Toy Takeover

Technical Design Document

v 1.0

By Paul King

Change Log

15/09/2021	V0.1	Started filling in all of the template sections
18/09/2021	v0.2	Filled in Asset list including the Unity C# scripts and some of the game objects and prefabs.
30/09/2021	v0.3	Filled in the rest of the GameObject/prefabs section, inserted two diagrams and updated the table of contents.
12/10/2021	v0.4	Updated the title page image. Made updates to the mechanic's section. Fixed some spelling and grammar errors Added some additional solid objects to the asset list. Add EnemySound to scripts section
15/10/2021	v0.5	Updated Ammo in the items section Added Range enemy information to Artificial intelligence section
18/10/2021	v0.6	Inserted new diagrams for ranged enemy A.I. Updated Information in the enemy A.I. Updated weapons sections to match new weapon names Updated enemy scripts in the "Scripts" section Added "Taking Damage" paragraph to mechanics
	v1.0	Converted document into PDF.

Table of Contents

Development Environment

Game Engine: Unity 2021.1.10f1
IDE: Visual Studio 2019
Source Control procedures: Git, Github
Third-Party Libraries: UnityEngine
Other Software: Photoshop, Blender

Game Overview

Technical Goals

To create a first-person shooter in Unity running at a smooth frame rate with input from keyboard and mouse as well as from a game controller

Game Objects and Logic

The game will contain a series of game objects created in the Unity Game Engine. Some will be stored in the scene, others will be prefabs which will load at run time.

Player
The main game object will be a player that has an fps controller attached to it. A camera will be attached to the player's transform so that it moves with the player.

Wall
There will be walls located inside the level that no other objects can pass through. Several different 3D models can be treated as walls.

Bullet
This is a projectile prefab that is instantiated whenever the player fires a gun. Once the bullet is fired it will travel forward until it hits an object with a collider on it and then it is destroyed.

Enemy
Enemies will be prefabs that spawn on a spawn point whenever a new enemy wave begins. The enemies

Item
There are item prefabs placed throughout the level. When the player runs into an item it triggers, removing the object from the scene and invoking a function that applies the effect of that item.

Game Flow

When the game loads, the player will be taken to the main menu where they will see a "start game", "options" and "exit" button. After clicking "start game", Unity will load the level scene.

