Player Health class is defined with several public variables, including starting Health, current Health, health Slider, damage Image, death Clip, flash Speed, and flash Color.
- The value of the starting Health variable is now set to 100, which represents the player's maximum possible health.
- The value of the player's beginning health is stored in the "starting Health" variable, which is utilized by the current Health variable to maintain track of the player's current health.
- A link to the slide in the user interface (UI) that shows the player's health bar may be found in the health Slider variable.
- Whenever the player sustains damage, the UI picture that corresponds to that damage is loaded into the damage Image variable as a reference.
- The death Clip variable refers to an audio clip that is played once the player is eliminated from the game.
- Whether the damage Image disappears rapidly or slowly is determined by the flash Speed variable.
- Whenever the player requires harm, the hue of the flashing damage image is determined by the flash Color variable.
- The animation, player Audio and player Movement, as well as the player You may get reference to different parts of the player's game object via firing variables, such as the programmer, the audio source, the player's motion script, and the shooting script.
- The is Dead variable is a Boolean flag that is used to keep track of whether the player is dead or alive. Its purpose is to determine if the player is currently playing.

- The damaged variable is a Boolean flag that is used to keep track of whether the player has received damage. It is used to maintain track of whether the player suffered harm.
- When the script is loaded, the Awake () method is the one that gets called. It obtains references to the necessary components and sets the current Health to the value that was present at the beginning of the process.
- During each frame, the Update () method is invoked exactly once. It determines whether or not the player has sustained any damage and then gradually fades out the damage Image if they have.
- When the player sustains damage, the Take Damage () function is the one that is invoked. If the current Health is less than or equal to zero, it plays an audible effect, changes the health Slider, and triggers the Death () function. If the current Health is more than zero, it decreases the current Health by the amount of damage suffered.
- While the player's health is increased, the Healing () method is invoked on the object representing the player. The current Health value is raised by 40 points, or it is reset to 100 if the new health value is higher than the highest value allowed.
- The Death () function is invoked whenever the player is killed off in the game. The death animation and sound effect are played, the player's movement and firing scripts are disabled, and the is Dead flag has been set to the true position.
- When the player needs to restart the current game level, the Restart Level () function is the one that is called. The scene is reloaded with the index 0 in the build settings when you do this.

The following script pictures discuss the amount of time that passes between attacks, as well as the damage that is dealt by each assault. In addition to this, it has private variables for the Animator component, the player Game Object, and the health components that correspond to each of these. In addition, there are Boolean variables for things like player in Range and whether the player has died.

```
//fungsi untuk mendapatkan damage
public void TakeDamage(int amount)
{
    damaged = true;

    //mengurangi health
    currentHealth -= amount;

    
    healthSlider.value = currentHealth;


    playerAudio.Play();


    if (currentHealth <= 0 && !isDead)
    {
        Death();
    }
}
```

```
    void Awake()
    {
        //Cari game object with tag player
        player = GameObject.FindGameObjectWithTag("Player").transform;

        //Mendapatkan componen reference
        playerHealth = player.GetComponent<PlayerHealth>();
        enemyHealth = GetComponent<EnemyHealth>();
        nav = GetComponent<UnityEngine.AI.NavMeshAgent>();
    }


    void Update()
    {
        //Pindah ke player position
        if (enemyHealth.currentHealth > 0 && playerHealth.currentHealth > 0)
        {
            nav.SetDestination(player.position);
        }
        else //Stop moving
        {
            nav.enabled = false;
        }
    }
}
```

- During the Awake () method, the script looks for the player Game Object by utilizing the tag on the item, and then it sets the Player Health part, the Animation component, and the Enemy Health component to the variables that correspond to those components.
- Additionally, the script includes two functions known as On Trigger Enter and On Trigger Exit. These methods are called whenever the player moves into or out of the adversary's attack range, respectively. These routines are responsible for setting the player In Range Boolean variable in the appropriate manner.
- The timer object has its value increased by Time every time the Update () method is used. Delta Time per frame. The Attack () method is invoked if the timer has elapsed for a period that is longer than the time Between Attacks value, the player is in spectrum, if the adversary is still alive. When the player's health reaches 0 or below, the Player Dead trigger, which is in the antagonist's Animator, is activated.
- The timer is restarted when the Attack() function is called, and the Take Damage() method in the player's Player Health component is contacted with the attack Damage value as the argument.

