Unity Lessons

Source Code - https://github.com/jasonweibel/HyperStream-Unity

Lessons

- Unity 2D Rogue Game
- Unity 2D UFO Game
- Unity Lander Game
- Unity Roll-a-ball Game
- Unity Simple Ball Rolling Game
- Unity Space Shooter Game
- Unity Survival Shooter Game
- Unity Tanks Game

Unity 2D Rogue Game

Original Project

Tutorial - https://unity3d.com/learn/tutorials/projects/2d-roguelike-tutorial

Q & A - https://forum.unity3d.com/threads/2d-roguelike-q-a.297180/

Lessons

- Unity 2D Rogue - Lesson 1 - Move Character
- Unity 2D Rogue - Lesson 2 - Create and Eat Food
- Unity 2D Rogue - Lesson 3 - Create and Move Enemy
- Unity 2D Rogue - Lesson 4 - Add Music

Unity 2D Rogue - Lesson 1 - Move Character

1. Open MoveObject script in Visual Studio
2. Uncomment AttemptMove method, lines 93 - 115.

```
91      //The virtual keyword means AttemptMove can be overridden by inheriting classes using the override keyword.
92      //AttemptMove takes a generic parameter T to specify the type of component we expect our unit to interact wi
            5 references | 0 changes | 0 authors, 0 changes
93      protected virtual void AttemptMove <T> (int xDir, int yDir)
94          where T : Component
95      {
96          //Hit will store whatever our linecast hits when Move is called.
97          RaycastHit2D hit;
98
99          //Set canMove to true if Move was successful, false if failed.
100         bool canMove = Move (xDir, yDir, out hit);
101
102         //Check if nothing was hit by linecast
103         if(hit.transform == null)
104             //If nothing was hit, return and don't execute further code.
105             return;
106
107         //Get a component reference to the component of type T attached to the object that was hit
108         T hitComponent = hit.transform.GetComponent <T> ();
109
110         //If canMove is false and hitComponent is not equal to null, meaning MovingObject is blocked and has hit
111         if(!canMove && hitComponent != null)
112
113             //Call the OnCantMove function and pass it hitComponent as a parameter.
114             OnCantMove (hitComponent);
115     }
```

3. Open Enemy script and uncomment AttemptMove method, line 39 - 54.

```
        //Override the AttemptMove function of MovingObject to include functionality needed for Enemy to skip turns.
        //See comments in MovingObject for more on how base AttemptMove function works.
        0 references | 0 changes | 0 authors, 0 changes
        protected override void AttemptMove <T> (int xDir, int yDir)
        {
            //Check if skipMove is true, if so set it to false and skip this turn.
            if(skipMove)
            {
                skipMove = false;
                return;

            }

            //Call the AttemptMove function from MovingObject.
            base.AttemptMove <T> (xDir, yDir);

            //Now that Enemy has moved, set skipMove to true to skip next move.
            skipMove = true;
        }
```

4. Open Player script and uncomment AttemptMove method, lines 131-157.

```
    //AttemptMove overrides the AttemptMove function in the base class MovingObject
    //AttemptMove takes a generic parameter T which for Player will be of the type Wall, it also takes integers for x and y direction to move in.
    1 reference | 0 changes | 0 authors, 0 changes
    protected override void AttemptMove <T> (int xDir, int yDir)
    {
        //Every time player moves, subtract from food points total.
        food--;

        //Update food text display to reflect current score.
        foodText.text = "Food: " + food;

        //Call the AttemptMove method of the base class, passing in the component T (in this case Wall) and x and y direction to move.
        base.AttemptMove <T> (xDir, yDir);

        //Hit allows us to reference the result of the Linecast done in Move.
        RaycastHit2D hit;

        //If Move returns true, meaning Player was able to move into an empty space.
        if (Move (xDir, yDir, out hit))
        {
            //Call RandomizeSfx of SoundManager to play the move sound, passing in two audio clips to choose from.
            //SoundManager.instance.RandomizeSfx (moveSound1, moveSound2);
        }

        //Since the player has moved and lost food points, check if the game has ended.
        CheckIfGameOver ();

        //Set the playersTurn boolean of GameManager to false now that players turn is over.
        GameManager.instance.playersTurn = false;
    }
```

5. In Player script uncomment AttemptMove<Wall> (horizontal, vertical) call in method Update, line 125.

```
#endif //End of mobile platform dependendent compilation section started above with #elif
        //Check if we have a non-zero value for horizontal or vertical
        if(horizontal != 0 || vertical != 0)
        {
            //Call AttemptMove passing in the generic parameter Wall, since that is what Player may interact with if they encounter one (by attacking it)
            //Pass in horizontal and vertical as parameters to specify the direction to move Player in.
            AttemptMove<Wall> (horizontal, vertical);
        }
    }
```

Unity 2D Rogue - Lesson 2 - Create and Eat Food

1. Open Player script in Visual Studio
2. Locate OnTriggerEnter2D method and uncomment else if for food and soda, lines 189 - 218. Do not uncomment the SoundManager.instance.RandomizeSfx method calls on line 198 and 214.

```
private void OnTriggerEnter2D (Collider2D other)
{
    //Check if the tag of the trigger collided with is Exit.
    if(other.tag == "Exit")
    {
        //Invoke the Restart function to start the next level with a delay of restartLevelDelay (default 1 second).
        Invoke ("Restart", restartLevelDelay);

        //Disable the player object since level is over.
        enabled = false;
    }

    //Check if the tag of the trigger collided with is Food.
    else if(other.tag == "Food")
    {
        //Add pointsPerFood to the players current food total.
        food += pointsPerFood;

        //Update foodText to represent current total and notify player that they gained points
        foodText.text = "+" + pointsPerFood + " Food: " + food;

        //Call the RandomizeSfx function of SoundManager and pass in two eating sounds to choose between to play the eating sound effect.
        //SoundManager.instance.RandomizeSfx (eatSound1, eatSound2);

        //Disable the food object the player collided with.
        other.gameObject.SetActive (false);
    }

    //Check if the tag of the trigger collided with is Soda.
    else if(other.tag == "Soda")
    {
        //Add pointsPerSoda to players food points total
        food += pointsPerSoda;

        //Update foodText to represent current total and notify player that they gained points
        foodText.text = "+" + pointsPerSoda + " Food: " + food;

        //Call the RandomizeSfx function of SoundManager and pass in two drinking sounds to choose between to play the drinking sound effect.
        //SoundManager.instance.RandomizeSfx (drinkSound1, drinkSound2);

        //Disable the soda object the player collided with.
        other.gameObject.SetActive (false);
    }
}
```

3. Open BoardManager script and uncomment LayoutObjectAtRandom (foodTiles, foodCount.minimum, foodCount.maximum) call in SetupScene method, line 144.

```
//SetupScene initializes our level and calls the previous functions to lay out the game board
1 reference | 0 changes | 0 authors, 0 changes
public void SetupScene (int level)
{
    //Creates the outer walls and floor.
    BoardSetup ();

    //Reset our list of gridpositions.
    InitialiseList ();

    //Instantiate a random number of wall tiles based on minimum and maximum, at randomized positions.
    LayoutObjectAtRandom (wallTiles, wallCount.minimum, wallCount.maximum);

    //Instantiate a random number of food tiles based on minimum and maximum, at randomized positions.
    LayoutObjectAtRandom (foodTiles, foodCount.minimum, foodCount.maximum);

    //Determine number of enemies based on current level number, based on a logarithmic progression
    int enemyCount = (int)Mathf.Log(level, 2f);

    //Instantiate a random number of enemies based on minimum and maximum, at randomized positions.
    //LayoutObjectAtRandom (enemyTiles, enemyCount, enemyCount);

    //Instantiate the exit tile in the upper right hand corner of our game board
    Instantiate (exit, new Vector3 (columns - 1, rows - 1, 0f), Quaternion.identity);
}
```

Unity 2D Rogue - Lesson 3 - Create and Move Enemy

1. Open GameManager script in Visual Studio
2. Uncomment enemies private variable, line 23.

```
8 references | 0 changes | 0 authors, 0 changes
public class GameManager : MonoBehaviour
{
    public float levelStartDelay = 2f;                  //Time to wait before starting level, in seconds.
    public float turnDelay = 0.1f;                      //Delay between each Player turn.
    public int playerFoodPoints = 100;                  //Starting value for Player food points.
    public static GameManager instance = null;          //Static instance of GameManager which allows it to be accessed by any other script.
    [HideInInspector] public bool playersTurn = true;   //Boolean to check if it's players turn, hidden in inspector but public.


    private Text levelText;                             //Text to display current level number.
    private GameObject levelImage;                      //Image to block out level as levels are being set up, background for levelText.
    private BoardManager boardScript;                   //Store a reference to our BoardManager which will set up the level.
    private int level = 1;                              //Current level number, expressed in game as "Day 1".
    private List<Enemy> enemies;                        //List of all Enemy units, used to issue them move commands.
    private bool enemiesMoving;                         //Boolean to check if enemies are moving.
    private bool doingSetup = true;                     //Boolean to check if we're setting up board, prevent Player from moving during setup.
```

3. In GameManagaer script uncomment instanciation of list of enemies, line 48 of Awake method.

```
//Awake is always called before any Start functions
0 references | 0 changes | 0 authors, 0 changes
void Awake()
{
    //Check if instance already exists
    if (instance == null)

        //if not, set instance to this
        instance = this;

    //If instance already exists and it's not this:
    else if (instance != this)

        //Then destroy this. This enforces our singleton pattern, mea
        Destroy(gameObject);

    //Sets this to not be destroyed when reloading scene
    DontDestroyOnLoad(gameObject);

    //Assign enemies to a new List of Enemy objects.
    enemies = new List<Enemy>();

    //Get a component reference to the attached BoardManager script
    boardScript = GetComponent<BoardManager>();

    //Call the InitGame function to initialize the first level
    InitGame();
}
```

4. In GameManager script uncomment clearing of the enemies list in the InitGame method, line 96.

```
//Initializes the game for each level.
2 references | 0 changes | 0 authors, 0 changes
void InitGame()
{
    //While doingSetup is true the player can't move, prevent player from moving while title card is up.
    doingSetup = true;

    //Get a reference to our image LevelImage by finding it by name.
    levelImage = GameObject.Find("LevelImage");

    //Get a reference to our text LevelText's text component by finding it by name and calling GetComponent.
    levelText = GameObject.Find("LevelText").GetComponent<Text>();

    //Set the text of levelText to the string "Day" and append the current level number.
    levelText.text = "Day " + level;

    //Set levelImage to active blocking player's view of the game board during setup.
    levelImage.SetActive(true);

    //Call the HideLevelImage function with a delay in seconds of levelStartDelay.
    Invoke("HideLevelImage", levelStartDelay);

    //Clear any Enemy objects in our List to prepare for next level.
    enemies.Clear();

    //Call the SetupScene function of the BoardManager script, pass it current level number.
    boardScript.SetupScene(level);

}
```

5. In the GameManager script uncomment the enemies.Add method call.

```
//Call this to add the passed in Enemy to the List of Enemy objects.
1 reference | 0 changes | 0 authors, 0 changes
public void AddEnemyToList(Enemy script)
{
    //Add Enemy to List enemies.
    enemies.Add(script);
}
```

6. In the GameManager script uncomment the if statement that checks if enemies count is 0. Also uncomment the for loop that calls the WairForSeconds method.

```
148     //Coroutine to move enemies in sequence.
        1 reference | 0 changes | 0 authors, 0 changes
149     IEnumerator MoveEnemies()
150     {
151         //While enemiesMoving is true player is unable to move.
152         enemiesMoving = true;
153
154         //Wait for turnDelay seconds, defaults to .1 (100 ms).
155         yield return new WaitForSeconds(turnDelay);
156
157         //If there are no enemies spawned (IE in first level):
158         if (enemies.Count == 0)
159         {
160             //Wait for turnDelay seconds between moves, replaces delay caused by enemies moving when there are none.
161             yield return new WaitForSeconds(turnDelay);
162         }
163
164         //Loop through List of Enemy objects.
165         for (int i = 0; i < enemies.Count; i++)
166         {
167             //Call the MoveEnemy function of Enemy at index i in the enemies List.
168             //enemies[i].MoveEnemy ();
169
170             //Wait for Enemy's moveTime before moving next Enemy,
171             yield return new WaitForSeconds(enemies[i].moveTime);
172         }
173         //Once Enemies are done moving, set playersTurn to true so player can move.
174         playersTurn = true;
175
176         //Enemies are done moving, set enemiesMoving to false.
177         enemiesMoving = false;
178     }
179 }
```

7. Open Enemy.cs script in Visual Studio and uncomment MoveEnemy method, line 58.

```
57      //MoveEnemy is called by the GameManger each turn to tell each Enemy to try to move towards the player.
        1 reference | 0 changes | 0 authors, 0 changes
58      public void MoveEnemy ()
59      {
60          //Declare variables for X and Y axis move directions, these range from -1 to 1.
61          //These values allow us to choose between the cardinal directions: up, down, left and right.
62          int xDir = 0;
63          int yDir = 0;
64
65          //If the difference in positions is approximately zero (Epsilon) do the following:
66          if(Mathf.Abs (target.position.x - transform.position.x) < float.Epsilon)
67
68              //If the y coordinate of the target's (player) position is greater than the y coordinate of this en
69              yDir = target.position.y > transform.position.y ? 1 : -1;
70
71          //If the difference in positions is not approximately zero (Epsilon) do the following:
72          else
73              //Check if target x position is greater than enemy's x position, if so set x direction to 1 (move r
74              xDir = target.position.x > transform.position.x ? 1 : -1;
75
76          //Call the AttemptMove function and pass in the generic parameter Player, because Enemy is moving and e
77          AttemptMove <Player> (xDir, yDir);
78      }
```

8. Open GameManager.cs script and uncomment enemies[i].MoveEnemy() in the MoveEnemies method, line 168.

```
164         //Loop through List of Enemy objects.
165         for (int i = 0; i < enemies.Count; i++)
166         {
167             //Call the MoveEnemy function of Enemy at index i in the enemies List.
168             enemies[i].MoveEnemy ();
169
170             //Wait for Enemy's moveTime before moving next Enemy,
171             yield return new WaitForSeconds(enemies[i].moveTime);
172         }
173         //Once Enemies are done moving, set playersTurn to true so player can move.
```

9. Open BoardManager script and uncomment LayoutObjectAtRandom (enemyTiles, enemyCount, enemyCount) in SetupScene method, line 150.

```
131     //SetupScene initializes our level and calls the previous functions to lay out the game board
        1 reference | 0 changes | 0 authors, 0 changes
132     public void SetupScene (int level)
133     {
134         //Creates the outer walls and floor.
135         BoardSetup ();
136
137         //Reset our list of gridpositions.
138         InitialiseList ();
139
140         //Instantiate a random number of wall tiles based on minimum and maximum, at randomized positions.
141         LayoutObjectAtRandom (wallTiles, wallCount.minimum, wallCount.maximum);
142
143         //Instantiate a random number of food tiles based on minimum and maximum, at randomized positions.
144         LayoutObjectAtRandom (foodTiles, foodCount.minimum, foodCount.maximum);
145
146         //Determine number of enemies based on current level number, based on a logarithmic progression
147         int enemyCount = (int)Mathf.Log(level, 2f);
148
149         //Instantiate a random number of enemies based on minimum and maximum, at randomized positions.
150         LayoutObjectAtRandom (enemyTiles, enemyCount, enemyCount);
151
152         //Instantiate the exit tile in the upper right hand corner of our game board
153         Instantiate (exit, new Vector3 (columns - 1, rows - 1, 0f), Quaternion.identity);
154     }
```

Unity 2D Rogue - Lesson 4 - Add Music

1. Open Loader Script in Visual Studio
2. Uncomment Sound Manager variable, line 9.

```
        0 references | 0 changes | 0 authors, 0 changes
6   public class Loader : MonoBehaviour
7   {
8       public GameObject gameManager;          //GameManager prefab to instantiate.
9       public GameObject soundManager;         //SoundManager prefab to instantiate.
10
```

3. Uncomment if statement instanciating soundmanager.

```
20          //Check if a SoundManager has already been assigned to static variable GameManager.instance or if it's still null
21          if (SoundManager.instance == null)
22
23              //Instantiate SoundManager prefab
24              Instantiate(soundManager);
25          }
26      }
```