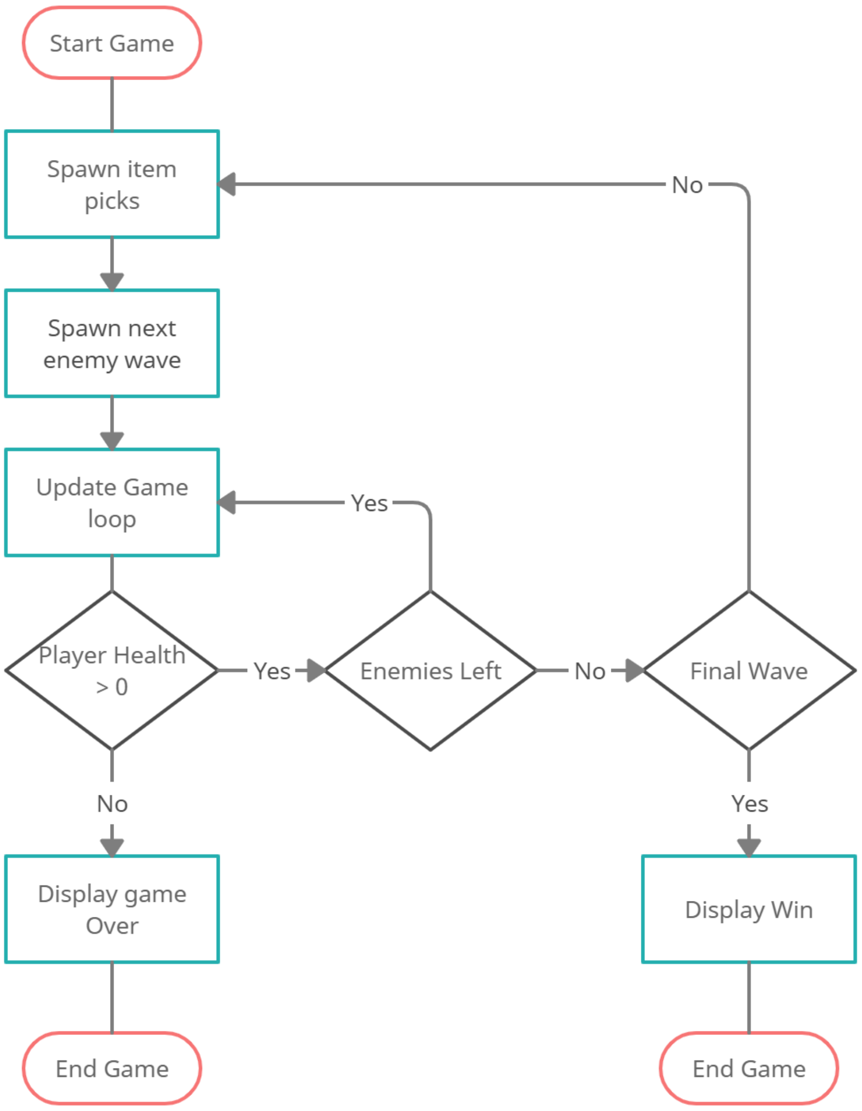


Diagram of Main game loop

While the level scene is running, waves of enemies will appear which the player will have to defeat by destroying all the enemies using the gun weapon. When there are no more enemies alive in the wave the game will load the next enemy wave and spawn all the enemies in it.

After defeating the enemy wave a new series of item pickups will appear at their appropriate location.

Once the player has defeated all enemies waves the game has been won. If the player runs out of health at any point, then he/she dies, and the game has been lost.

Mechanics

While the game is running, there are several mechanics that the player can perform as long as their character is still alive.

Player movement
The first of these mechanics is character movement. The player can move in four different directions: forward, back, left and right. This will be controlled by invoking move() on the ChracterController component. They can also rotate the camera using the mouse or right thumbstick which changes which direction the character is facing.

Firing Weapons
One of the core mechanics is shooting. Whenever the camera rotates, so does the weapon that the player is holding. When the player fires a projectile will move in a straight line from the weapon until it hits something. If it is a rapid-firing weapon it will fire continuously with a new bullet spawning over a time interval. If the player uses a laser weapon a ray will be cast from the gun to the first thing it hits. This will be drawn on screen using the LineRenderer.

Jumping & flying
Another core mechanic is lifting the player up into the air. This can be done in two ways; by jumping and by flying around in a jetpack. When the player presses the jump button the player will jump. If the player presses the jetpack button and there is no fuel left in the jetpack then the jetpack will be used

When using the jetpack, the player will continue to hover around in the air until the key/button is released or they run out of fuel. When this happens, they will fall until colliding with the ground or a platform. the movement speed when moving around on the jetpack will be based on lift, thrust and weight variables.

Taking damage
The player will have a health meter which they must keep above zero otherwise they die Whenever the player collides with an enemy, damage will be dealt. The player will also take damage if a melee enemy hits the player while in attack range or a projectile fired by a ranged enemy hits the player. Another way of taking damage is from falling off the edge of the level.

Collecting pickups
There are item prefabs placed throughout the level. When the player runs into an item it triggers, removing the object from the scene and invoking a function that applies the effect of that item.

Falling off the edge
The level will be set on an elevated platform such as a table or cardboard box. When the player goes over the edge they will fall off until reaching the floor. At this point, the player will lose health and be respawned back at the start position.