- The enemy's attack behavior is controlled by the script, and it does so by checking to see whether the player is within range, and if they are, assaulting the player. In addition to this, it determines whether the player has died and then changes the Animator appropriately.

The screen photos that follow provide an explanation of the shooter1, including how his attacks work and how he chooses which foes to target.

```
public void DisableEffects()
{
    //disable line renderer
    gunLine.enabled = false;

    //disable light
    gunLight.enabled = false;
}


public void Shoot()
{
    timer = 0f;

    //Play audio
    gunAudio.Play();

    //enable Light
    gunLight.enabled = true;

    //Play gun particle
    gunParticles.Stop();
    gunParticles.Play();

    //enable Line renderer dan set first position
    gunLine.enabled = true;
    gunLine.SetPosition(0, transform.position);
```

```
void Update()
{

    timer += Time.deltaTime;

    if (Input.GetButton("Fire1") && timer >= timeBetweenBullets && Time.timeScale != 0)
    {
        Shoot();
    }

    if (timer >= timeBetweenBullets * effectsDisplayTime)
    {
        DisableEffects();
    }
}
```

```
    gunParticles.Stop();
    gunParticles.Play();

    //enable Line renderer dan set first position
    gunLine.enabled = true;
    gunLine.SetPosition(0, transform.position);

    //Set posisi ray shoot dan direction
    shootRay.origin = transform.position;
    shootRay.direction = transform.forward;

    //Lakukan raycast jika mendeteksi id nemy hit apapun
    if (Physics.Raycast(shootRay, out shootHit, range, shootableMask))
    {
        //Lakukan raycast hit hace component Enemyhealth
        EnemyHealth enemyHealth = shootHit.collider.GetComponent<EnemyHealth>();

        if (enemyHealth != null)
        {
            //Lakukan Take Damage
            enemyHealth.TakeDamage(damagePerShot, shootHit.point);
        }

        //Set line end position ke hit position
        gunLine.SetPosition(1, shootHit.point);
    }
    else
    {
        //set line end position ke range freom barrel
        gunLine.SetPosition(1, shootRay.origin + shootRay.direction * range);
    }
```

- Damage Per Shot is an integer that represents the amount of damage done by the weapon with each shot.

- 'Time Between Bullets' is a float that represents the amount of time that passes between each shot fired by the weapon, range is a float that represents the weapon's effective firing distance.
- In addition, these private variables are included inside the script: timer: a float that represents the amount of time that has passed since the last shot was fired a Ray object that represents the ray that is utilized for firing; referred to as "shoot Ray."
- A Ray cast Hit object that represents the hit of the ray cast is referred to as shoot Hit.
- Shootable Mask is an integer that represents the layer mask that was used for shooting.
- gun Particles are an instance of the Particle System class, and they represent the weapon's particle system in its entirety.
- A Line Renderer object that represents the line renderer that is utilized for the weapon is referred to as the gun Line.
- gun Audio is an object of type Audio Source that represents the audio source that is utilized by the firearm.
- a Light object that represents the light that is utilized for the weapon, denoted by the variable gun Light.
- Effects Display Time is a float that represents the length of time that the weapon's visual effects will be active.

The script serves two different purposes:

- Disable Effects () is a public method that turns off the line renderer as well as the light that is attached to the weapon.
- A public function that manages the firing mechanics is referred to as Shoot (). The timer is reset, the gun audio is played, the gun light and particles are enabled, and the beginning point of the line renderer is set when it is called. After then, it sends out a beam from the location of the player in the direction that they are facing. The Enemy Health component of the Game Object is retrieved from the memory of the ray if it strikes a Game Object that has the Shootable layer.
- Object is sent as a parameter to the Take Damage () method of the target object, along with the damage Per Shot value and the shoot Hit point. After that, it adjusts the line renderer's end location such that it ends at the shoot Hit point. If the ray does not strike anything, the maximum possible range of the weapon is used to determine the end location of the line renderer. At last, it adjusts the effects Display Time so that it is a proportion of the time Between Bullets value. This is done so that the visual effects can be turned off after a predetermined amount of time has elapsed.
- The Update () method is in charge of analyzing the input provided by the player and, if required, invoking the Shoot () function. After a predetermined amount of time has passed, it also makes a call to the disabled Effects () function to turn off the weapon's visual effects.