4. Open Enemy.cs script in Visual Studio
5. Uncomment SoundManager.instance.RandomizeSfx (attackSound1, attackSound2) in the OnCantMove method, line 95.

```
81      //OnCantMove is called if Enemy attempts to move into a space occupied by a Player, it overrides the OnCantMove function of MovingObject
82      //and takes a generic parameter T which we use to pass in the component we expect to encounter, in this case Player
        3 references | 0 changes | 0 authors, 0 changes
83      protected override void OnCantMove <T> (T component)
84      {
85          //Declare hitPlayer and set it to equal the encountered component.
86          Player hitPlayer = component as Player;
87
88          //Call the LoseFood function of hitPlayer passing it playerDamage, the amount of foodpoints to be subtracted.
89          hitPlayer.LoseFood (playerDamage);
90
91          //Set the attack trigger of animator to trigger Enemy attack animation.
92          animator.SetTrigger ("enemyAttack");
93
94          //Call the RandomizeSfx function of SoundManager passing in the two audio clips to choose randomly between.
95          SoundManager.instance.RandomizeSfx (attackSound1, attackSound2);
96      }
```

6. Open Player.cs script in Visual Studio.
7. Uncomment SoundManager.instance.RandomizeSfx (moveSound1, moveSound2) in the AttemptMove method, line 149.

```
//AttemptMove overrides the AttemptMove function in the base class MovingObject
//AttemptMove takes a generic parameter T which for Player will be of the type Wall, it also takes integers for x and y direction to move in.
6 references | 0 changes | 0 authors, 0 changes
protected override void AttemptMove <T> (int xDir, int yDir)
{
    //Every time player moves, subtract from food points total.
    food--;

    //Update food text display to reflect current score.
    foodText.text = "Food: " + food;

    //Call the AttemptMove method of the base class, passing in the component T (in this case Wall) and x and y direction to move.
    base.AttemptMove <T> (xDir, yDir);

    //Hit allows us to reference the result of the Linecast done in Move.
    RaycastHit2D hit;

    //If Move returns true, meaning Player was able to move into an empty space.
    if (Move (xDir, yDir, out hit))
    {
        //Call RandomizeSfx of SoundManager to play the move sound, passing in two audio clips to choose from.
        SoundManager.instance.RandomizeSfx (moveSound1, moveSound2);
    }

    //Since the player has moved and lost food points, check if the game has ended.
    CheckIfGameOver ();

    //Set the playersTurn boolean of GameManager to false now that players turn is over.
    GameManager.instance.playersTurn = false;
}
```

8. In Player.cs script uncomment SoundManager.instance.RandomizeSfx (eatSound1, eatSound2) in the OnTriggerEnter2D method, line 198.

```
private void OnTriggerEnter2D (Collider2D other)
{
    //Check if the tag of the trigger collided with is Exit.
    if(other.tag == "Exit")
    {
        //Invoke the Restart function to start the next level with a delay of restartLevelDelay (default 1 second).
        Invoke ("Restart", restartLevelDelay);

        //Disable the player object since level is over.
        enabled = false;
    }

    //Check if the tag of the trigger collided with is Food.
    else if(other.tag == "Food")
    {
        //Add pointsPerFood to the players current food total.
        food += pointsPerFood;

        //Update foodText to represent current total and notify player that they gained points
        foodText.text = "+" + pointsPerFood + " Food: " + food;

        //Call the RandomizeSfx function of SoundManager and pass in two eating sounds to choose between to play the eating sound effect.
        SoundManager.instance.RandomizeSfx (eatSound1, eatSound2);

        //Disable the food object the player collided with.
        other.gameObject.SetActive (false);
    }
```

9. In Player.cs script uncomment SoundManager.instance.RandomizeSfx (drinkSound1, drinkSound2) in the OnTriggerEnter2D method, line 214.

```
    //Check if the tag of the trigger collided with is Soda.
    else if(other.tag == "Soda")
    {
        //Add pointsPerSoda to players food points total
        food += pointsPerSoda;

        //Update foodText to represent current total and notify player that they gained points
        foodText.text = "+" + pointsPerSoda + " Food: " + food;

        //Call the RandomizeSfx function of SoundManager and pass in two drinking sounds to choose between to play the drinking sound effect.
        SoundManager.instance.RandomizeSfx (drinkSound1, drinkSound2);

        //Disable the soda object the player collided with.
        other.gameObject.SetActive (false);
    }
}
```

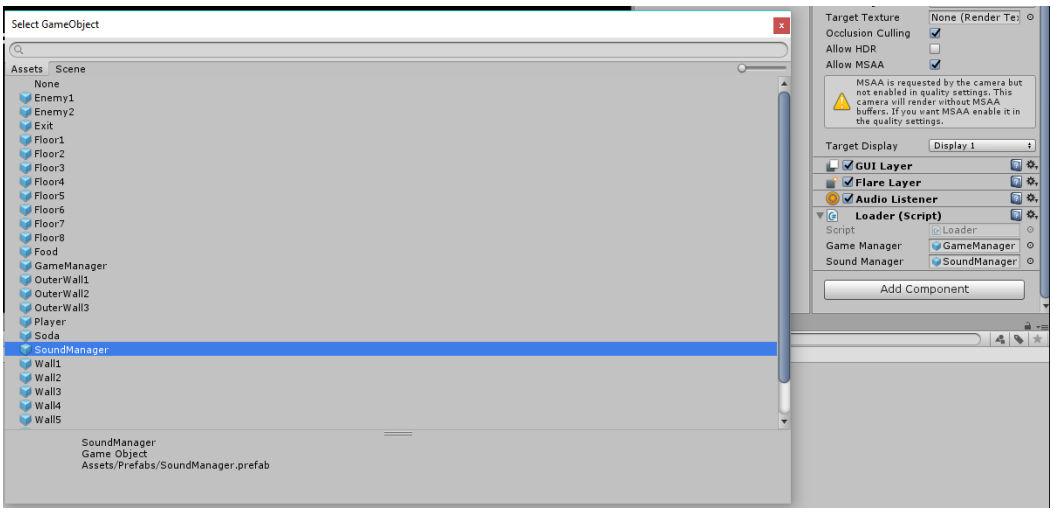
10. In Player.cs script uncomment SoundManager.instance.PlaySingle (gameOverSound) and SoundManager.instance.musicSource.Stop() in the CheckIfGameOver method, line 256 and 259.

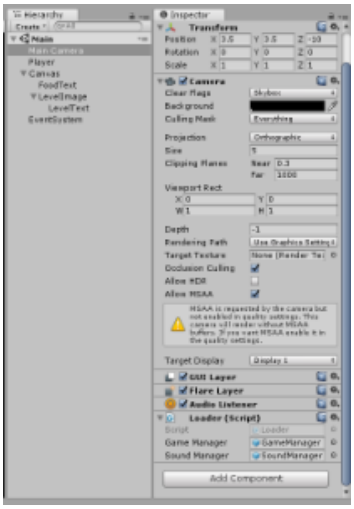
```
249         //CheckIfGameOver checks if the player is out of food points and if so, ends the game.
            2 references | 0 changes | 0 authors, 0 changes
250         private void CheckIfGameOver ()
251         {
252             //Check if food point total is less than or equal to zero.
253             if (food <= 0)
254             {
255                 //Call the PlaySingle function of SoundManager and pass it the gameOverSound as the audio clip to play.
256                 SoundManager.instance.PlaySingle (gameOverSound);
257
258                 //Stop the background music.
259                 SoundManager.instance.musicSource.Stop();
260
261                 //Call the GameOver function of GameManager.
262                 GameManager.instance.GameOver ();
263             }
264         }
265     }
```

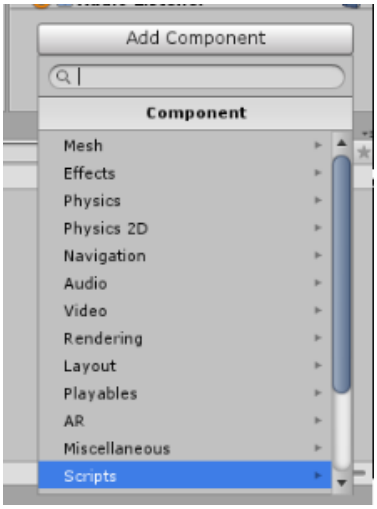
11. On Wall.cs script in Visual Studio
12. Uncomment SoundManager.instance.RandomizeSfx (chopSound1, chopSound2) in method DamageWall, line 28.

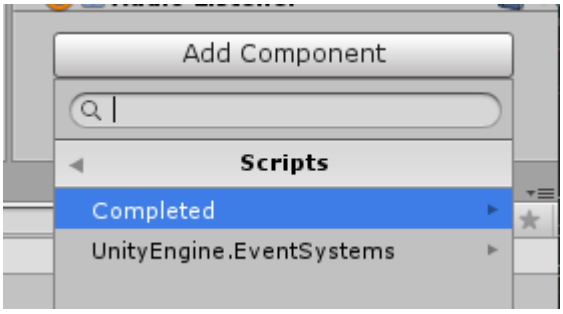
```
24          //DamageWall is called when the player attacks a wall.
            1 reference | 0 changes | 0 authors, 0 changes
25          public void DamageWall (int loss)
26          {
27              //Call the RandomizeSfx function of SoundManager to play one of two chop sounds.
28              SoundManager.instance.RandomizeSfx (chopSound1, chopSound2);
29
30              //Set spriteRenderer to the damaged wall sprite.
31              spriteRenderer.sprite = dmgSprite;
32
33              //Subtract loss from hit point total.
34              hp -= loss;
35
36              //If hit points are less than or equal to zero:
37              if(hp <= 0)
38                  //Disable the gameObject.
39                  gameObject.SetActive (false);
40          }
```

13. On Unity and select Main Camera.
14. Select Loader (Script) on the Main Camera.
15. Update the Sound Manager to use SoundManager Asset.



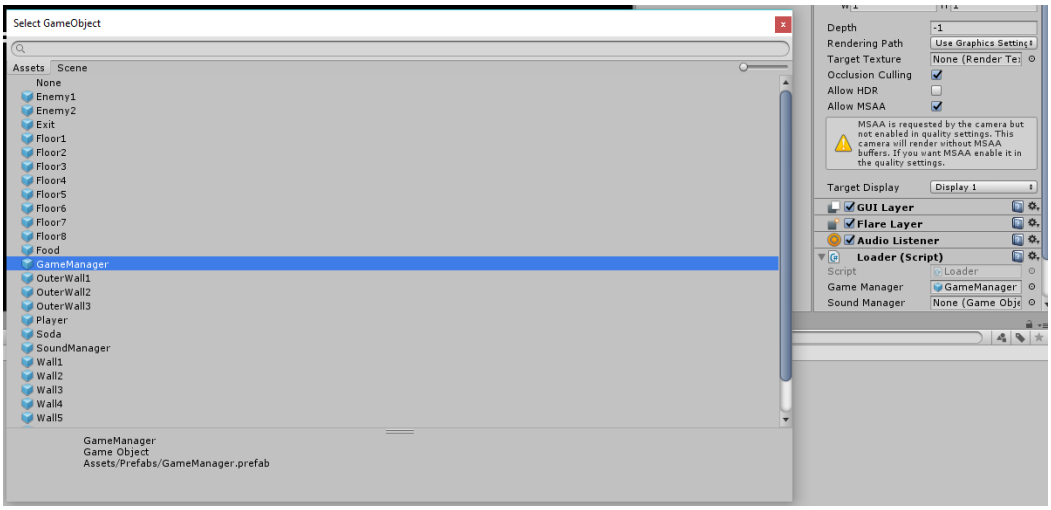



Add Component
Component
Mesh
Effects
Physics
Physics 2D
Navigation
Audio
Video
Rendering
Layout
Playables
AR
Miscellaneous
Scripts


Add Component
Scripts
Completed
UnityEngine.EventSystems


Add Component
Completed
Board Manager
Enemy
Game Manager
Loader
Player
Sound Manager
Wall

16. Update the Loader (Script) properties



17.

Unity 2D UFO Game

Original Project

Play Game: https://lukearmstrong.github.io/games/unity-tutorial-2d-ufo-project/

Tutorial: https://unity3d.com/learn/tutorials/projects/2d-ufo-tutorial

Source: https://github.com/lukearmstrong/unity-tutorial-2d-ufo-project

Lessons

- Unity 2D UFO Lesson 1 - Controlling the Player
- Unity 2D UFO Lesson 2 - Adding Collision
- Unity 2D UFO Lesson 3 - Picking Up Collectables
- Unity 2D UFO Lesson 4 - Counting Collectables and Displaying Score

Unity 2D UFO Lesson 1 - Controlling the Player

Lesson

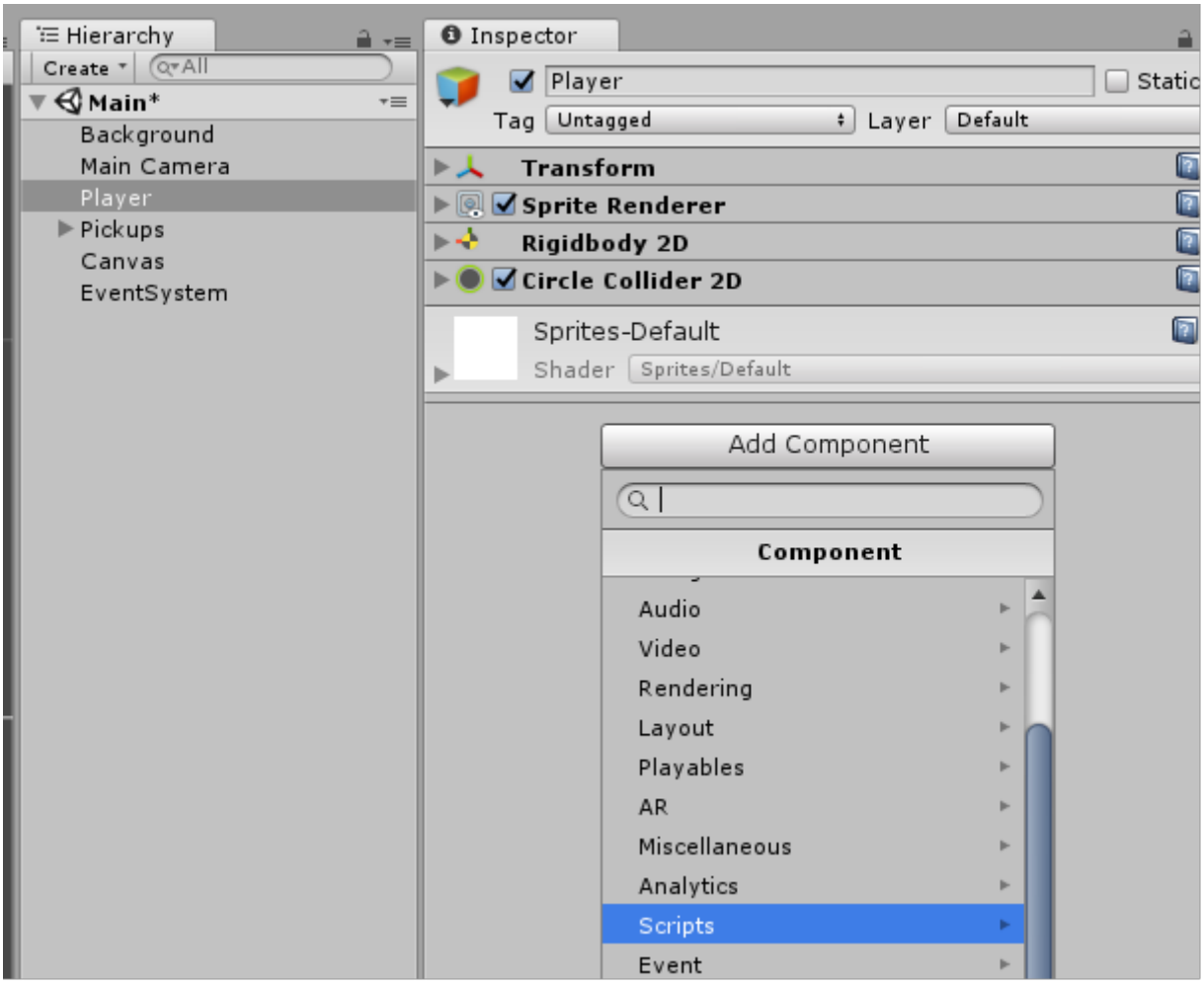
1. Open PlayerController script file in Visual Studio.
2. Uncomment speed public variable.

