Microgame #4 TowerDefense Progress Report

Name: Alan L. Perez

Id: 004862867

Unity File: https://play.unity.com/mg/other/towerdefensegamepublish

1. Create a new project TowerDefense
2. Create folders under Assets: Animations, Fonts, Prefabs, Scenes, Scripts, Sprites, Tiles
3. Download Assets.zip from Canvas. Unzip, and drag sprites into Sprite folder.
4. Drag the Pride font into the Fonts folder.
5. Click on the Titles in sprite and change “Pixel Per Unit” to 32. Change “Filter Mode” to “Point (no filter)”. Change “Sprite Mode” to “Multiple”.

Graphical user interface

Description automatically generated

1. Then click on “Sprite Editor” and click “Apply”
2. Slice the tiles automatically and select “Type” to be “Grid By Cell Size”.

Graphical user interface, application

Description automatically generated

1. Select the “Pixel Size” as “X: 32, Y: 32”. The click “Slice” and “Apply”.

Graphical user interface

Description automatically generated

1. Create the tile map



1. Create a new “Tile Palette” with the tiles created earlier, and drag the tiles in “Sprites” to the “Tile Palette” named “Background”.

A screenshot of a computer

Description automatically generated with medium confidence

1. Then grab tiles from “Background” to create a grass-like background, and grab tiles to create the road.
2. Then rotate the tiles to create a more aesthetic looking path. Some tiles will have to be flipped either 90, 180, or 270 degrees.

Graphical user interface

Description automatically generated with medium confidence

Graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated

1. The final map should look similar to the one below

Chart

Description automatically generated

1. Create Tower Spawn, change “Order In Layer” to 1. Change size, and position.

Graphical user interface

Description automatically generated

1. Change the “Pixels Per Unit” to 32 in the “Sprites” folder.

Graphical user interface

Description automatically generated

1. Create “TowerSpawnController.cs” and attach it to the “TowerSpawn” game object. Then Attach a “Box Collider 2D” to the game object. Then set “Is Trigger”.

Graphical user interface

Description automatically generated

1. Go into “TowerSpawnController.cs” and

Graphical user interface, application

Description automatically generated

Text

Description automatically generated

1. Create the “Game Controller” game object and attach “GameController.cs” to it.

Graphical user interface

Description automatically generated

1. Go into “GameController.cs” and create variables.

Graphical user interface, application

Description automatically generatedText

Description automatically generated

1. Then go to the “TowerSpawnController.cs” and create a variable “GameController cont” and create the “Awake()” function. In “Awake()” initialize the “cont” variable. Then go into “OnMouseDown()” use “Instantiate()” to build a tower.

Graphical user interface, application

Description automatically generated

1. Go into “Sprites” and drag “TDCannonTurret” and rename it “Cannon”. Then change the size. Afterwards we create the base of the “Cannon”. We drag “CannonBase” to the “Hierarchy” and make it a child game object to “Cannon”.



1. Save Cannon as a Prefab.

Graphical user interface, application

Description automatically generated

1. Drag the cannon prefab into the “GameController” game object under “GameController.cs” in the variable “Tower”.

Graphical user interface, text

Description automatically generated

1. Tower Spawn “Order in Layer” is 1.

A screenshot of a computer

Description automatically generated with medium confidence

1. “Cannon” set “Order In Layer” to 2 and “CannonBase” 1.

A screenshot of a computer

Description automatically generated with medium confidenceGraphical user interface

Description automatically generated

1. Go into “TowerSpawn.cs” and under “OnMouseDown()” make it so no other towers can be spawned if there is already one there.

Graphical user interface, text

Description automatically generated

1. Create “CannonController.cs” and attach it to the Cannon.

A screenshot of a computer

Description automatically generated with medium confidence

1. Go into “CannonController.cs” and create variables. Then create “OnEnable()” function and set the attack radius.

Shape

Description automatically generated with low confidenceText

Description automatically generated

1. Set “Is Trigger” in the “Cannon” game object.

A screenshot of a computer

Description automatically generated with medium confidence

1. Go into “Sprites” and drag “EnemyAPC” and drag it to the Hierarchy. Create the tag “Enemy” and label “EnemyAPC” as “Enemy”. Set size, and rotation, and create box collider 2D. Make “Order In Layer”. Attach RigidBody2D. Finally create “EnemyController.cs” and attach it to “EnemyAPC” game object.