Functionality:

The instructions that are provided below provide an in-depth explanation of how the functionality of the game was implemented:

Vision: Providing more in-depth information about the project via the use of textured models and 3D models. The game has high-quality graphics and 3D models, which work together to provide an atmosphere that is realistic and immersive for the player.

The whole experience of playing the game is improved by the inclusion of a variety of sound effects as well as music in the background. The presence of ambient noises, such as the sound of the wind or the hum of machinery, contributes to the creation of an atmosphere that is more immersive.

The game has several animated elements, including the player's weapon as well as the robots that the player must defeat. The player will get a sense of increased engagement and dynamism in the game because of the animations.

Interactivity is included throughout the game in the form of several different events that are triggered by the player, such as picking up weapons or pressing buttons to access doors. These events provide an additional dimension of engagement to the game, making it more interesting and exciting for the player to participate in.

Characters and Avatars: The game has animated agents that have behavior that allows them to follow a course. While the player is free to move anywhere within the environment using the keyboard controls, the player's foes, which are robots, are moving toward them.

Sensors: The game incorporates a wide variety of sensors, including proximity sensors, which cause hostile robots to spawn in the area around the player if they detect that the player is near them. Touch sensors are used to engage buttons and other in-game elements, whilst time sensors are employed to activate events that occur at certain periods during the game.

Player: The environment is augmented with either a first-person controller or a third-person controller so that the player may exercise control over their character.

Implementation of AI: The game has AI capability that allows the player to direct the actions of the hostile robots. Elements of the user interface: The game includes a user interface with menu items such as buttons that allow the player to interact with the environment and select different options. The AI is implemented using a variety of behaviors, such as learning or adaptive behaviors, which are selected by the user through a menu.

In conclusion, a variety of elements were included in the development of the Robot Shooter Game so that it could provide players a gaming experience that was both immersive and interesting. The incorporation of different textures, sounds, animations, interactive elements, characters, sensors, and functionalities provided by artificial intelligence all work together to provide a dynamic and difficult gaming experience. The components of the user interface make it simple for players to traverse the game and pick various choices, which contributes to an improved experience for the user.

The game includes a variety of agents and avatars, each of which may be customized to behave in a certain manner inside the virtual world. Pathfinding, collision detection, firing, and dodging are only some of the activities that have been programmed into the agents.

The game's agents and avatars are really robots that have been designed to react in a variety of unique ways depending on the part they play in the game. Pathfinding behavior is shown by the agent that is controlled by the player to explore the surroundings of the game and locate the hostile robots. In order to engage the player in a fight, the antagonistic robots display a variety of behaviors, including seeking, fleeing, and attacking actions. In addition, the agents and avatars feature collision detection and avoidance behaviors, which allow them to navigate the game world while avoiding obstacles and other agents and avatars.

Because this game is set in a virtual reality environment, the developers were able to use textures and 3D models to give players more information about the in-game world. The music and background noises that are provided by the sound system serve to further submerge the player in the setting of the game. In addition to that, there are at least three animated items in the game, such as doors, moving platforms, and elevators. There are five user-triggered events that contribute to the game's interactivity. Some examples of these events include opening doors, operating elevators, and firing opposing robots.

The game makes use of a variety of sensors, including proximity sensors, which determine when the player is near an item or agent, and touch sensors, which determine when the player is in contact with an object or agent. A time sensor is also included in the game, and it will activate certain events once a certain period has passed.

The user interface of the game consists of many menu items, such as buttons for beginning and halting the game, selecting different levels, and selecting other game parameters.

This program is beneficial because it offers a gaming experience that is both immersive and engaging. This helps users improve their hand-eye coordination, decision-making abilities, and reaction speed. The use of virtual reality technology is suitable for this project because it creates a setting that is both realistic and immersive, which in turn improves the game experience for the participant.