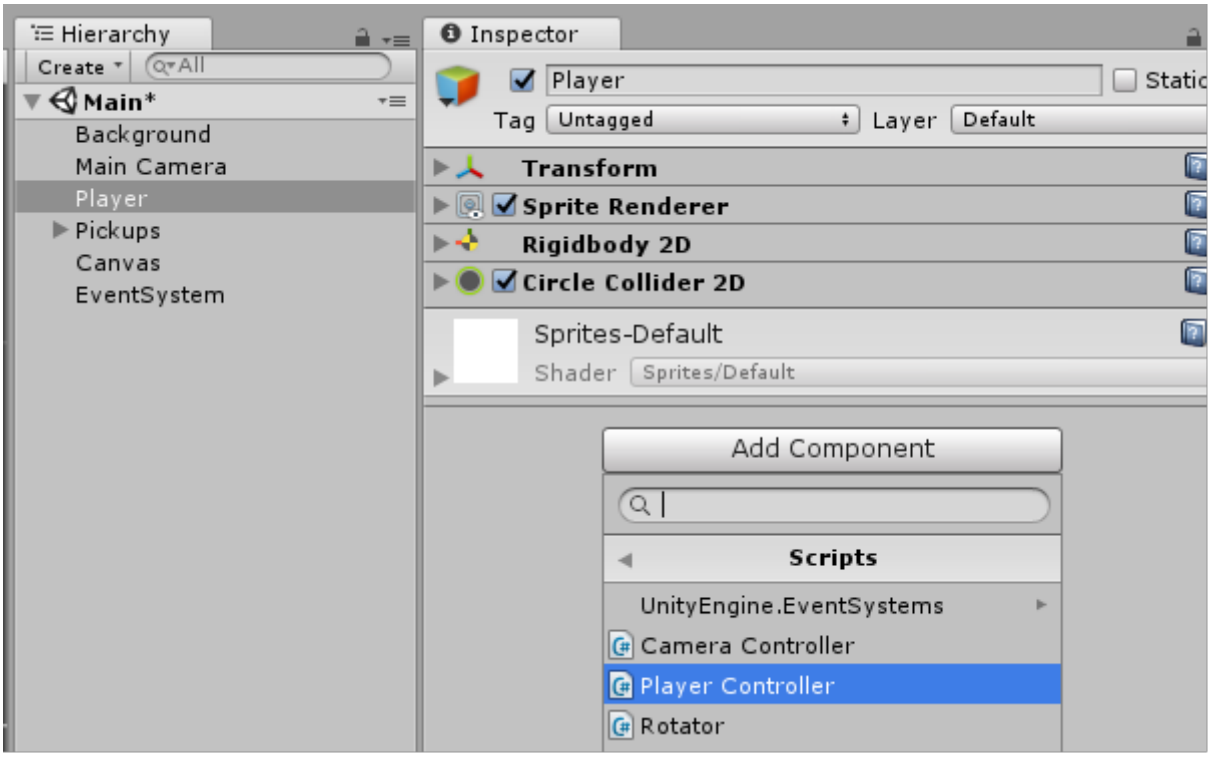
```
0 references | 0 changes | 0 authors, 0 changes
public class PlayerController : MonoBehaviour {

    public float speed;             //Floating point variable to store the player's movement speed.
    //public Text countText;            //Store a reference to the UI Text component which will display the number of pickups collected.
    //public Text winText;          //Store a reference to the UI Text component which will display the 'You win' message.
```

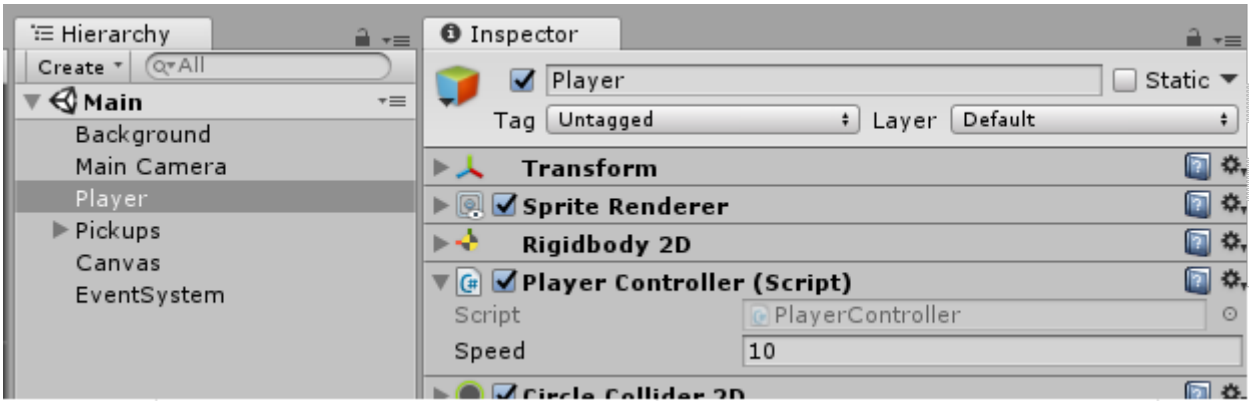
3. Uncomment FixedUpdate method.

```
32      //FixedUpdate is called at a fixed interval and is independent of frame rate. Put physics code here.
        0 references | 0 changes | 0 authors, 0 changes
33      void FixedUpdate()
34      {
35          //Store the current horizontal input in the float moveHorizontal.
36          float moveHorizontal = Input.GetAxis ("Horizontal");
37
38          //Store the current vertical input in the float moveVertical.
39          float moveVertical = Input.GetAxis ("Vertical");
40
41          //Use the two store floats to create a new Vector2 variable movement.
42          Vector2 movement = new Vector2 (moveHorizontal, moveVertical);
43
44          //Call the AddForce function of our Rigidbody2D rb2d supplying movement multiplied by speed to move our player.
45          rb2d.AddForce (movement * speed);
46      }
```

4. Add PlayerController script component to Player game object. Set Speed to 10.



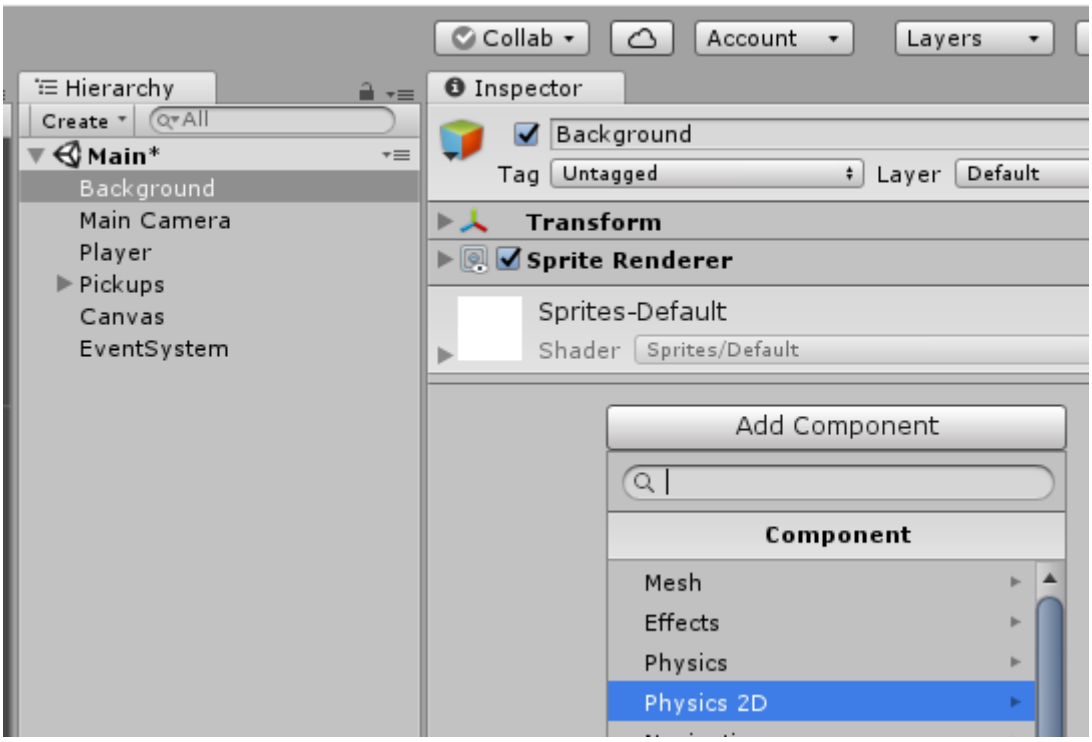


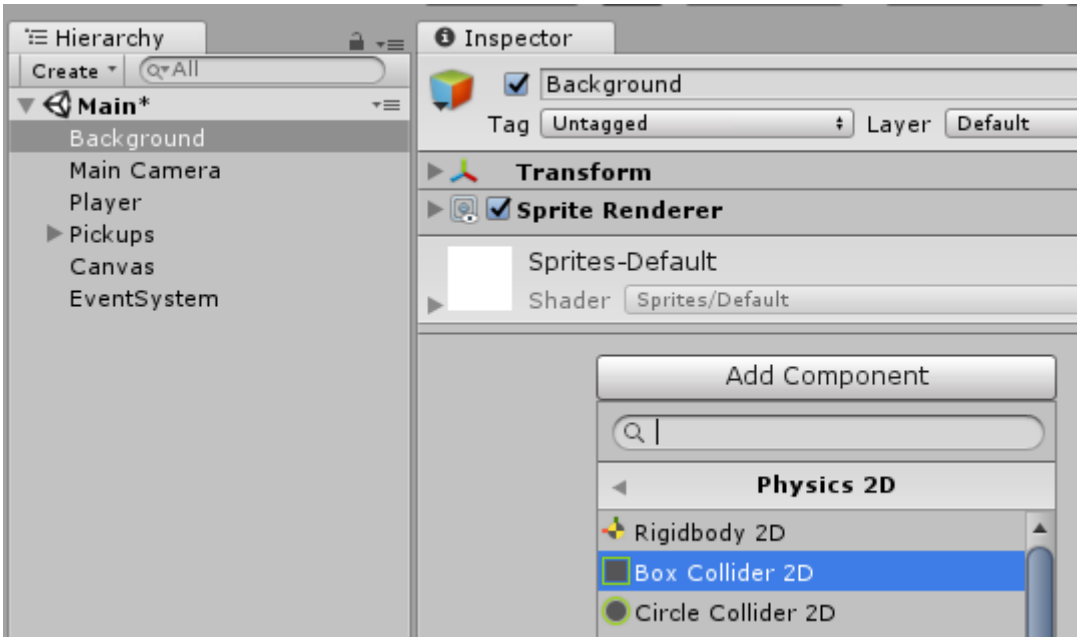


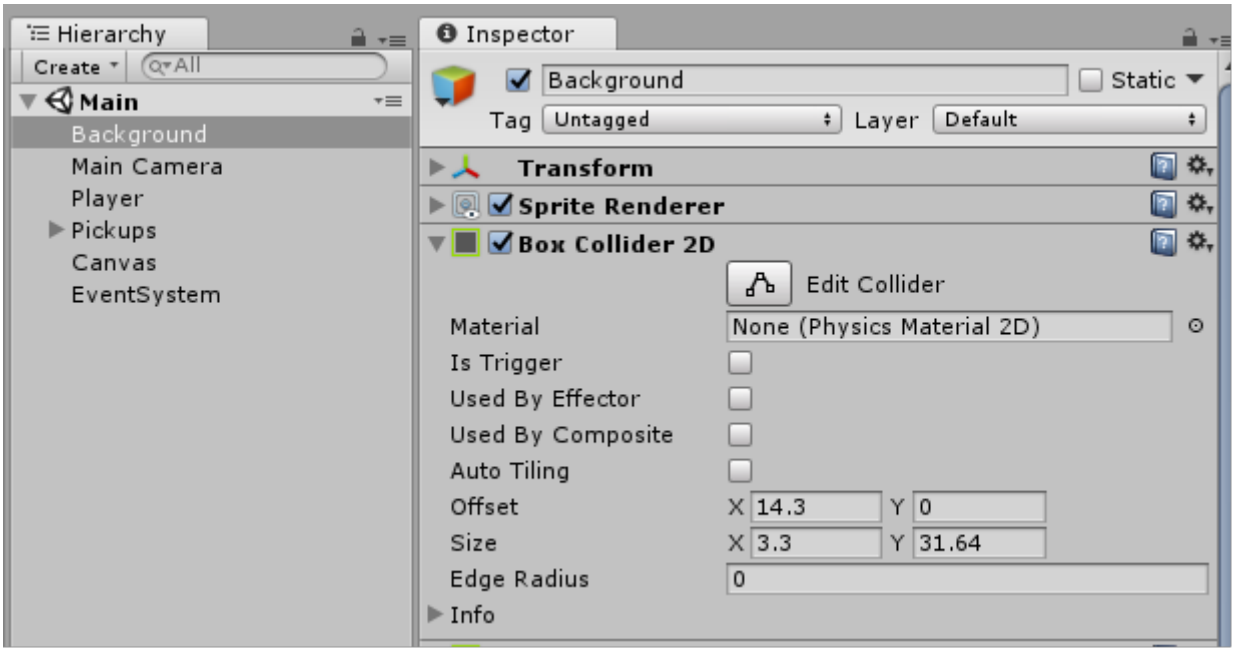
Challenge

Unity 2D UFO Lesson 2 - Adding Collision

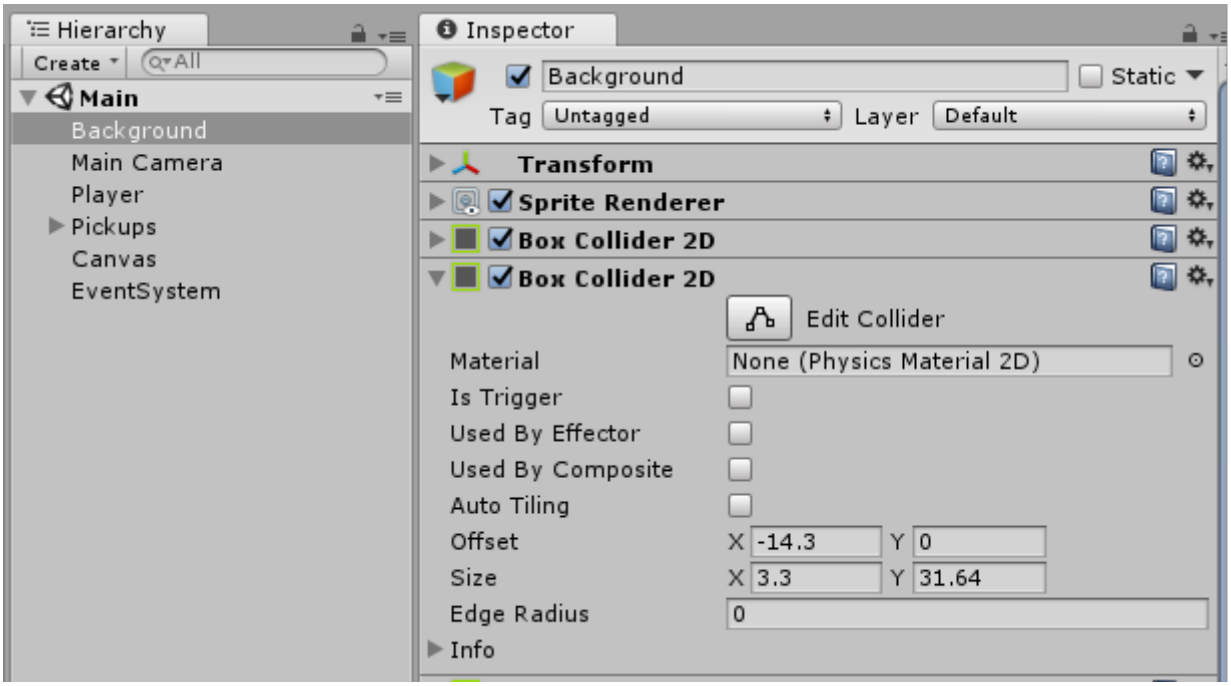
1. In Unity, select the Background game object and add a Box Collider 2D.
   Set Offset X to 14.3
   Set Offset Y to 0
   Set Size X to 3.3
   Set Size Y to 31.64