Graphical user interface

Description automatically generated

1. Then go into “EnemyController.cs” and have it so that the if an enemy comes into the attack range of a tower, the enemy will be the target.

Text

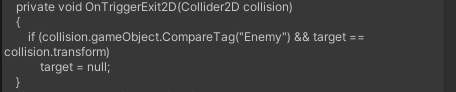
Description automatically generated

1. Then when the enemy leaves the range of a tower, the taget will change to “null”.

Text

Description automatically generated

1. Then create the “OnTriggerStay2D()” to switch from one enemy to another once the original target has left the range.



1. If the target is not “null” we will rotate the top of the “Cannon” in the direction of the enemy. We will do this in the “Update()” function.

Text

Description automatically generated

1. We set rotation speed as 5 in “EnemyAPC” game object.



1. We just want the top of the “Cannon” to rotate. We create a variable “child” and create a function “LateUpdate()” in order to handle this.  
   Graphical user interface, text

   Description automatically generated
2. Set the child variable by passing it the “CannonBase”.



1. We need time cool down time between when the “Cannon” is able to shoot a bullet. We do this in “CannonController.cs”.

Text

Description automatically generated

1. Take the “Bullet” sprite from the “Sprites” folder and add it to the “Hierarchy”. Then attach “Circle Collider 2D” and “Rigidbody 2D”to it. Set Mass and Gravity. Create “BulletController.cs” and attach it to the “Bullet” game object. Save bullet as prefab.

Graphical user interface

Description automatically generated

1. Go into “BulletController.cs” and create variables. Then in “Awake()” we get the reference to the “Rigidbody2D”. Then in “OnEnable” we add a force to the bullet. Then we create the “Disable()” to set bullet inactive after 4 seconds.

Text

Description automatically generated

1. Then in the “Bullet” game object under “BulletController.cs” we set the bullet “Speed” and “Damage”.

Graphical user interface, text, application

Description automatically generated

1. Then drag “Bullet” prefab into inspector for “Cannon”.

Graphical user interface, application

Description automatically generated

1. Go into “Cannon” game object and set shoot speed to 0.3 under “Cannon Controller (Script).

Graphical user interface, application

Description automatically generated

1. In the “BulletController.cs” we will make the function “OnTriggerEnter2D()”, so that if a bullet collides with an enemy it will take damage.

Text

Description automatically generated

1. Create “EnemyController.cs” and attach it to “EnemyAPC”.

A screenshot of a computer

Description automatically generated with medium confidence

1. In “EnemyController.cs” we create the function “TakeDamage()”.



1. Save “EnemyAPC” as a Prefab.

Graphical user interface, application

Description automatically generated

1. Go into “BulletController.cs” under under “OnTriggerEnter2D()” we will disable the bullet if it comes into contact with the enemy.

Graphical user interface, text

Description automatically generated

1. Then go into “EnemyController.cs” and create a variable for the “Rigidbody2D” named “EnemyRigidBody”. Create other variable such as “speed”, “maxHealth” and “health”. Then in “OnEnable()” we initialize “health” as “maxHealth” and we decrement it in “TakeDamage”.

Text

Description automatically generated



Shape

Description automatically generated with medium confidence

Graphical user interface, text

Description automatically generated

1. Next, we will create the “Waypoints” game object along with its children to have point for our enemy to turn on. This is how the enemy will know which direction it needs to be moving in.

Text

Description automatically generated

1. Then we will create an empty game object “SpawnPostion” so that the enemy will have a place to spawn from.  
   
2. Go into “GameController.cs” we add an array, and we will store the 5 waypoints into this array.



1. Drag the 5 waypoints from the Hierarchy and put them into the array we created in the “GameController” game object.

Text

Description automatically generated

1. Go into “EnemyController.cs” and create new variables. Then go into the function “Awake()” and reference the GameController. In the “OnEnable()” function we set “currentWaypoint” as 0, and we set “target” as “cont.waypoints[currentWaypoints]”. Which means we set the first target as first element in “waypoints” and our target is the current waypoint.

Graphical user interface, text, website

Description automatically generated

Shape

Description automatically generated with medium confidenceGraphical user interface, text, website

Description automatically generated

Graphical user interface

Description automatically generated

1. Next we will update the position in the “Update()” function. Then we add force to the enemy so it can move.

Text

Description automatically generated



1. Next, we update the target, so that the enemy arrives at the first waypoint we update it to the next waypoint. Also, if the enemy is at the last waypoint we will disable the enemy and the player will take damage. If the distance value is too small, the enemy will just spin in circles.