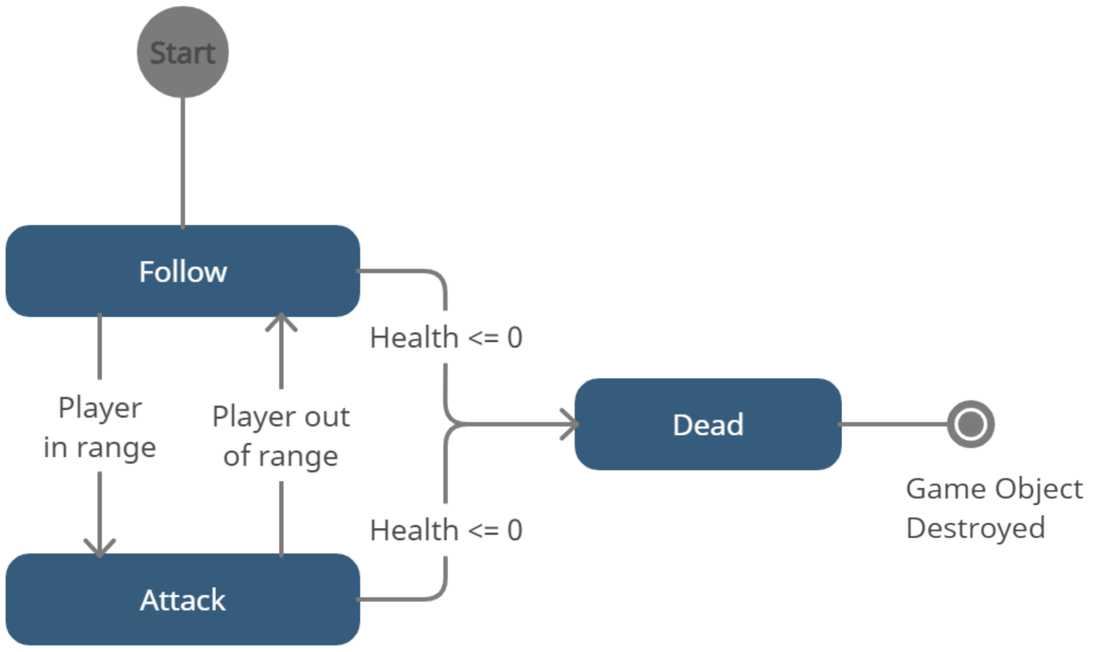
Artificial Intelligence

The enemies in the game will act as NavMesh agents that will use a navigation mesh that has been baked into the level for pathfinding. When each enemy has been spawned it will follow a path from its current location to the player location until it gets in attack range.What the enemy A.I. does, will depend on the type of enemy being used.

Self destructing enemy

There will be a small, fast enemy that will follow the enemy until colliding and then get destroyed on collision.