Collab
Account
Layers
Hierarchy
Create
All
Main*
Background
Main Camera
Player
Pickups
Canvas
EventSystem
Inspector
Background
Tag
Untagged
Layer
Default
Transform
Sprite Renderer
Sprites-Default
Shader
Sprites/Default
Add Component
Component
Mesh
Effects
Physics
Physics 2D

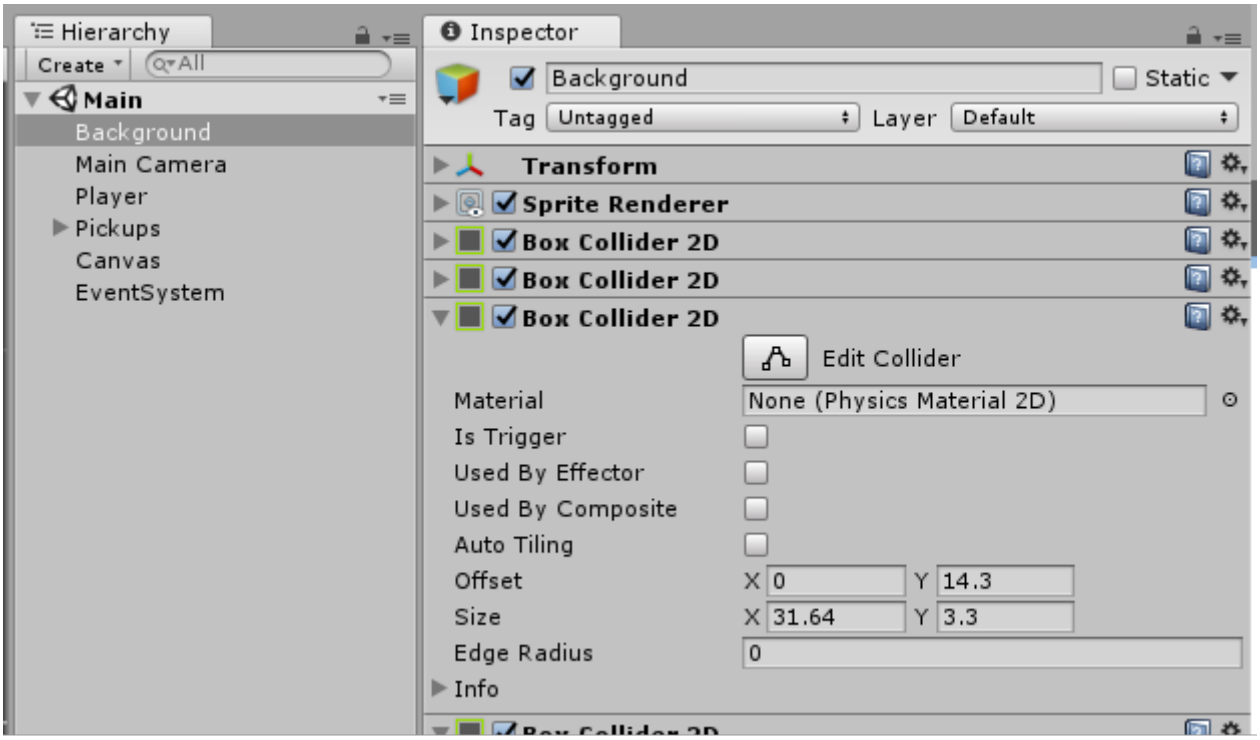

Hierarchy
Create
All
Main*
Background
Main Camera
Player
Pickups
Canvas
EventSystem
Inspector
Background
Tag
Untagged
Layer
Default
Transform
Sprite Renderer
Sprites-Default
Shader
Sprites/Default
Add Component
Physics 2D
Rigidbody 2D
Box Collider 2D
Circle Collider 2D



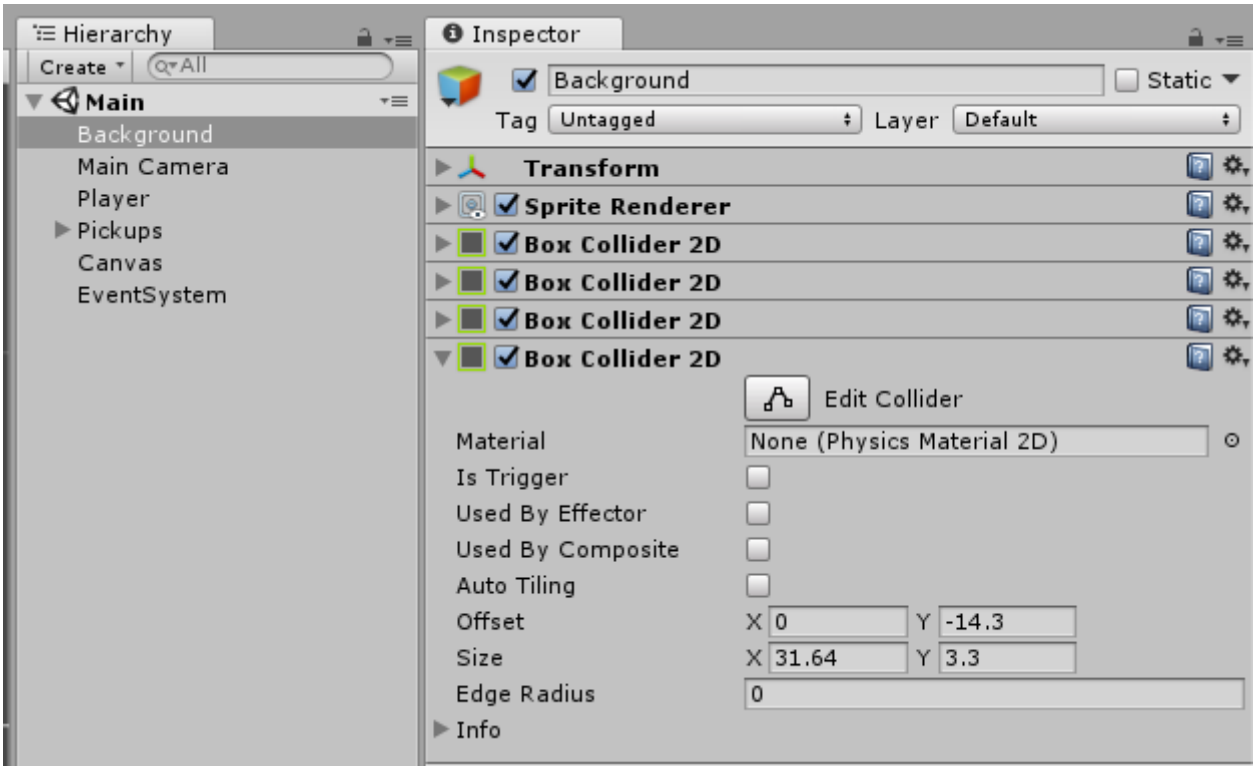
2. Add a second Box Collider 2D.
   Set Offset X to -14.3
   Set Offset Y to 0
   Set Size X to 3.3
   Set Size Y to 31.64



3. Add a third Box Collider 2D.
   Set Offset X to 0
   Set Offset Y to 14.3
   Set Size X to 31.64
   Set Size Y to 3.3



4. Add a forth Box Collider 2D.
   Set Offset X to 0
   Set Offset Y to -14.3
   Set Size X to 31.64
   Set Size Y to 3.3

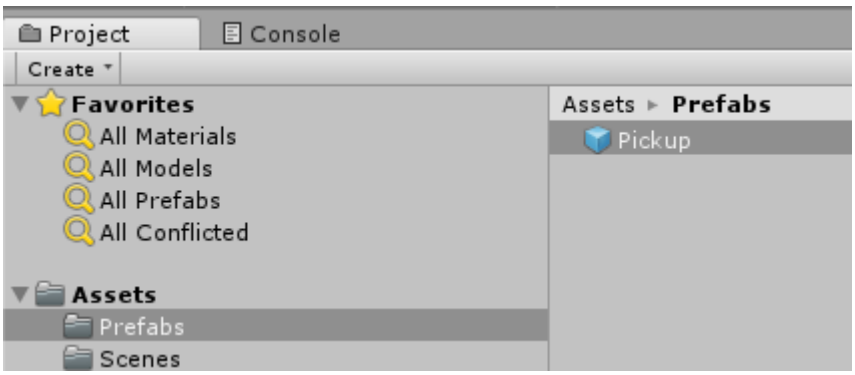


Unity 2D UFO Lesson 3 - Picking Up Collectables

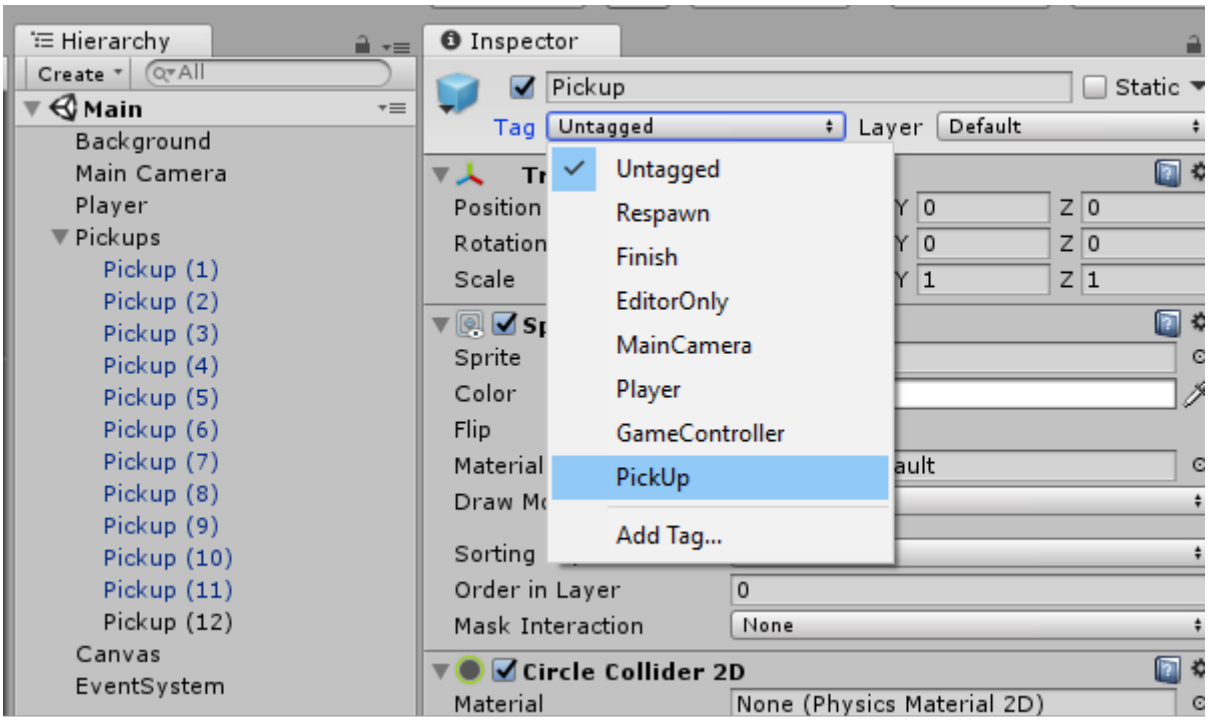
1. Open PlayerController script in Visual Studio.
2. Uncomment the if statement in the OnTriggerEnter2D method.

```
48      //OnTriggerEnter2D is called whenever this object overlaps with a trigger collider.
        0 references | 0 changes | 0 authors, 0 changes
49      void OnTriggerEnter2D(Collider2D other)
50      {
51          //Check the provided Collider2D parameter other to see if it is tagged "PickUp", if it is...
52          if (other.gameObject.CompareTag ("PickUp"))
53          {
54              //... then set the other object we just collided with to inactive.
55              other.gameObject.SetActive(false);
56
57              //Add one to the current value of our count variable.
58              count = count + 1;
59
60              //Update the currently displayed count by calling the SetCountText function.
61              //SetCountText ();
62          }
63
64
65      }
```

3. Assign PickUp tag to Pickup Prefabs, select the Pickup Prefab.



4. Change Tag to PickUp.



5. Play Game.

Unity 2D UFO Lesson 4 - Counting Collectables and Displaying Score

1. Open PlayerController script in Visual Studio
2. Uncomment countText and winText public variables.

```
0 references | 0 changes | 0 authors, 0 changes
public class PlayerController : MonoBehaviour {

    public float speed;             //Floating point variable to store the player's movement speed.
    public Text countText;          //Store a reference to the UI Text component which will display the number of pickups collected.
    public Text winText;            //Store a reference to the UI Text component which will display the 'You win' message.

    private Rigidbody2D rb2d;       //Store a reference to the Rigidbody2D component required to use 2D Physics.
    private int count;              //Integer to store the number of pickups collected so far.
```

3. Uncomment setting of winText.text property to an empty string at the start of the game, line 26.

```
    // Use this for initialization
    0 references | 0 changes | 0 authors, 0 changes
    void Start()
    {
        //Get and store a reference to the Rigidbody2D component so that we can access it.
        rb2d = GetComponent<Rigidbody2D> ();

        //Initialize count to zero.
        count = 0;

        //Initialze winText to a blank string since we haven't won yet at beginning.
        winText.text = "";

        //Call our SetCountText function which will update the text with the current value for count.
        SetCountText ();
    }
```

4. Uncomment Set Count Text method.

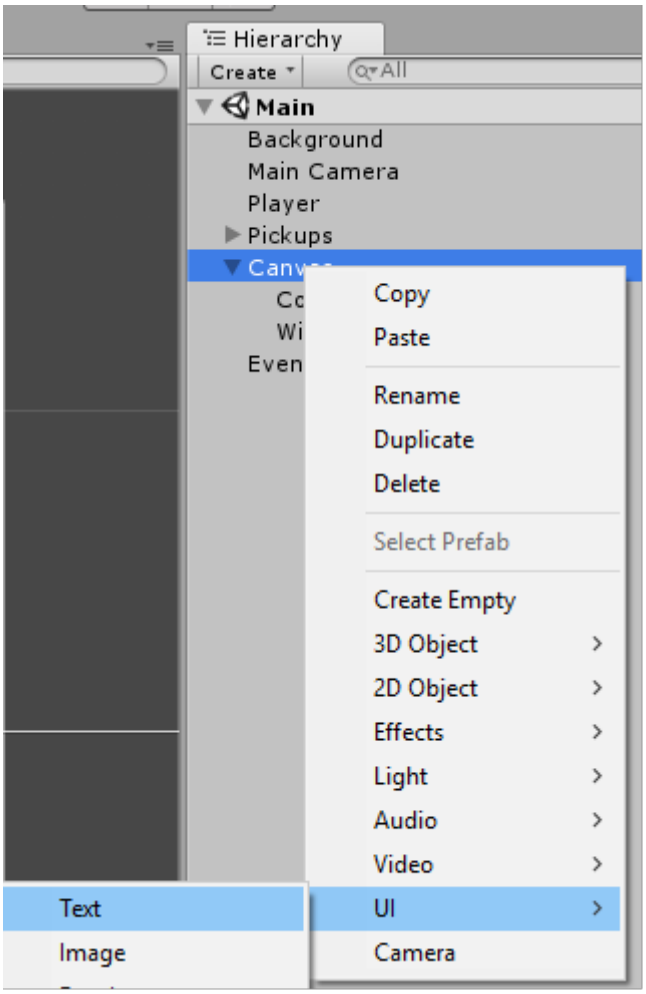
```
//This function updates the text displaying the number of objects we've collected and displays our victory message if we've collected all of them.
2 references | 0 changes | 0 authors, 0 changes
void SetCountText()
{
    //Set the text property of our our countText object to "Count: " followed by the number stored in our count variable.
    countText.text = "Count: " + count.ToString ();

    //Check if we've collected all 12 pickups. If we have...
    if (count >= 12)
        //... then set the text property of our winText object to "You win!"
        winText.text = "You win!";
}
```