Text

Description automatically generatedText

Description automatically generated

1. When the enemy rotates the enemy does not move. We add the variable “canMove = true”. Then in “Update()” when the enemy is rotating it will have no speed, so “canMove = false”. Then it will be able to move after 1 second when it is done rotating.

Text

Description automatically generatedText

Description automatically generated

1. Then we make the function “CanMove()” that sets the variable “canMove” back to true.Shape

   Description automatically generated with medium confidence
2. In the “EnemyController.cs” under “Update()” add the condition that if “canMove” is false we can’t move the enemy.



1. In the “GameController.cs” we define the player’s health. Then we create the function “TakeDamage()”.



1. Next, we go into “EnemyController.cs” and use the “TakeDamage()” function from “GameController.cs” so the player’s health will decrease if an enemy leaves the map successfully.

Text

Description automatically generated

1. In the “EnemyAPC” game object we set the “Damage” as 1 under “Enemy Controller (Script)”.

Graphical user interface

Description automatically generated

1. In the “Game Controller” game object. We set the “Max Health” to 10.

A screenshot of a computer

Description automatically generated with medium confidence

1. We go into the “EnemyController.cs” and under “TakeDamage()” function we create a condition where if the health of the enemy is less than or equal to zero we will take the game object of the Enemy Controller and set active equal to false to get rid of enemy.Graphical user interface, text

   Description automatically generated
2. We will then create a health bar for the player. We go into “Game Object” -> “UI” -> “Image”. We name the “Image”: “Health” and we set the position to “top center”. We then set the x and y positions, and color. Then we will add a white image to “Sprite” folder, and set the image source with the white image. “Image Type” is filled and fill method is “Horizontal”.

Graphical user interface, application

Description automatically generated

1. Next, we duplicate the “Health” image and put it above “Health”. Then we set the color to black.

Graphical user interface, text

Description automatically generated

Graphical user interface, application

Description automatically generated

1. Afterwards, go into “GameController.cs” and add “using UnityEngine.UI”. The add variables. In the Awake function, we create the health bar, and in “Update()” we update the health bar.

Text

Description automatically generatedText

Description automatically generated

1. We then go into the “GameController” game object and set “Health Image” with the “Health Image” game object, and we set “Lerp Speed” to 10.

A screenshot of a computer

Description automatically generated with medium confidence

1. In “GameController.cs” we will have it so that if the player has to pay to set a tower, and if they do not have enough money they can’t set a tower. We declare variables. Then we create a function “GiveMoney” so the player money can go up.

Text

Description automatically generated with medium confidence

1. Then in “TowerSpawnController.cs” under “OnMouseDown()” if the money is greater than or equal to the cost of the tower, we place a tower. Otherwise, we do not. Then we decrease the money.

Graphical user interface, application

Description automatically generated

1. In the “TowerSpawnController.cs” we create the function "OnMouseOver()” if we have enough money the “TowerSpawn” will be green, but if we do not it will be red. Then we create the function “OnMouseExit()” the color will be white if the mouse is no longer hovering over the “TowerSpawn”.

Graphical user interface, text, application

Description automatically generated

1. Then we will set the cost of the tower and the amount of money the player has in the “GameController” game object.
2. Next, we save the “TowerSpawn” game object as a prefab.

Graphical user interface, application

Description automatically generated

1. Then we add multiple “TowerSpawn[s]” to the game.

A picture containing text

Description automatically generated

1. Next, we will show the amount of money the player has in the game. We select “GameObject” -> “UI” -> “Text”. Then we set the position, width, and height of the text. For the font we will use out “Pride” font in our “Fonts” folder. We set “Horizontal/Vertical Overflow” to “Overflow”.

A screenshot of a computer

Description automatically generated with medium confidence

1. In the “GameController.cs” create the variable “moneyText”. Then you go in to update and set the “moneyText” and update it throughout the game.

Graphical user interface, text

Description automatically generated

1. We go into “GameController” game object and we set “Money Text” with the “Money Text” game object from the “Hierarchy”.  
   Graphical user interface, text, application

   Description automatically generated
2. In the “BulletController.cs” we add a function “OnDisable(). We cancel any “Invoke”. Shape

   Description automatically generated with medium confidence
3. If a “TowerSpawn” is too close to one another, it will not let you. Therefore, we must go into the “Cannon” prefab and in “Layer” we select “Ignore Raycast”. The we select “Tes, change children”. This will remedy this issue.