Melee enemy

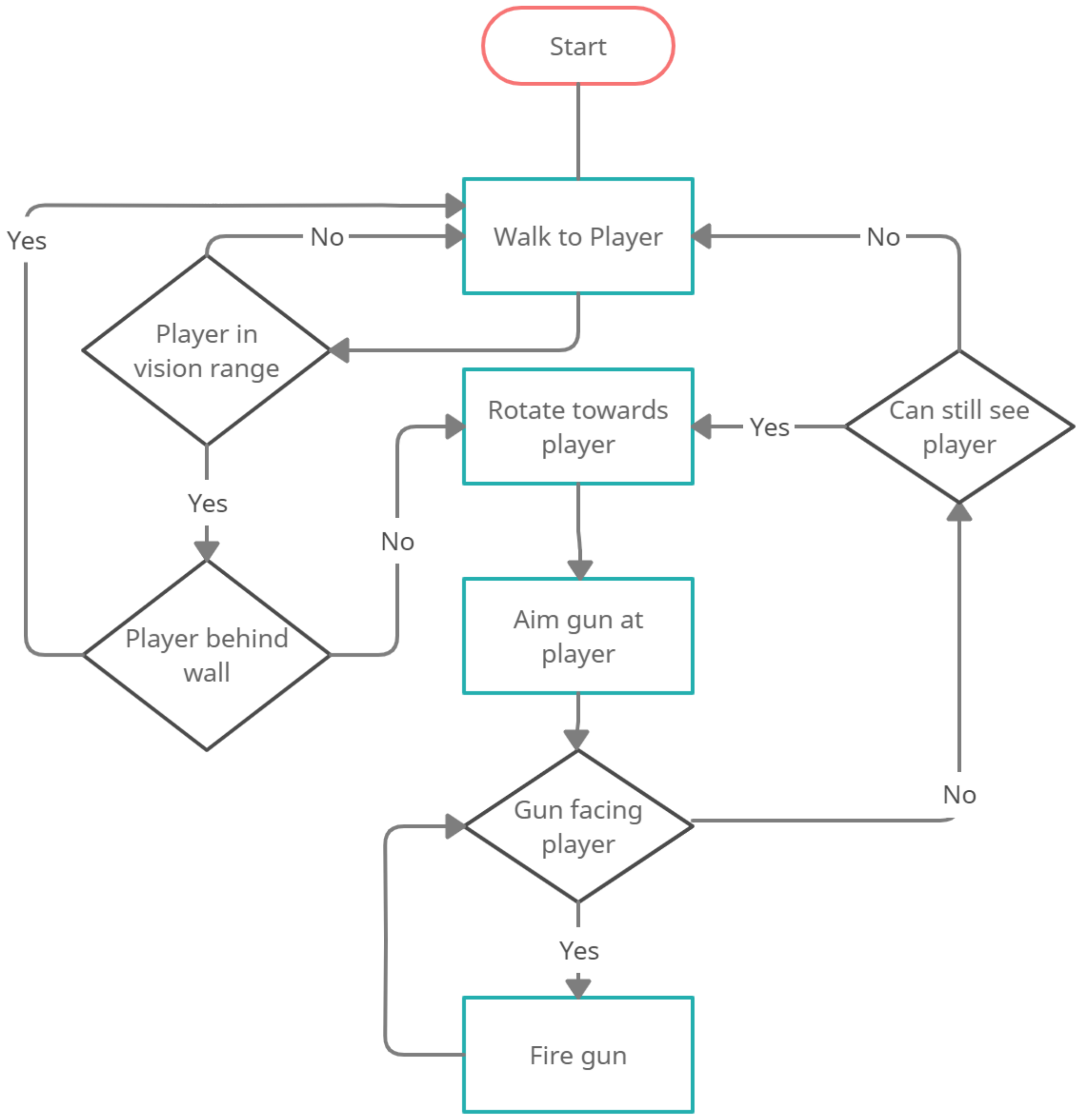


There will also be a medium-sized enemy that will perform melee attacks on the enemy once in range. This enemy will have 2 different states. A follow state and an attack state. They will switch between the 2 states depending on the length of the vector between the player and the enemy.

Ranged enemy

The will also be a large and slower-moving ranged enemy which will attack by firing projectiles at the player. This enemy will draw a raycast between its current location and the player location and while it is in follow state: If the player is in range and there is no wall in the way between the enemy and the player, it will switch from the follow state to attack state.

Below is the basic flow of how the A.I. for this enemy operates.



Physics

All physics in the game will be handled using the Unity Engine. Any object that needs to be knocked around can be controlled using the RigidBody component.

Unity colliders will be used for collision detection with capsule colliders for the player and enemies and box colliders for all other game objects. The Character Controller/RigidBody component will be attached to both the player and the enemy game objects to detect collisions with another object.

The game will include gravity that pushes the player back onto the ground. There will also be the ability to jump as well as fly upon the jetpack against the engine's gravity. To achieve this a calculation will need to be performed based on the lift/drag etc.

If the player walks off the edge of the level they will fall down until colliding with a kill box collider attached to the floor of the bedroom..

Items

Pickups

Items in the game will be placed at different locations in the level. When the player picks up each item it will trigger its effect which depends on the type of item. A list of items is shown below.

Ammo
This will increase the amount of ammo the player has to fire with. There will be three different types of ammo pickups. Arrow pickups, machine gun ammo, pickups and laser ammo pickup. Each of these ammo pickups will increase the ammo for that specific weapon

Health Pickup
This item increases the amount of health the player has if they are below maximum health.

Armour
This item adds armour to the character which acts as additional health. While the character has armour equipped if they are damaged, they will lose armour instead of health.

Weapons

There are 3 different weapons that the player will use. A crossbow, a water gun, and a laser gun.

Crossbow: Fires one projectile per press of the button/key
Water gun: continues to fire projectiles while the button/key is held down
Laser gun: This will use the physics raycast to send out a laser from the gun to the first object it collides with. Any enemy that is intercepting with the laser will take damage.

Game Flow

'Mission' / 'Level' structure

The main level will be stored within the scene of a Unity Project. The scene will be loaded when the player chooses to start the game from the main menu.

Objectives

The player must destroy all enemy waves without dying to complete the level. The HUD on the GUI layer will tell the player which wave they are up to. Player health is also displayed on the HUD.

Levels

The game will contain one level which is on a playmat located in a child's bedroom. Different levels will be made up of blocks with ramps connecting them. All walkable locations will be baked into the Navigation Mesh.

User Interface

Menus

There will be a main menu at the start of the gaming and an options menu when the player pauses the game. Menu items will include the following

Main Menu:

The main menu screen will contain the following buttons
Start Game: Starts a new game
Options: Goes to the options menu
Credits: Takes the player to a screen displaying the credits
Quit: Exits the unity project

Options Menu:

The options menu will contain the following GUI components
Volume Slider: Changes the volume of the sound effects and music
Brightness Slider: Changes
Fullscreen checkbox: Toggles between fullscreen and windowed mode
Back Button: Takes the player back to the previous menu

Pause Game Menu:

The pause game menu will contain the following GUI co

Camera

The camera used in the game will be a 3D perspective camera that is a child to the player transform. When the player moves, so does the camera. When the player changes the direction, they are facing using the mouse or controller joystick the camera rotates.