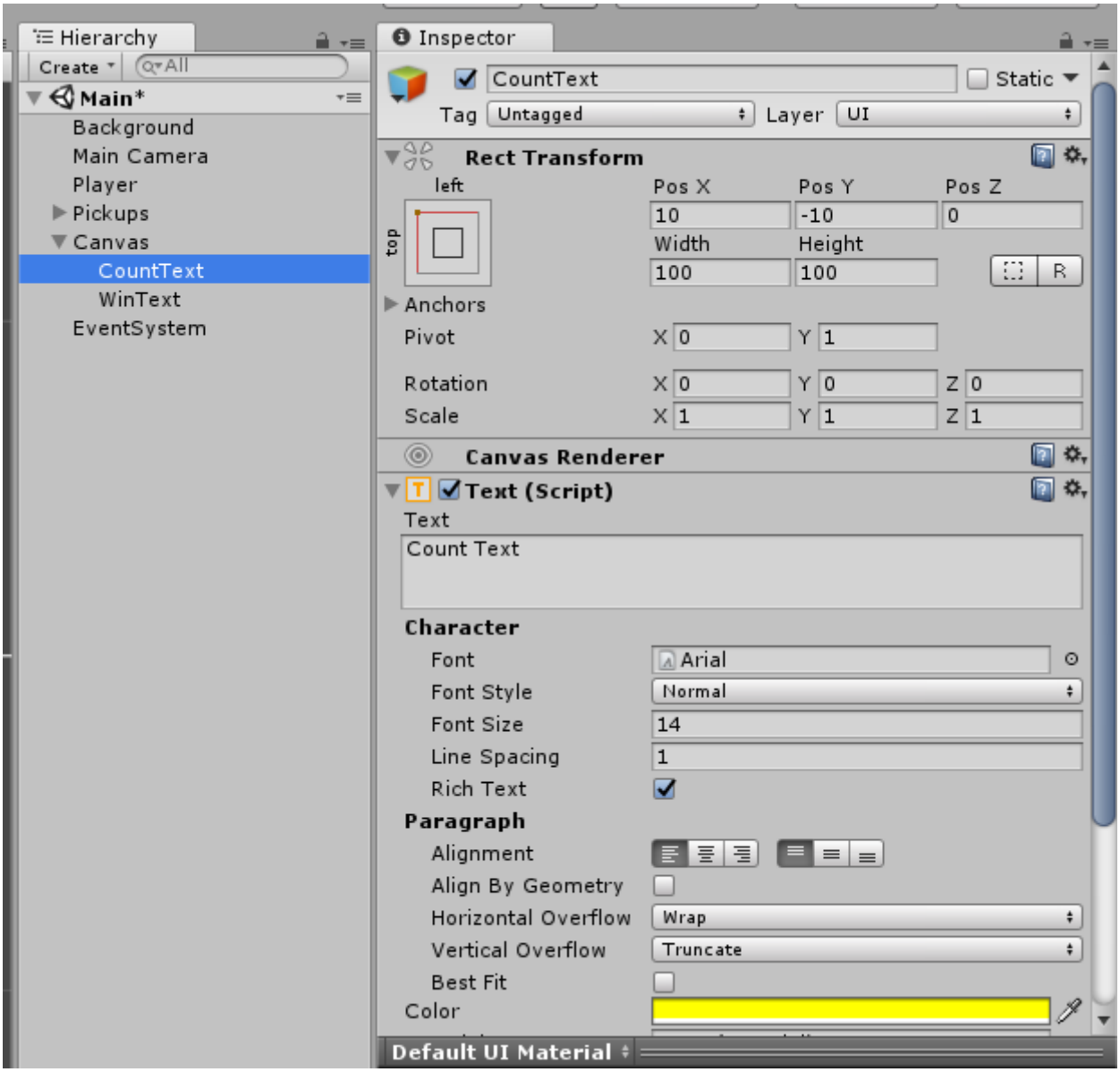
5. Uncomment calling of the SetCountText method in the Start method and the OnTriggerEnter2D method , line 61.

```
    // Use this for initialization
    0 references | 0 changes | 0 authors, 0 changes
    void Start()
    {
        //Get and store a reference to the Rigidbody2D component so that we can access it.
        rb2d = GetComponent<Rigidbody2D> ();

        //Initialize count to zero.
        count = 0;

        //Initialze winText to a blank string since we haven't won yet at beginning.
        //winText.text = "";

        //Call our SetCountText function which will update the text with the current value for count.
        SetCountText ();
    }
```

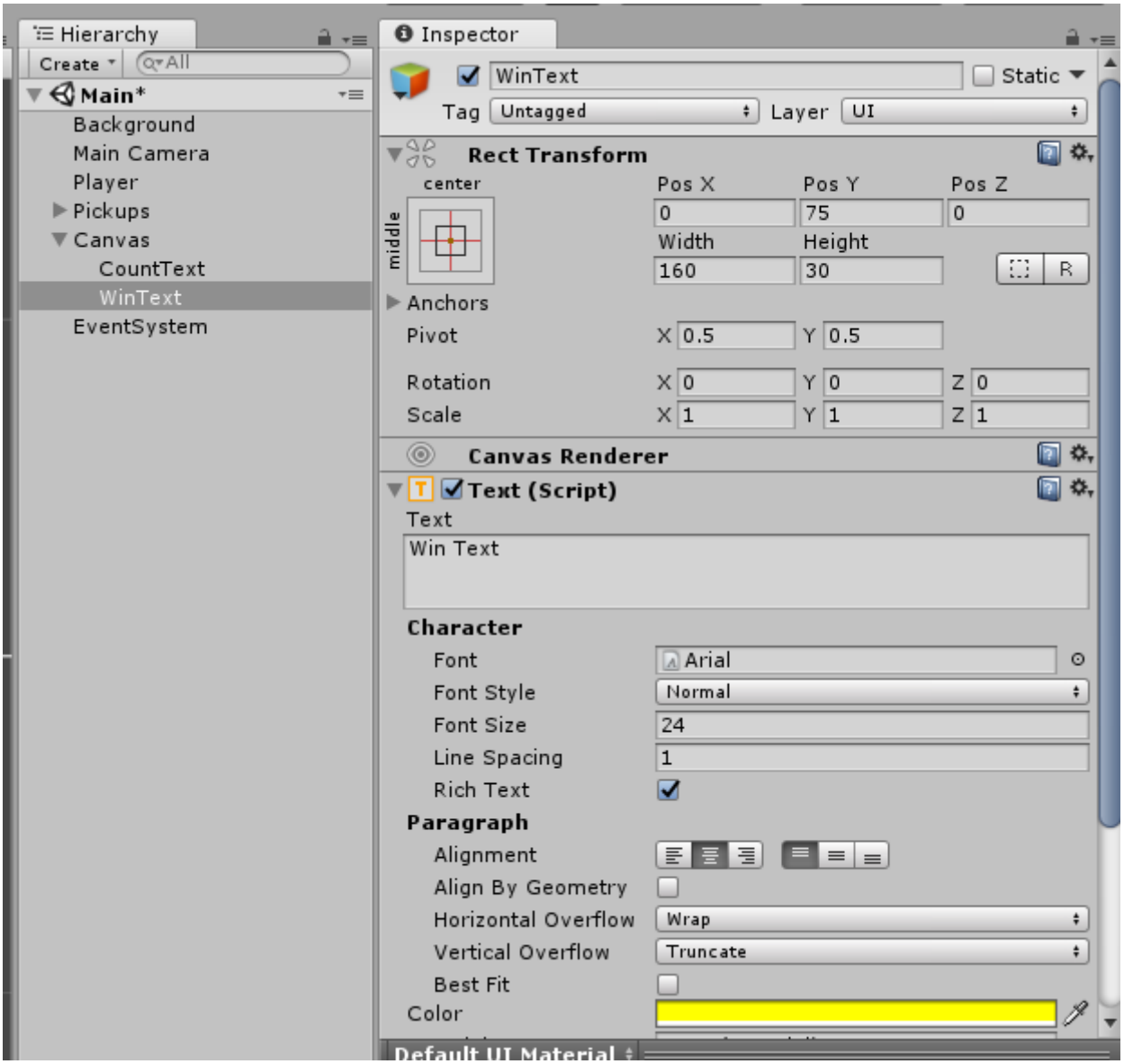
```
48      //OnTriggerEnter2D is called whenever this object overlaps with a trigger collider.
        0 references | 0 changes | 0 authors, 0 changes
49      void OnTriggerEnter2D(Collider2D other)
50      {
51          //Check the provided Collider2D parameter other to see if it is tagged "PickUp", if it is...
52          if (other.gameObject.CompareTag ("PickUp"))
53          {
54              //... then set the other object we just collided with to inactive.
55              other.gameObject.SetActive(false);
56
57              //Add one to the current value of our count variable.
58              count = count + 1;
59
60              //Update the currently displayed count by calling the SetCountText function.
61              SetCountText ();
62          }
```

6. Switch to Unity. Add a new Text Game Object to the Canvas and rename it to CountText

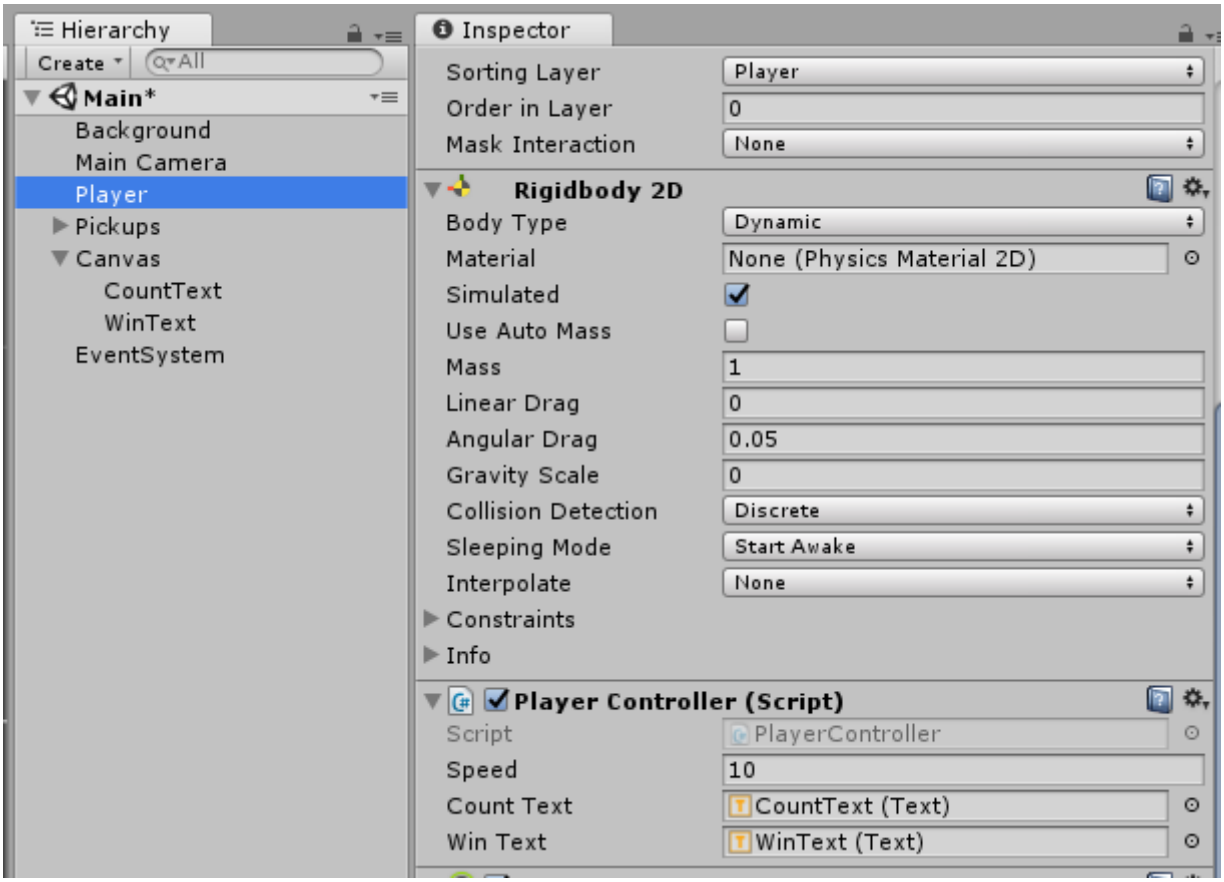




7. Add a new Text Game Object to the Canvas and rename it to WinText



8. Update Player Controller Script properties with CountText and WinText game objects.



Unity Lander Game

These lessons are based on the Playground Project developed by Ciro Continisio. Deviate from the lessons as you see appropriate.

Playground Project - https://blogs.unity3d.com/2016/09/20/teaching-unity-to-non-programmers-playground-project/

Source Code - https://github.com/jasonweibel/HyperStream-Unity

- Unity Lander Lesson 1 - Setup Scene
- Unity Lander Lesson 2 - Setup Player Movements
- Unity Lander Lesson 3 - Add Destroy Action Scripts