Graphical user interface, application

Description automatically generated

1. Then, in the “EnemyController.cs” we will implement if an enemy is destroyed the player will receive money. We set the variables.

Graphical user interface, text, application, chat or text message

Description automatically generated

1. In “EnemyAPC” we set the amount of money the enemy will give the player if the enemy is destroyed.

A screenshot of a computer

Description automatically generated with medium confidence

1. We the create “BaseTower.cs” and we take all the code from “CannonController.cs” and cut and paste it into “BaseTower.cs”. The “BaseTower.cs” is a parent class.

Text

Description automatically generated

Text

Description automatically generated

1. The “CannonController.cs” is a child class.



1. We will then go into “BaseTower.cs” and change “Shoot()”.

Graphical user interface, text

Description automatically generated

1. We will then go into the “CannonController.cs” and implement the “Shoot()”.

Text

Description automatically generated

1. Since we want the child class to use the variable cools from the “BaseTower.cs”, we will add protected in front of it.



1. For the “Cannon” we add a “bulletSpawnPosition” as a child game object.

Graphical user interface, text

Description automatically generated

1. Then we drag the “Cannon” prefab into the “Hierarchy” and we rename it to shotty. Then we unpack it. We then change the sprite to “TDShottyTurret”. We then change “CannonBase” game object to the “TDShottyBase” sprite. Then we add another bullet spawn positions. Then we change the “Shoot Speed”, “Rotation Speed”, and “Attack Range”. We remove the “Cannon Controller (Script)” and change it to “Shotty Controller (Scripts)”.

Graphical user interface, text, application

Description automatically generated

Graphical user interface

Description automatically generated

1. We then go into the “ShottyController.cs”. We make it into a child class of “BaseTower.cs” and implement “Shoot()”.

Graphical user interface, application

Description automatically generatedText

Description automatically generated

1. We go into “BaseTower.cs” and create an array of “bulletSpawnPositions”. We use this in “CannonController.cs”.



Graphical user interface, text

Description automatically generated

1. We then go into “Shotty” game object and under “Shotty Controller (Script)” we set the “Shoot Speed”, “Rotation”, “Attack Range”. Then save “Shotty” as a prefab. For the child variable we drag the “CannonBase” to the child.

Graphical user interface

Description automatically generated

1. Next, in the “Cannon” prefab, we lock the inspector and drag the “bulletSpawnPosition” into the array of “Bullet Spawn Positions”. Then we unlock it.

A screenshot of a computer

Description automatically generated with medium confidence

1. We then do the same for the “Shotty” game object.

Graphical user interface

Description automatically generated

1. Then we drag the “Bullet” prefab into the “Hierarchy” and rename it “ShottyBull.et. We then unpack it, change the size, speed, and damage. We then make it a prefab.

Graphical user interface

Description automatically generated

1. Then we go into the “Shotty” game object and we initialize the “Bullet” variable under “Shotty Controller (Script)” and attach the “ShottyBullet” prefab. A screenshot of a computer

   Description automatically generated with medium confidence
2. We go into the “ShottyController.cs” and create an outer for loop and an inner for loop. So, each spawn position will shoot 5 bullets. They will have a different rotation value.

Text

Description automatically generated

1. We are then going to create the Mortar cannon. We drag the “Cannon” prefab into the “Hierarchy”. We then unpack it. Rename it “Mortar”. Change the sprites of the “Cannon” and “CannonBase” to “TDMortarTurrent” and “TDMortarBase”, respectively. A screenshot of a computer screen

   Description automatically generated with medium confidenceGraphical user interface

   Description automatically generated
2. We then create the “MortarController.cs”, we remove “CannonController.cs” from the “Mortar” game object, and attach “MortarController.cs”. Graphical user interface

   Description automatically generated
3. The “MortarController.cs” is a child of the “BaseTower.cs”. We then implement the “Shoot()” function.

Text

Description automatically generated

1. We go into the “Mortar” game object and set “Shoot Speed”, “Rotation Speed”, and “Attack Range”. Then we make “Mortar” into a prefab.