Controls

There will be two main different input modes for this game. Keyboard and mouse or game controller. The following controls are listed below.

Mouse:
Horizontal Mouse Drag - Rotates camera on the y axis
Vertical Mouse Drag - Rotates camera on the x-axis
Left mouse button - Fire ammo
Right mouse button - Fire grappling gun (optional)
Mouse wheel - change weapon

Keyboard:
W or Up: Move forward
S or Down: Move backward
A or Left: Move sideways left
D or right: Move sideways right
Space: Jump
Horizontal mouse drag: change direction
Vertical mouse drag: Look up and down
Left mouse button: Fire weapon
Right-Click: Fly on Jetpack
Esc: Pauses the game and brings up the options menu

Game Controller:
Left joystick: Movement
Right joystick: Rotate camera
A/B: Jump
Left/Right bumpers: Switch between weapons
Right trigger: Fire weapon
Left trigger: Jetpack
Start/Pause: Pause
Select menu options: Up/down joystick

Technical Risks

The main technical risk is having every feature implemented by one programmer and having it all working. Specific difficulties programming this game may include the following.

- Having a game controller work with the game as well as the keyboard and mouse
- Having the camera rotate up and down without rotating the player
- Controlling the physics on the jetpack and/or grappling gun
- Ensuring that enemy agents don't end up blocking each other when moving towards the player. This may only happen if there are too many enemies is one of the waves
- Getting the A.I. on 2 different types of enemies working correctly.

Asset List

Below is a list of all of the Game object prefabs and scripts needed to make the game run.

Game Objects/Prefabs

Player

This game object contains a rigid body component, the player scripts and several children game objects

Children include

- Camera
- Capsule
- Weapons

Solid objects

These are 3D models with a material and a collider attached to them. They don't do anything other than blocking the characters or acting as the ground for them to move on.

Walkable objects

These are game objects in the level that the player walks on and collides with. They will also be baked into the NavMesh so that the enemies can use them. If these objects contain a slope then the player and enemies will walk up the slope, if they are vertical then they will block the characters.

- PlayMat
- Rectangular block vertical
- Rectangular block horizontal
- Rectangular Arch block
- Ramp small
- Ramp large
- Cylinder block

Background objects

These are other 3d objects that are not part of the level but are visible to the player. They are also solid, preventing the player from passing through them. These include the following

- Toybox (or cardboard box)
- Room
  - Floor
  - Walls
  - Ceiling
- Light
- Chair
- Poster
- Bed
- Door
- Bookshelf
- Desk
- Computer
- Bedside Table
- Alarm clock
- Lamp

Enemies

All of the prefabs below will contain the NavMesh agent component and EnemyController script.

Toy Car

Small in size and fast but has low health. This enemy deals a high amount of damage on impact but is destroyed as soon as it collides with the player.

Soldier

Medium in size and moves at medium speed and has medium health. This enemy will follow the player until it is in attack range.

Robot

Large in size and has large health but moves slowly. The enemy will also have a shield at the front of it. If the projectiles/lasers fired by the enemy collide with the shield, the enemy will take less damage than when being hit on other parts of the body.

Weapons

A different 3D model will be used for each type of gun. All of the weapons will be models that are placed in front of the player's camera and will switch on and off depending on which weapon the player has selected. Below are the different game objects that will be used for weapons.

- Rifle
- Machine gun
- Laser Gun

Items

These are prefabs that will be tagged as “collectable” and will have the ItemController script attached to them.

- Ammo pickup
- Jetpack fuel
- Health pack
- Armour

GUI

The user interface will include the main menu screen, a couple of additional dialogs and the heads up display (HUD) that appears in the game.

Menus

Below is a list of game objects that will appear on the canvas for each menu in the game

Main Menu

- Title logo
- Background
- Company logo
- Start Game button
- Options button
- Credits button
- Quit button

Options Menu

- Dialog background
- Dialog title
- Volume label
- Volume slider
- Brightness label
- Brightness slider
- Fullscreen label
- Fullscreen checkbox
- Back button

Pause Dialogue

- Pause dialogue
- Dialogue title
- Resume button
- Options button
- End game button

Note: All buttons on the menu will have a function attached to them from Unity’s onClick event.