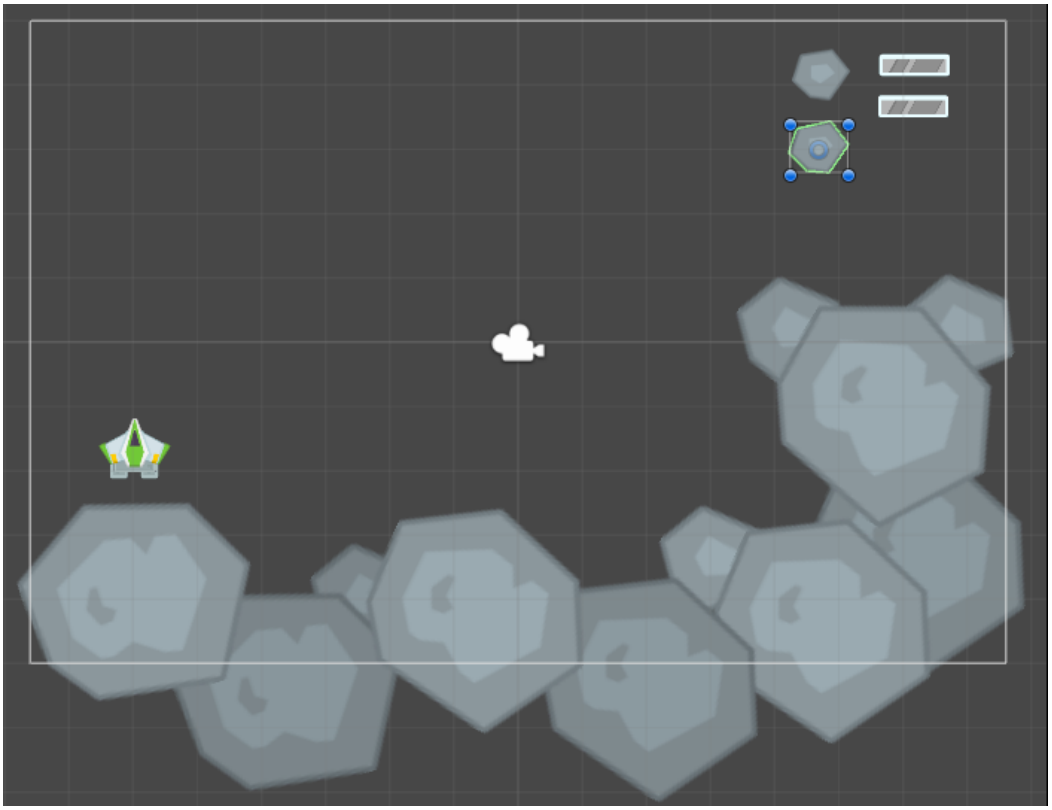
Unity Lander Lesson 1 - Setup Scene

Reference Tutorials

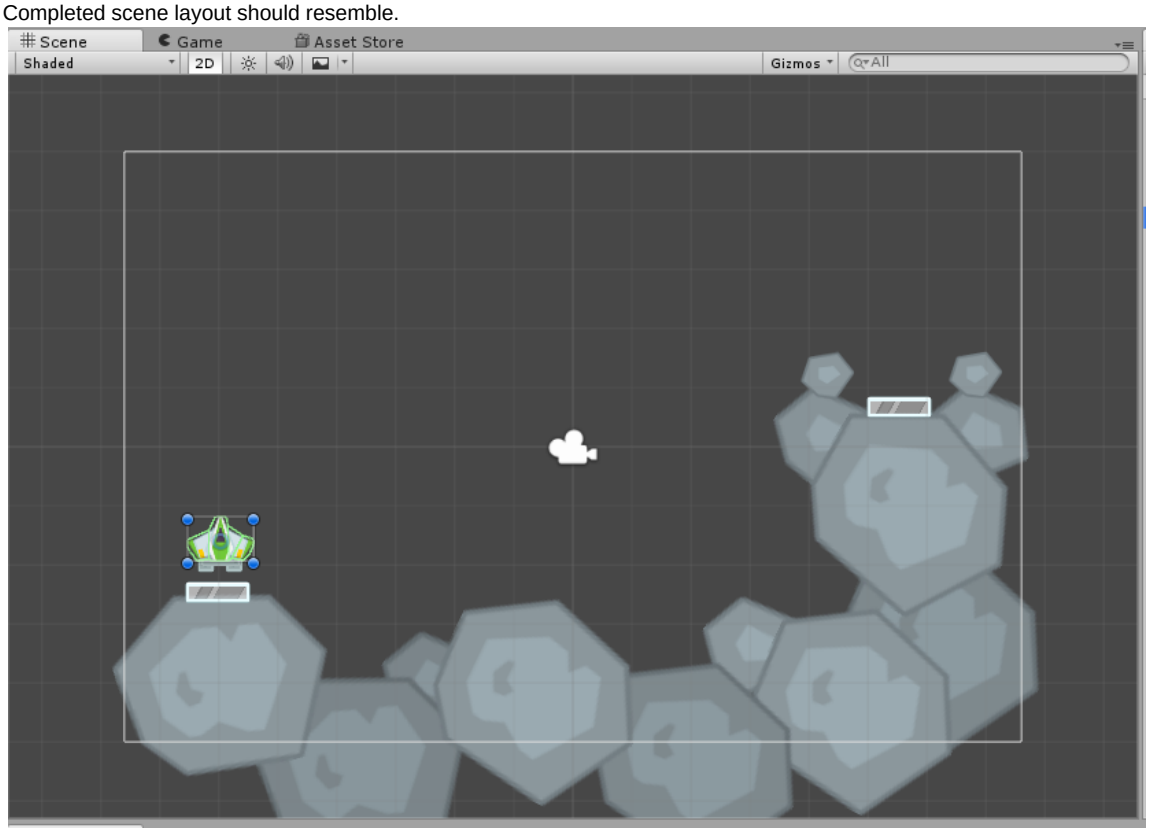
The Scene View - https://unity3d.com/learn/tutorials/topics/interface-essentials/scene-view?playlist=17090

Lesson

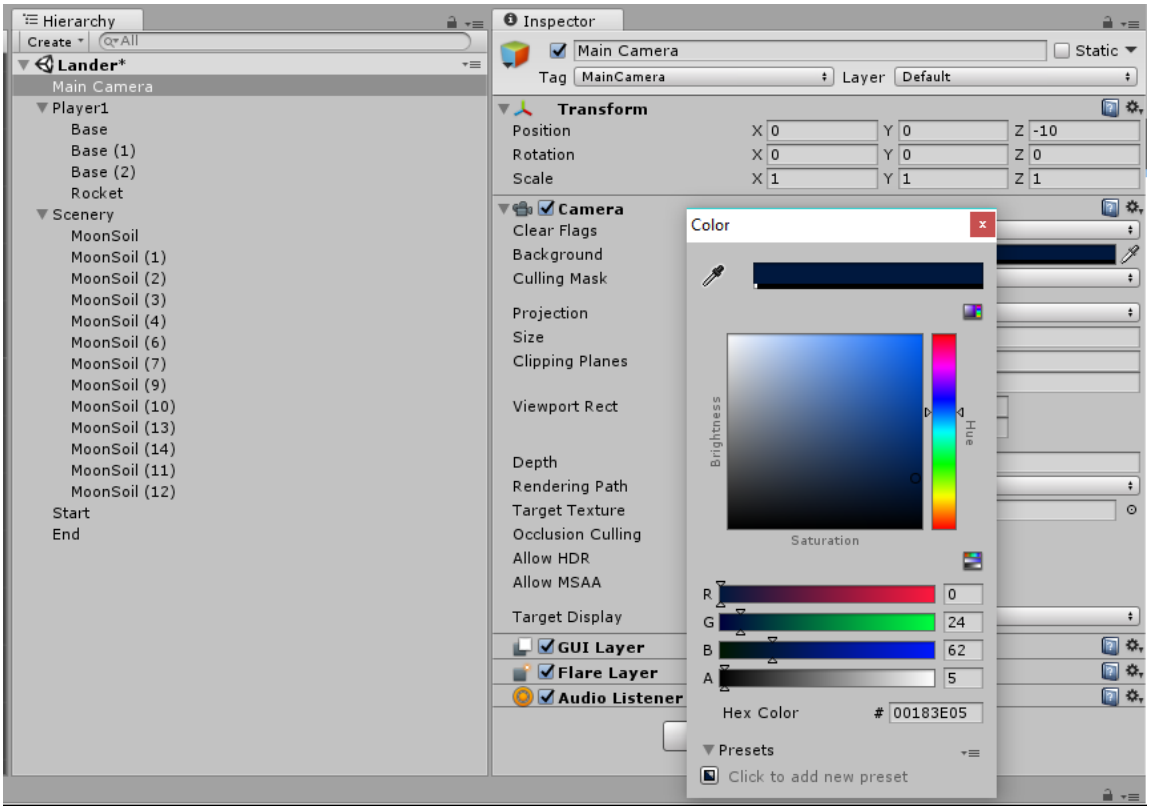
1. Launch Unity and open 'HyperStream-Unity\Lander\Lesson 1 - Setup Scene' project.
2. Start game and watch the rocket fall off the screen.
3. Move rocks into place.
4. Change size of rocks.
5. Move the start and end platform, make sure the starting platform is below the rocket.

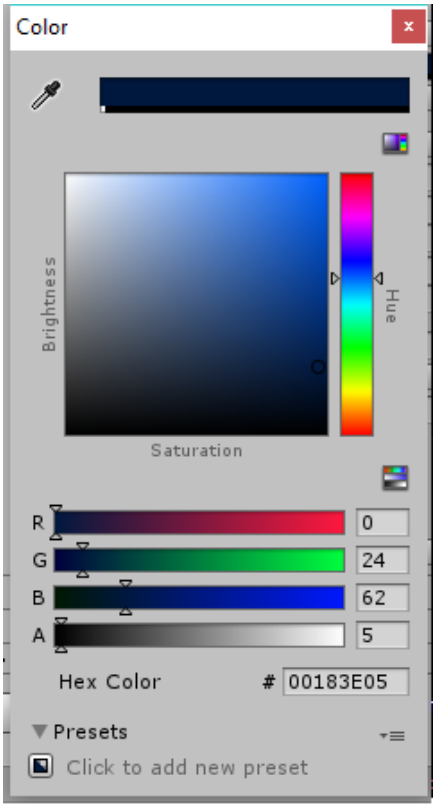


6. Completed scene layout should resemble.



7. Change sky color

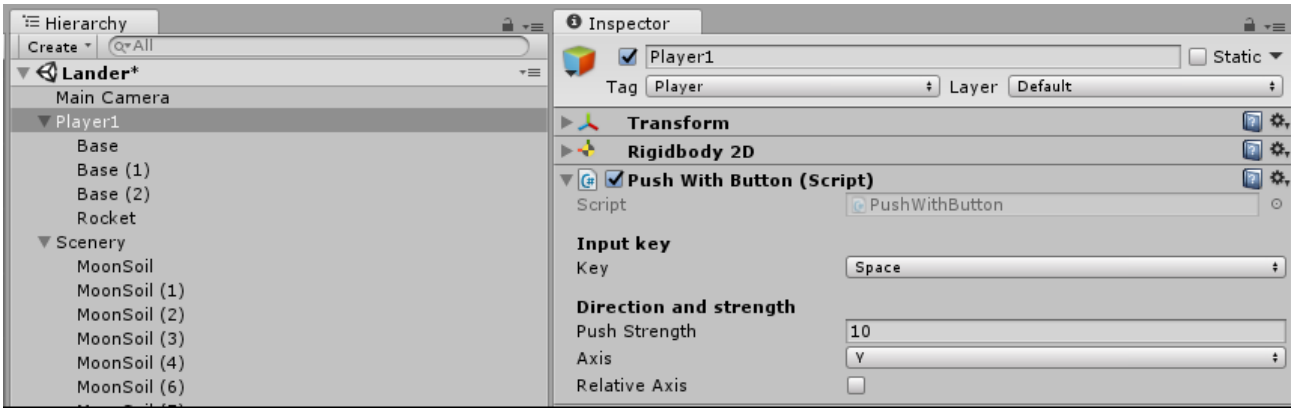




8. Run game. Lander should fall onto the starting platform.

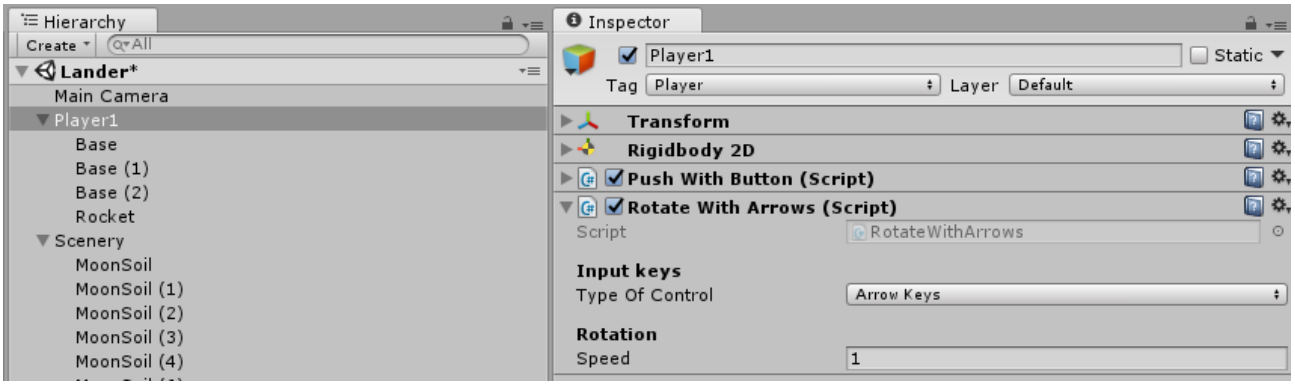
Unity Lander Lesson 2 - Setup Player Movements

1. Attach 'Push With Button' script to Player1



2. Change 'Push With Button' script default settings
    a. Change push strength
    b. Change Axis from Y to X
    c. Change Input key to another character.

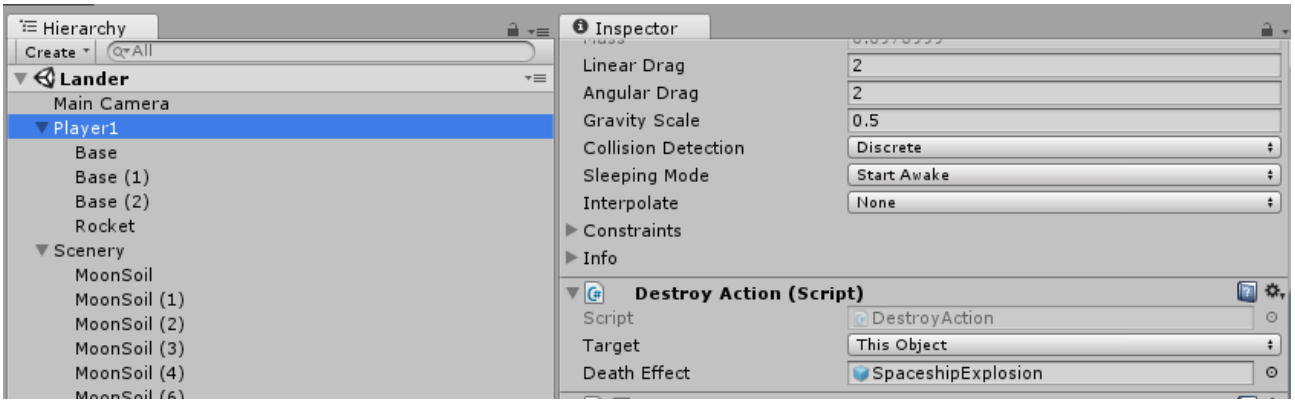
3. Attach 'Rotate With Arrows' script to Player1



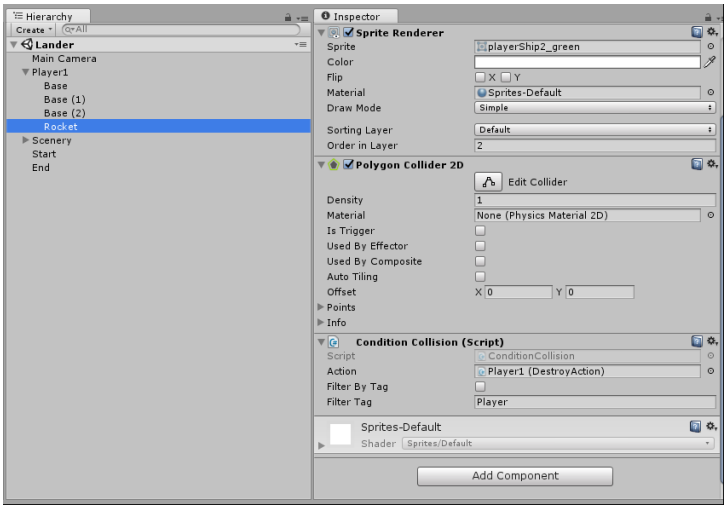
4. Change 'Rotate With Arrows' script Speed.
5. Open 'Push With Button' script in Visual Studio and change it. Run the game and see the changes.

Unity Lander Lesson 3 - Add Destroy Action Scripts

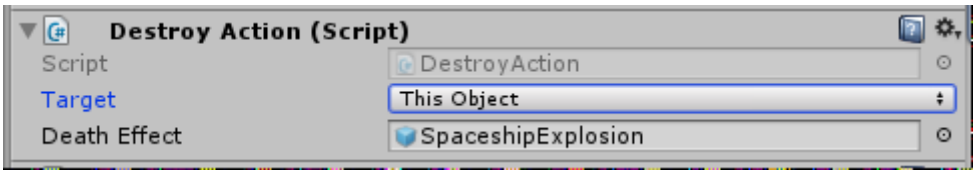
1. Attach Destroy Action Script to Player1



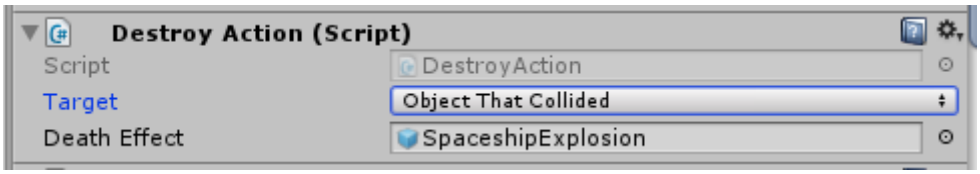
2. Attach Condition Collision script to Rocket



3. Run app with Target equal to This Object



4. Change Target equal to Object That Collided



Unity Roll-a-ball Game

These lessons are based on the Roll-a-ball tutorials provided by Unity. They are a simplified version of those lessons so students see a working game sooner. Deviate from the lessons as you see appropriate.