Graphical user interface

Description automatically generated

1. We go into “Mortar” game object and drag the “CannonBase” to the child variable. For the “Bullet Spawn Positions” array we drag the “bulletSpawnPosition” into it. Graphical user interface

   Description automatically generated
2. The we drag “Bullet” prefab to the “Hierarchy” and we rename it “MortarShell”. We unpack it. Then we set the Scale, the Speed, mass, and the damage.

Graphical user interface

Description automatically generated

1. Drag “MortarShell” to the prefab folder.

Graphical user interface, application, Teams

Description automatically generated

1. Go into the “Mortar” prefab and under “Mortar Controller (Script)” insert “MortarShell” into “Bullet".

Graphical user interface, application

Description automatically generated

1. We then go into the “BulletController.cs” copy and paste everything into the “MortarShellController.cs”

Text

Description automatically generated

Text

Description automatically generated

1. We then add variables to “MortarShellController.cs”

Graphical user interface

Description automatically generated

1. Then in the “OnTriggerEnter2D” we add a collider2D array named “hit”. Then we add a for each condition.

Graphical user interface, text, application

Description automatically generated

1. We add the function “OnDrawGizmos()” so we can see the circle drawn by the “MortarShell”.

Graphical user interface, text, website

Description automatically generated

1. Then on EnemyAPC, we create an “Enemy” layer.

Graphical user interface, application

Description automatically generated

1. The in “MortarShell” game object we make “Enemy Mask” the enemy layer.

A screenshot of a computer

Description automatically generated with medium confidence

1. The we go into “Edit” -> Project Settings” -> “Physics2D” we uncheck “Enemy”, “Enemy”.

Text

Description automatically generated

1. We go into “GameObject” -> “UI” -> “Button”. The button is then bottom center. Set “Pos (x/y)”, and “Width/Height”. For the Button we write “Cannon”. We use our “Pride” font and choose “Overflow” for “Horizontal/Vertical Overflow”.

Graphical user interface, application

Description automatically generated

1. Then we go into the “GameController.cs” and add variablesn and add “HideInspector”.Graphical user interface

   Description automatically generated
2. We then make the function “UpdateTower()”. We will store three towers in the “tower”array. Then the user will select which tower with “currentTowersCost”. In base Tower we have “cost”. Then in Awake, we will call “UpdateTower()”.

Text

Description automatically generated with medium confidenceText

Description automatically generated

1. Then in “GameController” game object we fill the “Towers” array with the towers.

Graphical user interface, application

Description automatically generated

1. Then in the “Button” we drag the “GameController to “On Click()”

Graphical user interface, application

Description automatically generated

1. Then we duplicate the “Button” twice to make the buttons for the “Mortar” and “Shotty”. We use the same process as we did for “Cannon”.

Graphical user interface, application

Description automatically generated

1. Then we set the cost for all the towers.

A screenshot of a computer

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated with medium confidence

1. Go into “GameController.cs” we make variables “timeBetweenSpawnLow” and “timeBetweenSpawnHigh” and “cools” will pick a number in between those variable for the cool down time. Then we make the variable “spawnPosition”. Lastly we will create the variable “enemies” which is an array that will hold the towers.

Text

Description automatically generated

1. Then we will create the function “SpawnEnemy()” to handle spawning enemies. It will handle the movement of the turret, as well as the cool time. Depending on the size of the tower will determine the cool down time it takes to spawn a tower.

Text

Description automatically generated

1. In the update function we will update the cool down time.

Text

Description automatically generated

1. Then we go into the “GameController” game object and initialize the variables “timeBetweenSpawnLow”, “timeBetweenSpawnHigh”, “SpawnPosition” under “Game Controller (Script)”. We also initialize the “Enemies” array.

A screenshot of a computer

Description automatically generated with medium confidence

1. We then take the “EnemyAPC” from prefab to the “Hierarchy” and rename it “EnemyJeep”. Then we unpack the prefab in the “Hierarchy”. We then go into the game object and change its sprite, as well as, the size of the sprite.

Graphical user interface

Description automatically generated

1. Since the sprite did not come out as clear as we wanted to. We click on the “EnemyJeep” sprite int the “Sprites” folder and under “Filter Mode” we select “Point (No filter). Then click apply.

Graphical user interface

Description automatically generated

1. Then we go back into its game object and adjust its box collider to the size of the sprite.
2. Next, go into “EnemyAPC” prefab and adjust its “Max Health”.