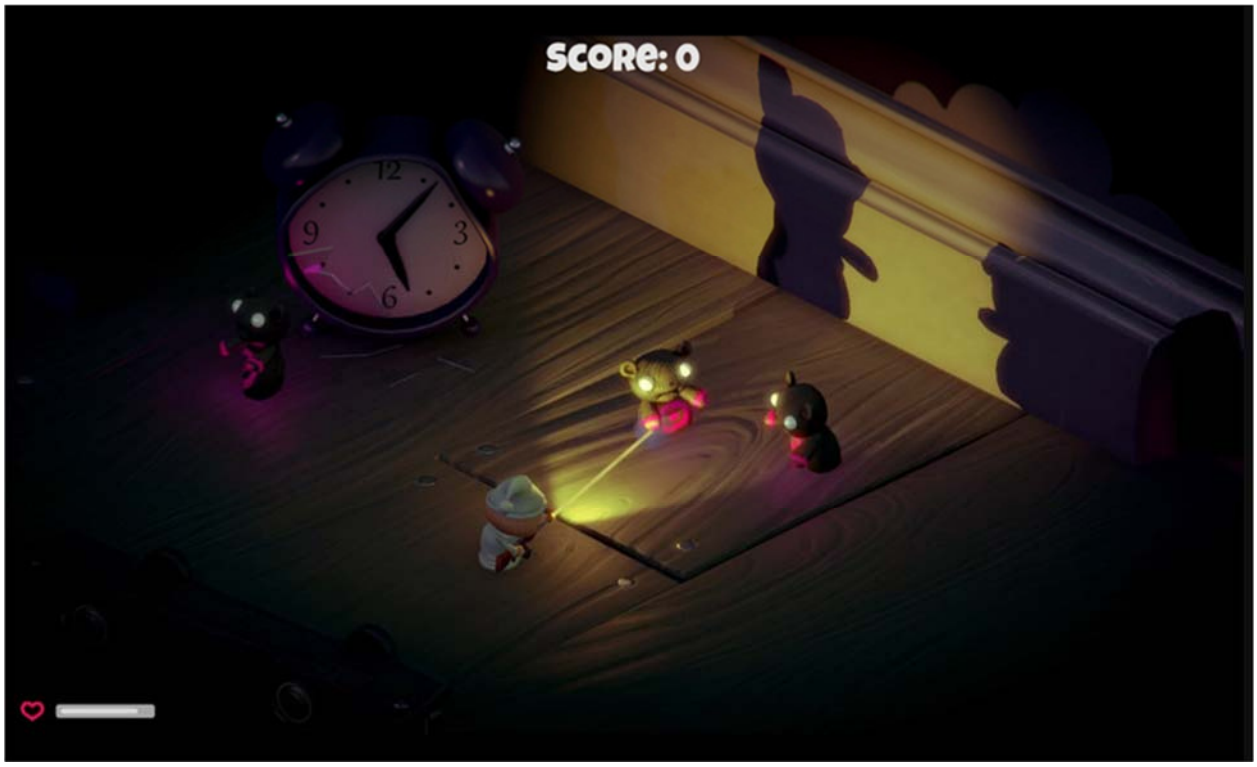
During the process of creating the game, we ran into a few issues, such as performance problems brought on by the complexity of the game environment and the actions taken by the agents and avatars. Work that is done in the future might concentrate on improving the game's performance and introducing more complex AI characteristics, such as learning and adaptive behaviors, to make the game more difficult and interesting.