Roll-a-ball - https://unity3d.com/learn/tutorials/projects/roll-ball-tutorial

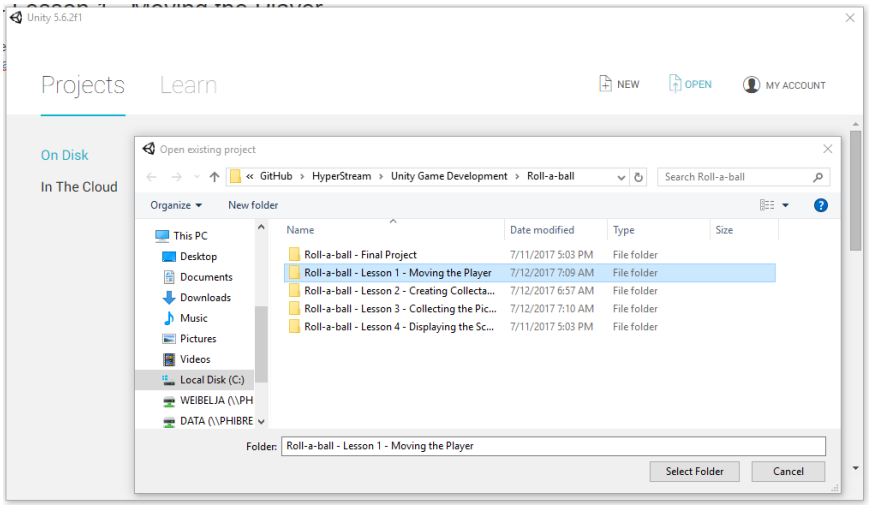
Source Code - https://github.com/jasonweibel/HyperStream-Unity

- Unity Roll-a-ball Game - Lesson 1 - Moving the Player
- Unity Roll-a-ball Game - Lesson 2 - Creating Collectable Objects
- Unity Roll-a-ball Game - Lesson 3 - Collecting the Pick Up Objects
- Unity Roll-a-ball Game - Lesson 4 - Displaying the Score and Text

Unity Roll-a-ball Game - Lesson 1 - Moving the Player

Lesson

1. Open 'Roll-a-ball - Lesson 1 - Moving the Player' project in Unity.



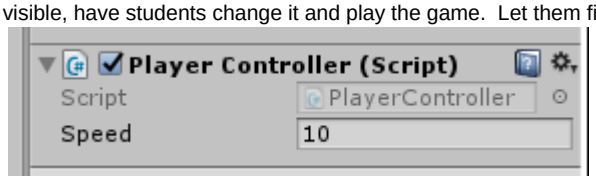
2. Open PlayerController Script
    a. Uncomment speed variable

```
0 references | 0 changes | 0 authors, 0 changes
public class PlayerController : MonoBehaviour {

    // Create public variables for player speed, and for the Text UI game objects
    //public float speed;
    //public Text countText;
    //public Text winText;
```

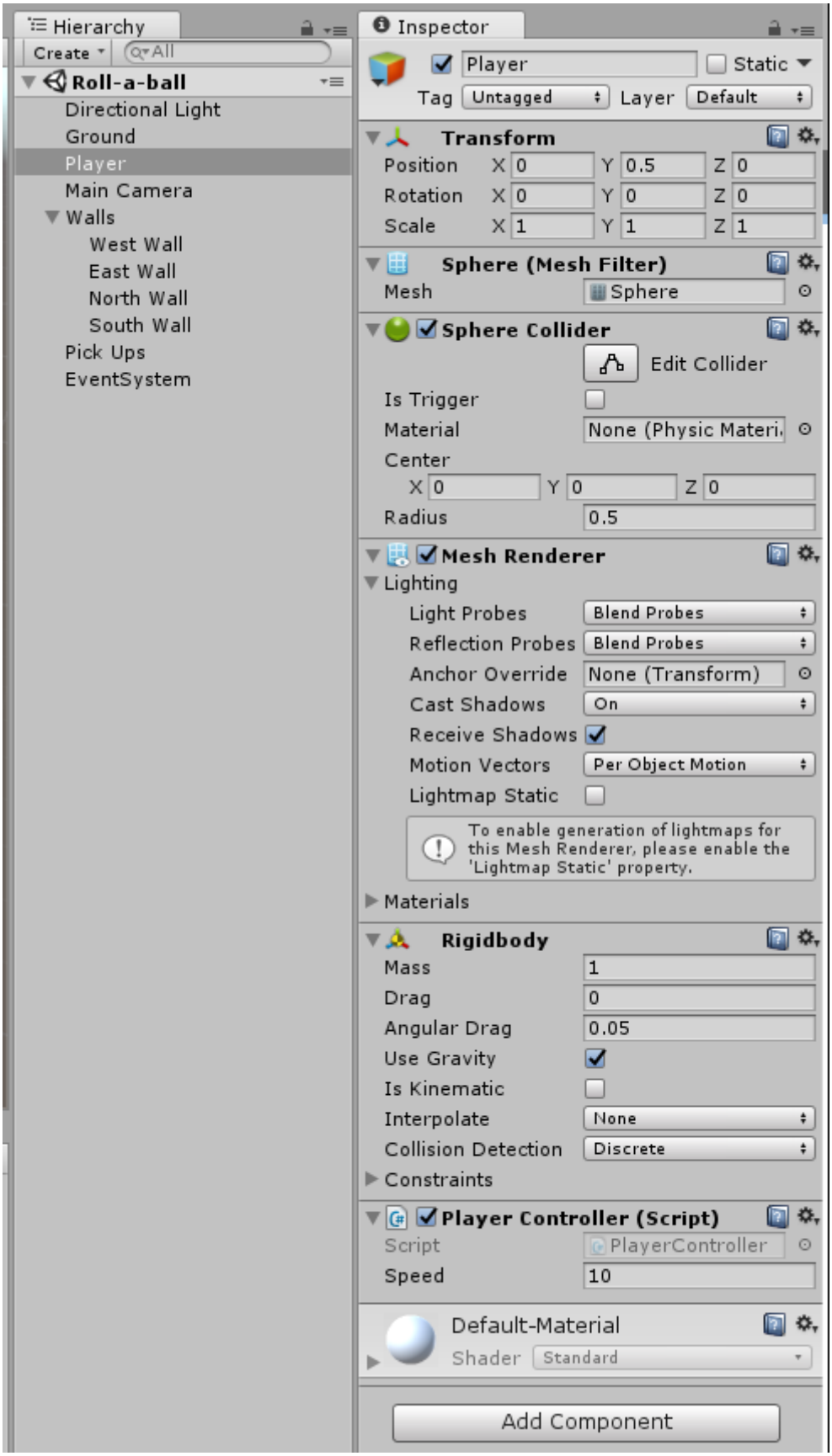
    b. Uncomment Fixed Update method

```
//// Each physics step..
//void FixedUpdate ()
//{
//  // Set some local float variables equal to the value of our Horizontal and Vertical Inputs
//  float moveHorizontal = Input.GetAxis ("Horizontal");
//  float moveVertical = Input.GetAxis ("Vertical");

//  // Create a Vector3 variable, and assign X and Z to feature our horizontal and vertical float variables above
//  Vector3 movement = new Vector3 (moveHorizontal, 0.0f, moveVertical);

//  // Add a physical force to our Player rigidbody using our 'movement' Vector3 above,
//  // multiplying it by 'speed' - our public player speed that appears in the inspector
//  rb.AddForce (movement * speed);
//}
```

3. Return to Unity, select the player object and find the Player Controller script in the Inspector Window. Speed parameter should be visible, have students change it and play the game. Let them find a speed they like.



4. Finished Player Inspector Window



5. Run Game

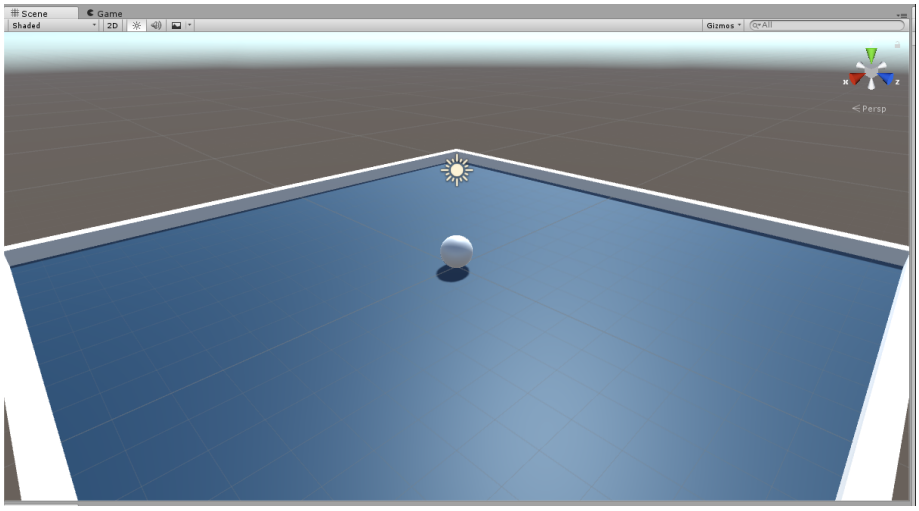
Unity Roll-a-ball Game - Lesson 2 - Creating Collectable Objects

Reference Tutorials

The Scene View - https://unity3d.com/learn/tutorials/topics/interface-essentials/scene-view?playlist=17090

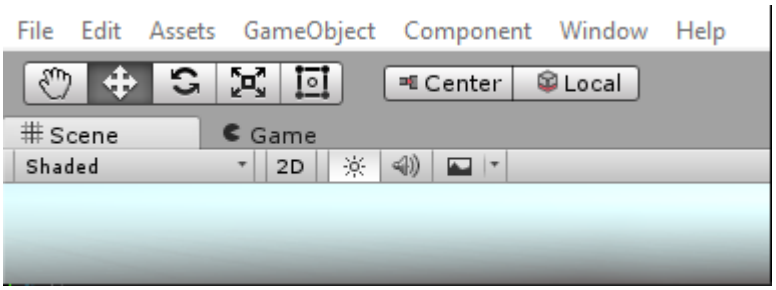
Prefabs - Concept & Usage - https://unity3d.com/learn/tutorials/topics/interface-essentials/prefabs-concept-usage?playlist=17090

Starting Scene

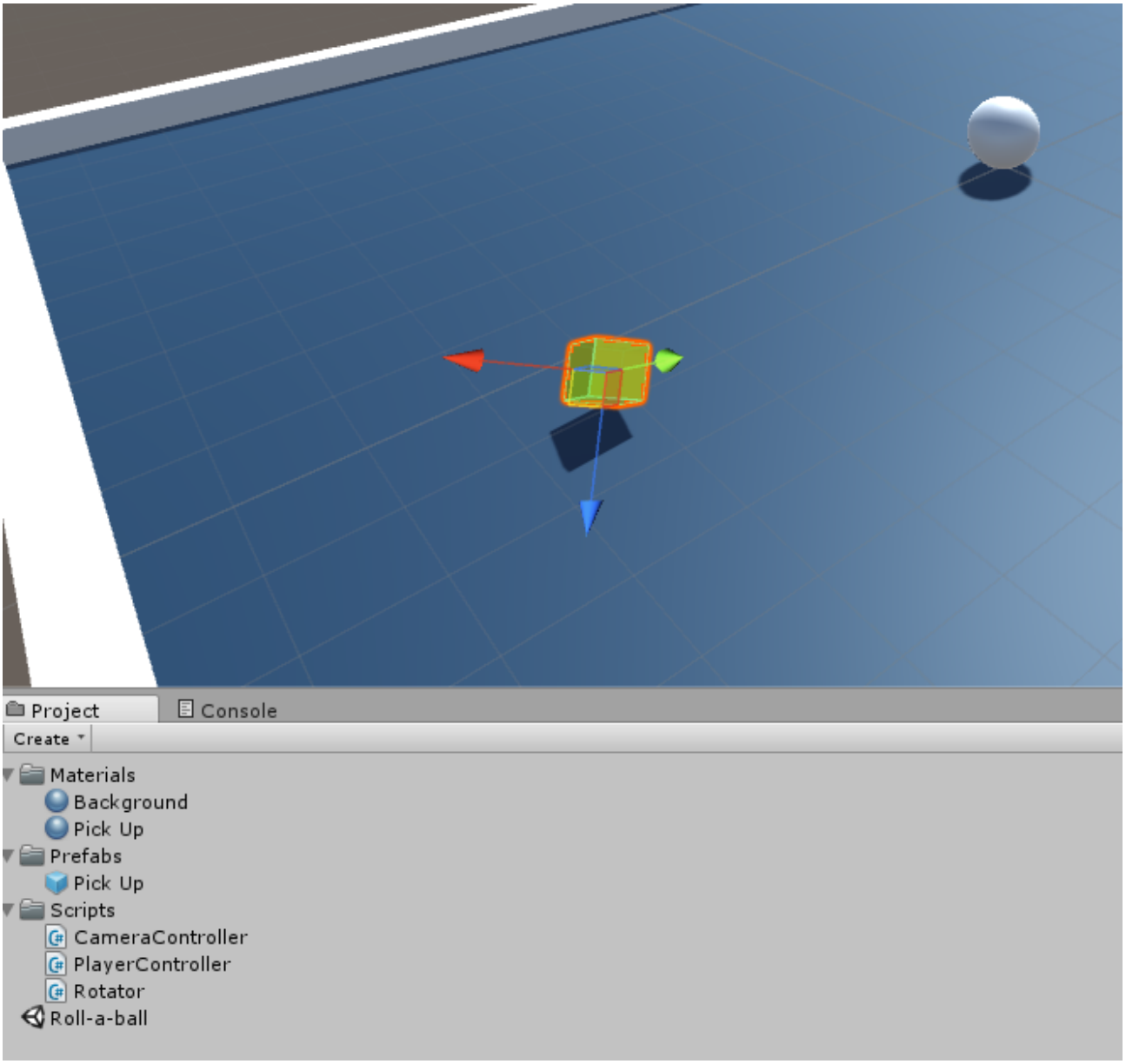


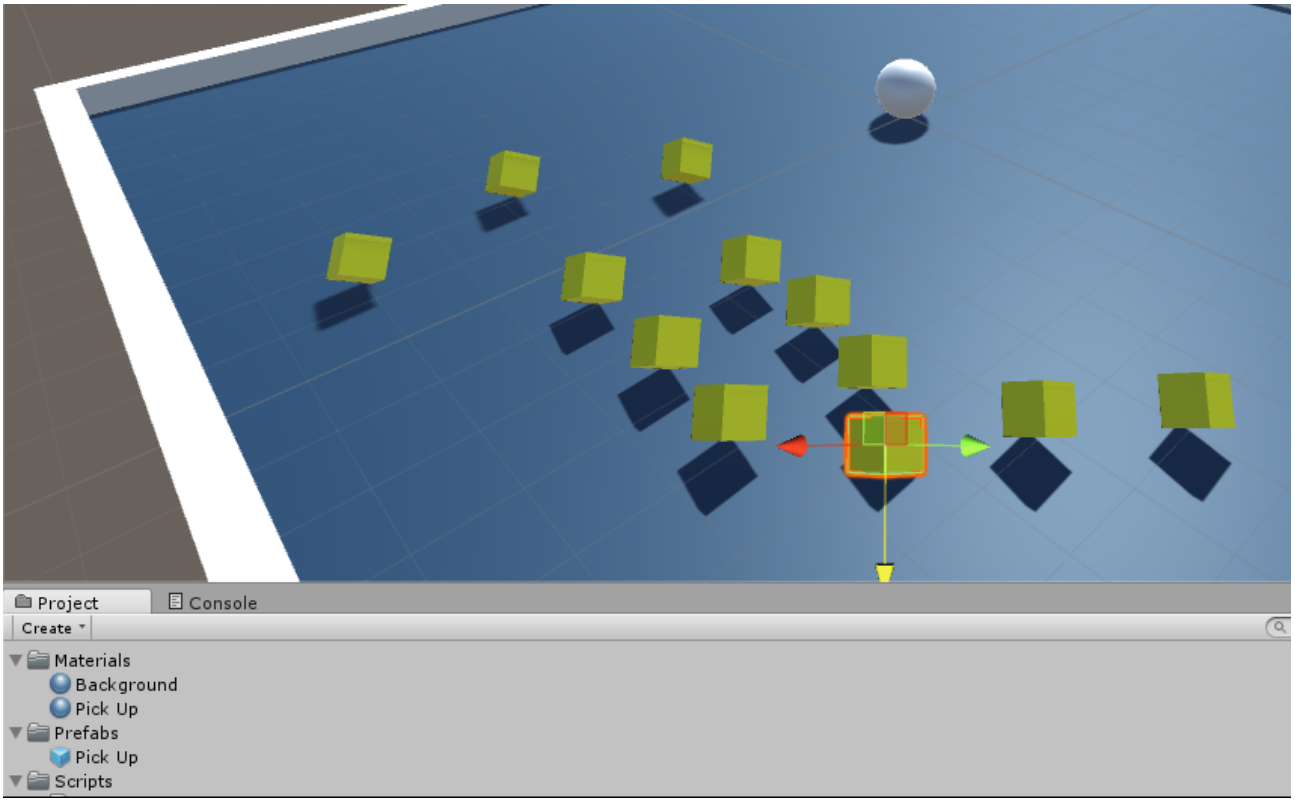
Add 12 Pick Up Prefab objects to the Scene