Graphical user interface

Description automatically generated

1. We follow the same process again for the “EnemyTank” and “EnemyTruck” sprite that we did for “EnemyJeep”.

Graphical user interface

Description automatically generatedGraphical user interface

Description automatically generated

1. Once we have our enemies in order, we will go to the “GameController” game object and add our new enemies into the “Enemies” array.

Graphical user interface

Description automatically generated

1. The go into “GameController.cs” we will update “SpawnEnemies()” to account for the new enemies.

Text

Description automatically generated

1. Next go to “Sprites” folder click on the “explosion” sprite. Under “Sprite Mode” click “Multiple” and click “Apply”. The click the “Slice” -> “Grid By Cell Size”. Then for “Pixel Size (X/Y) input 192, we then click “Slice”.

Graphical user interface, application

Description automatically generated

Shift click the first and last sprite in “explosion and input it into the “Hierarchy”. We save it into the animation folder, and save it as “Explosion”.

Graphical user interface

Description automatically generated

1. The we create the script “ExplosionController.cs”. Then we attach it to the “Explosion” game object.

Graphical user interface

Description automatically generated

1. We then go into “ExplosionController.cs” and we add “OnEnable()” so that we can add the animation “Explosion” game object.

Text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

1. We then go into the “EnemyController.cs” and declare a variable game object “explosion”. We update “TakeDamage()” to instantiate the “explosion”.

Graphical user interface, text, application

Description automatically generated

1. We set the “Layer In Order” for explosion, and for every enemy prefab we will drag explosion prefab to it.

Graphical user interface

Description automatically generatedGraphical user interface

Description automatically generatedA screenshot of a computer

Description automatically generated with medium confidenceGraphical user interface

Description automatically generated

1. For the “Flash” we will shift click the five sprites and drag it into the “Hierarchy”. We will save it in the animation folder as “Flash” . Then we will change its size. We will also attach “ExplosionController.cs” to the game object.

A screenshot of a video game

Description automatically generated with medium confidence



A picture containing graphical user interface

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

1. We then go into “BaseTower.cs” and declare the variable game object “flash”. We then go into “CannonController.cs”, “MortarController.cs”, and “ShottyController.cs” and instantiate the flash.



Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

1. Then we save the “Flash” as a prefab.

Graphical user interface

Description automatically generated with medium confidence

1. The for each enemy in the prefab under each of their respective controller we add the “Flash” animation to the “Flash” variable.

A screenshot of a computer

Description automatically generated with medium confidence

Graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated

1. Next, we will go into Canvas, and download “Audios.zip”. Then we unzip the file. Then we create an “Audios” folder in “Assets” and drag the audio files into the folder. Graphical user interface, application

   Description automatically generated
2. Then we create an empty game object in the “Hierarchy”. We change the Position for X,Y,Z to zero. Then we attach an Audio Source to it and drag the “Penance v0.1” to the “AudioClip” variable. We click “Play On Awake” and “Loop” to it. As well as, adjust the volume.

Graphical user interface

Description automatically generated

1. Then we go into the “Explosion” prefab and attach an “AudioSource” to it. Then we attach the “explosion” audio clip to it. Then we click “Play On Awake”, as well as, adjust the volume.

Graphical user interface

Description automatically generated

1. The we go into “BaseTower.cs” and create a “protected AudioSource” named “src” so that all the children of “BaseTower.cs” can access it. Then we create the “Awake()” function to insert “GetComponent<AudioSource>” into “scr”.



Graphical user interface, text

Description automatically generated

Text

Description automatically generatedText

Description automatically generatedText

Description automatically generated

1. Then we go into the “Cannon” and “Mortar” prefab and attach “bullet” audio file to it.

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

1. The go into “Shotty” and do the same but with the “shotgun” audio clip.

A screenshot of a computer

Description automatically generated with medium confidence

1. Go into “GameController.cs” and we create the variables “gameOver” and “gameOverUI”. Then we will update the “Awake()”, “Update()”, and “TakeDamage()” functions to account for the game meets the criteria for being over.

Text

Description automatically generatedText

Description automatically generatedGraphical user interface, text, application, chat or text message

Description automatically generated

1. Then we go back to Unity Editor and add another child game object to the “Canvas” game object named “GameOverUI”. We adjust the size, color, position, add text. We then add our font “Pride” and for “Horizontal/Vertical Overflow” we apply “Overflow”.



Graphical user interface, application

Description automatically generated

1. Then we go into “GameController” game object and add the” GameOverUI” into the variable “Game Over UI” under “Game Controller (Script). 