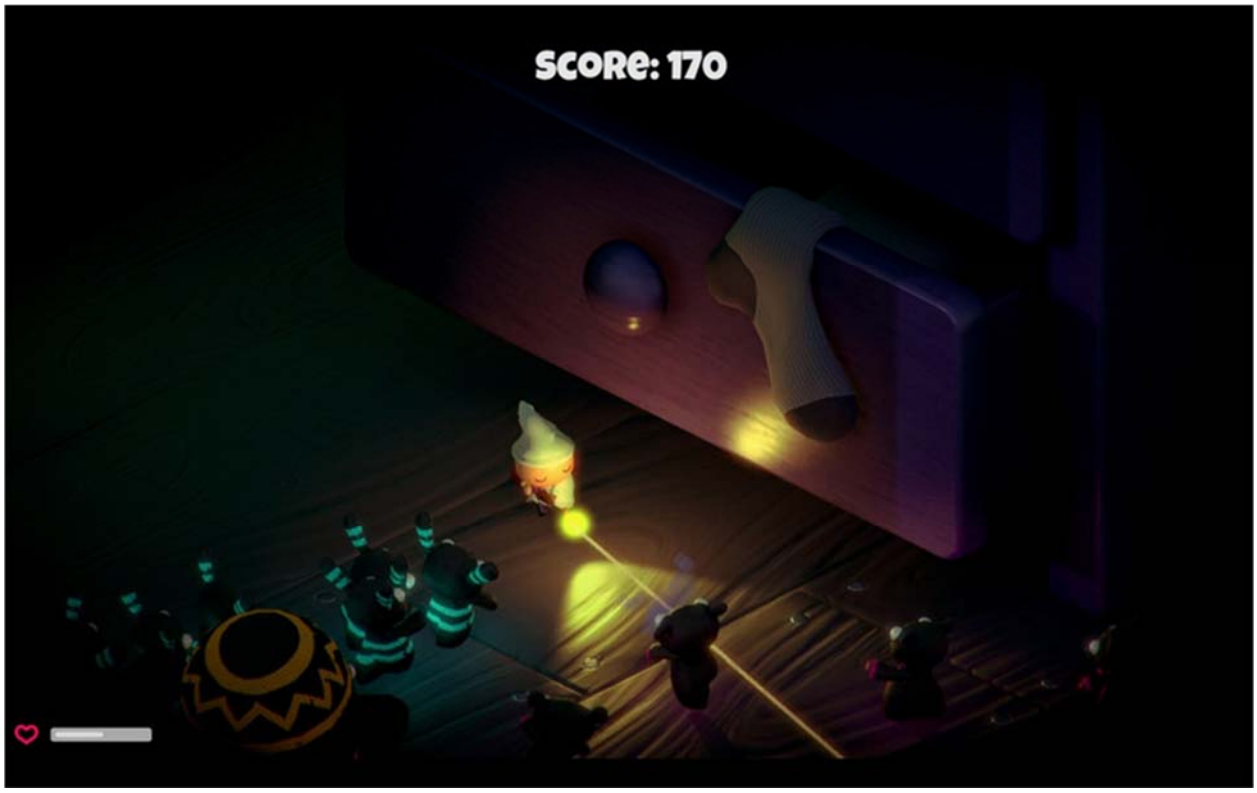
In conclusion, the virtual reality gaming experience provided by the Robot Shooter Game, which was designed using Unity 3D and the C# programming language, was successful in providing players with an exciting and immersive environment to play in. The game is a wonderful instrument for both amusement and the development of skills due to its stunning visuals, straightforward gameplay concepts, and difficult gameplay. Players who like immersive first-person shooting games are likely to find this one appealing since it makes use of augmented reality technologies. The player is immersed in the experience thanks, in part, to the many kinds of sensors and interactive elements that have been introduced to the game.

Despite this, the game still has a few issues that need to be addressed, such as the need for a more sophisticated artificial intelligence, more levels, and other weaponry. These are things that can be worked on in the future to make the game better and provide the player with a better experience.

In general, the Robot Shooter Game has shown that virtual reality technology has the potential to provide an exciting and engaging gaming experience that players of all talent levels may enjoy. This promise has been proved by the game. Because of its popularity, we now know that technology based on virtual reality can be utilized to create immersive gaming experiences that provide more than simply amusement.

The Robot is seen below in a few screenshots engaging in combat with various foes:





References:

Some of the resources were taken from and motivated by existing robot shooter games and documentation for the Unity game engine.

Following are the important resources and inspiration:

- Unity Technologies (https://unity.com/) is a game development company.
- Unreal Engine (https://www.unrealengine.com/) is a game engine.
- Stack Exchange for Game Development (https://gamedev.stackexchange.com/)
- Game Developer (https://www.gamedeveloper.com/) is a gaming website.