HUD

This is the UI that appears on the screen while playing the game

- Health label and health meter
- Fuel Label and jetpack meter
- Armour label and armour mater
- Wave label and wave number
- Time Elapsed
- Aiming crosshair

Scripts

Monobehaviour classes

GameManager Object

GameManager: Controls the main flow of the game. Will contain game states for Init, running, paused and dead. This class will call functions on the spawn manager. The variable for gravity will also be stored here.

This class will also keep track of all the enemies' health and destroy them when they reach zero.

SpawnManager: Takes an array of EnemyWave scriptable objects and spawns them all sequentially. When the game begins it will iterate through the SpawnedEnemy objects and instantiate the appropriate enemy prefab for each one.

The script also contains a variable for the amount enemies left in the wave which decreases each time one is destroyed. If the number of enemies remaining reaches zero, then it will spawn the next wave.

This script also spawns new item pickups when an enemy wave is defeated.

GUIController: Updates the information on the Heads up Display (HUD) at runtime

Canvas

GUIManager: Handle the button click of all menus in the game and update the values on the HUD when the GameManager tells it to.

Player Gameobject

PlayerController: Stores all of the player's main variables. This includes values for player max health, current health, current fuel, max fuel, armour, selected weapon and the amount of ammo in each weapon. Any changes to current health, fuel, ammo or armour such as taking damage will be controlled by this script. It will also send the player back to the starting point after falling off the level as well as any other collision checks that need to be made. If the player runs out of health it will then change the game state by calling the GameManager script.

PlayerMovement: Controls all player movement in the game from player input using a CharacterController component. Still will include running as well as jumping and using the jetpack. Variables include movement speed, jump height, jetpack lift speed.

GunController: Firing each shot with the weapon is controlled in this script.

PlayerSound: Takes sound objects for all different sound effects to be played by the player's audio source and plays them at the correct time.

Enemy GameObject

EnemyController: Takes an enemy scriptable object. The enemies current health is stored here. If the enemy collides with the player's bullet or laser damage is applied to their health.

This script also contains the code needed to move enemy agents along the NavMesh towards the player while it is in "follow" state. Enemy animations will also be controlled on this script by sending parameters to the enemy's animator. Melee enemies will have 2 different states: A follow attack state, while ranged enemies will have a follow, attack and aim state.

SelfDestuctingEnemy: Attached to the car enemy. When the box collider trigger collides with the player the card game object will be destroyed. An explosion particle system will then be played and damage will be dealt to the player's health.

MeleeEnemyAI: Attached to the soldier enemy. It contains values for the attack range and whether or not it is destroyed on the condition. When the enemy is in attack range it will switch to attack state and if the player moves out of range it will change back to follow state. When in attack state the enemy will deal damage to the player every x amount of frames while the player is still in range. This type of enemy will have an attack speed value.

RangeEnemyAI: Attached to the robot enemy. This script contains all the conditions that the ranged enemy makes when transitioning between following the player. A raycast will be drawn at all times from the ranged enemy to the player to check if it can see the player. If it can, the enemy will change to an aim state where it will rotate itself as well as the gun to face the player. When the gun is facing the player the enemy will switch to attack state where it will fire projectiles continuously at the player with a time delay between each one.

Public values for this script will be the view distance, firing delay, enemy turn speed and aiming speed.

EnemySound: Takes sound objects for all the different sound effects to be played by the enemy's audio source and plays them at the correct time.

Projectile

ProjectileController: Controls what the bullet does after being instantiated.

Item

ItemController: Takes an item scriptable object. The variables in this script will be accessed by the PlayerController.

GUI

MenuController: Contains a series of functions that handle the clicking of different buttons on the

ScriptableObject classes

Weapon: Contains values for damage and maximum ammo
MachineGun: Extends weapon class. Contains the value for firing rate.
LaserGun: Extends weapon class. Contains the value for damage rate.
Enemy: Contains variables for max health, movement speed, attack rate and damage per hit
RangedEnemy: Extends enemy class and includes a variable for attack range
EnemyWave: Contains an array of type SpawnedEnemy which contains the variables for the enemy prefab game object and the spawn point game object
Item: Contains an Enum for different values to adjust and value for modification