1. Select Scene window

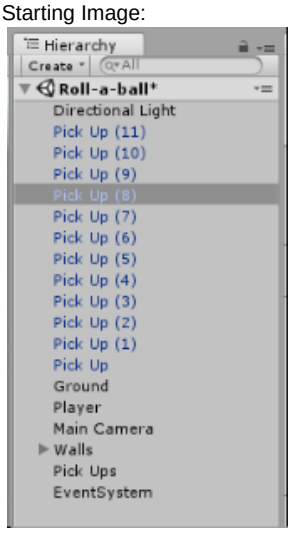


2. Drag and Drop 12 Pick Up Prefab objects onto the scene

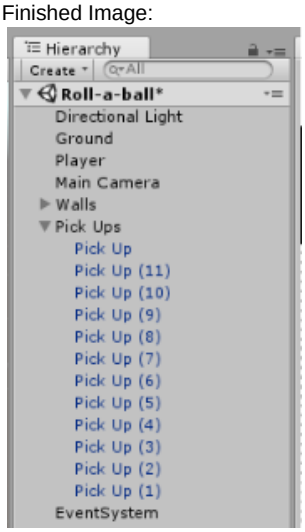

Project
Console
Create
Materials
Background
Pick Up
Prefabs
Pick Up
Scripts
CameraController
PlayerController
Rotator
Roll-a-ball



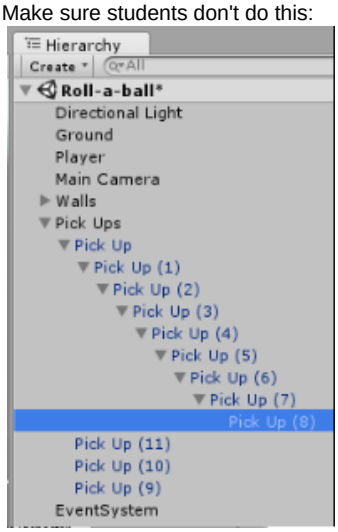
3. Reorganize Hierarchy Window so all Pick Ups are under the empty Pick Ups object.
   Starting Image:



   Finished Image:

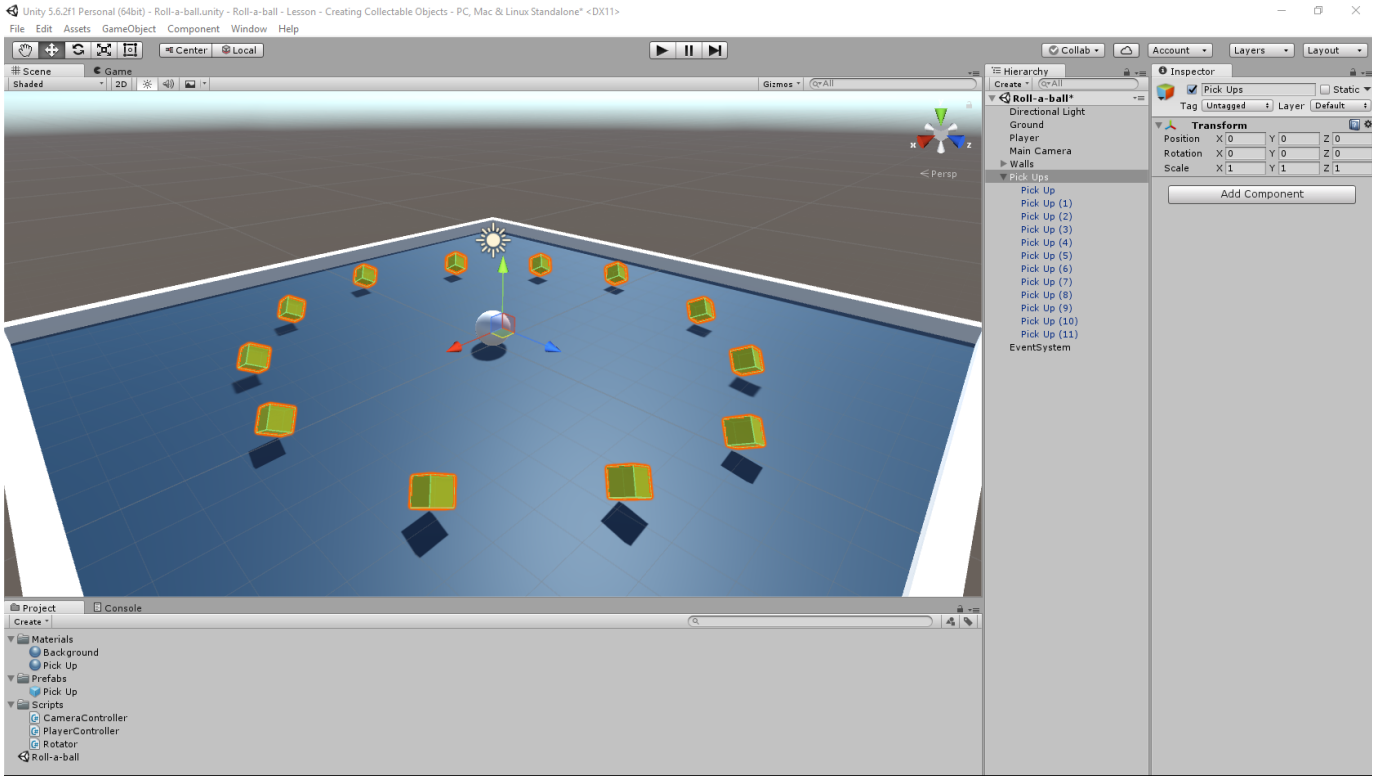


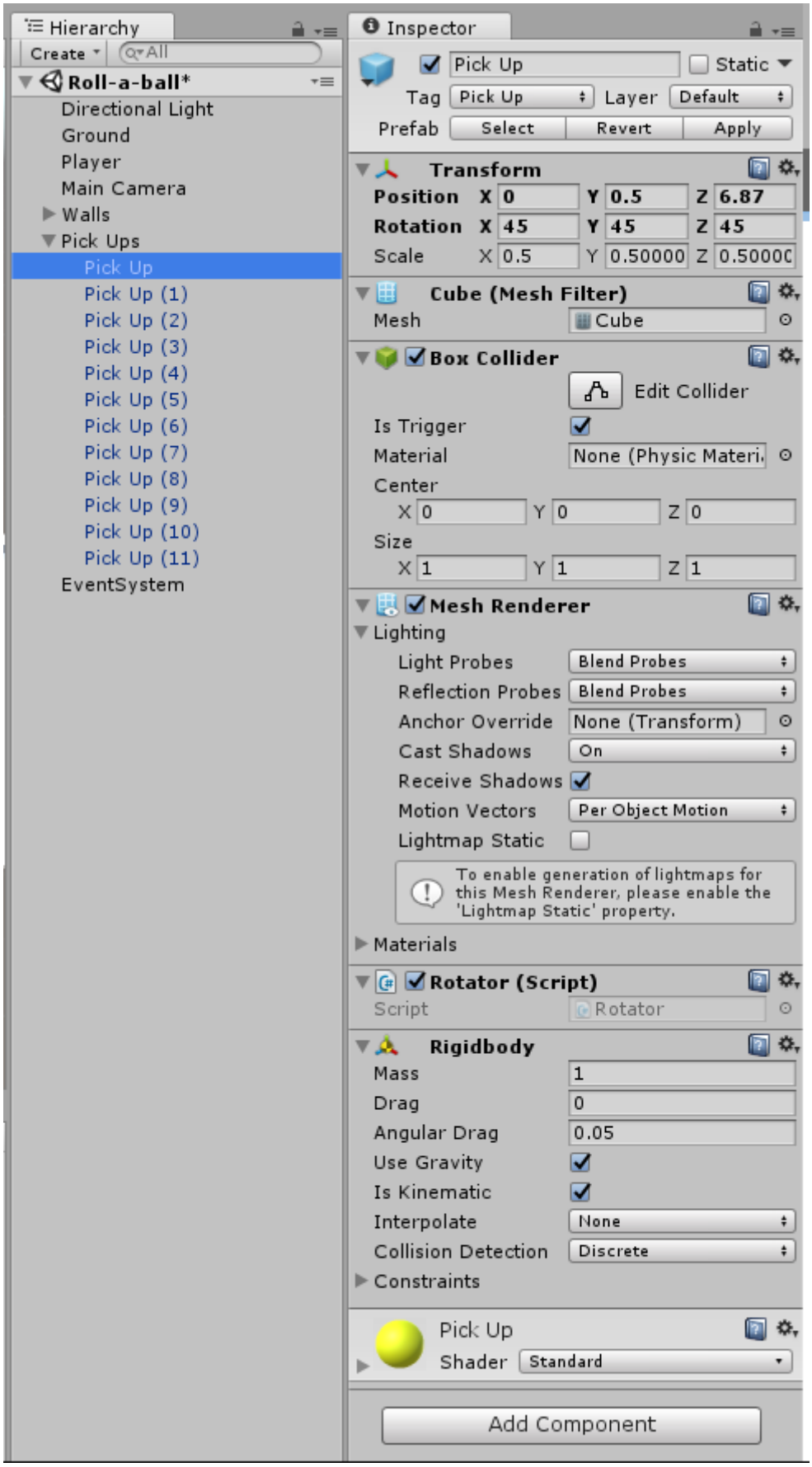
Make sure students don't do this:



4. Move Pickups around the scene. If students are having problems doing this with the mouse have them type the coordinates into the Transform settings of each Pick Up.
   Important - Make sure the Y Position is always set to 0.5.

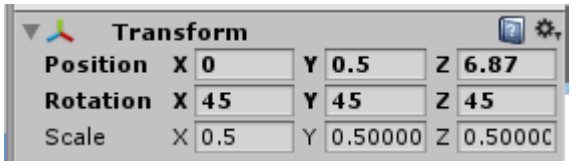
Finished Scene

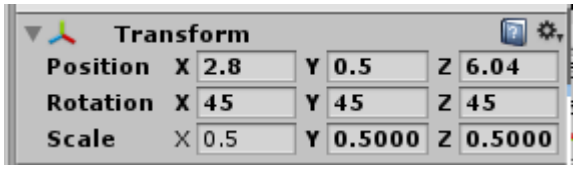


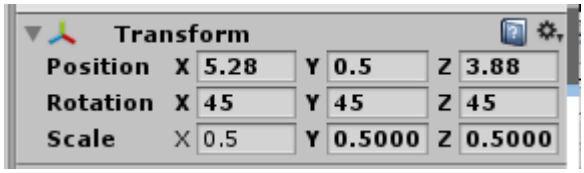


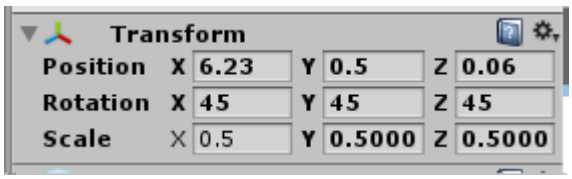
Transform setting for each pick up - for reference only, encourage the students to place the pick ups where ever they want.

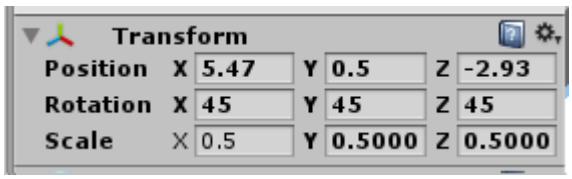
Important - Make sure the Y Position is always set to 0.5.

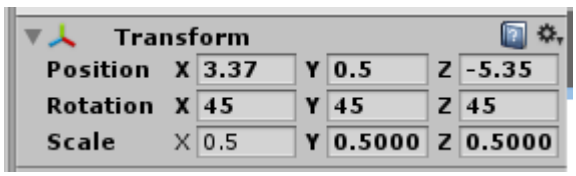

Transform
Position X 0 Y 0.5 Z 6.87
Rotation X 45 Y 45 Z 45
Scale X 0.5 Y 0.50000 Z 0.50000

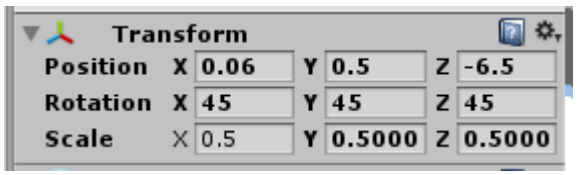

Transform
Position X 2.8 Y 0.5 Z 6.04
Rotation X 45 Y 45 Z 45
Scale X 0.5 Y 0.5000 Z 0.5000

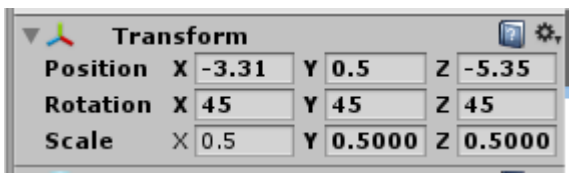

Transform
Position X 5.28 Y 0.5 Z 3.88
Rotation X 45 Y 45 Z 45
Scale X 0.5 Y 0.5000 Z 0.5000

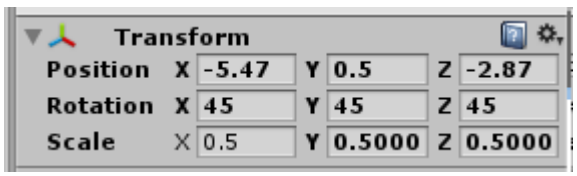

Transform
Position X 6.23 Y 0.5 Z 0.06
Rotation X 45 Y 45 Z 45
Scale X 0.5 Y 0.5000 Z 0.5000

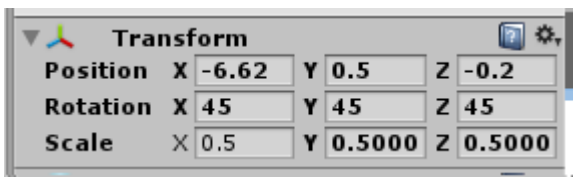

Transform
Position X 5.47 Y 0.5 Z -2.93
Rotation X 45 Y 45 Z 45
Scale X 0.5 Y 0.5000 Z 0.5000

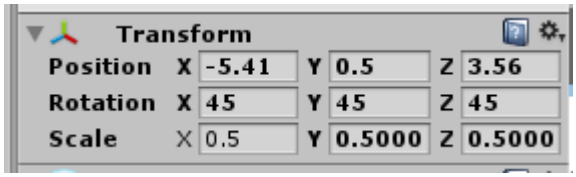

Transform
Position X 3.37 Y 0.5 Z -5.35
Rotation X 45 Y 45 Z 45
Scale X 0.5 Y 0.5000 Z 0.5000

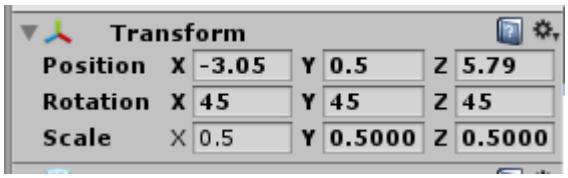

Transform
Position X 0.06 Y 0.5 Z -6.5
Rotation X 45 Y 45 Z 45
Scale X 0.5 Y 0.5000 Z 0.5000


Transform
Position X -3.31 Y 0.5 Z -5.35
Rotation X 45 Y 45 Z 45
Scale X 0.5 Y 0.5000 Z 0.5000


Transform
Position X -5.47 Y 0.5 Z -2.87
Rotation X 45 Y 45 Z 45
Scale X 0.5 Y 0.5000 Z 0.5000


Transform
Position X -6.62 Y 0.5 Z -0.2
Rotation X 45 Y 45 Z 45
Scale X 0.5 Y 0.5000 Z 0.5000





Unity Roll-a-ball Game - Lesson 3 - Collecting the Pick Up Objects

Reference

Unity Tutorial - https://unity3d.com/learn/tutorials/projects/roll-ball-tutorial/collecting-pick-objects?playlist=17141

Debugging Unity games in Visual Studio - https://unity3d.com/learn/tutorials/topics/scripting/debugging-unity-games-visual-studio

Lesson

1. Open PlayerController.cs script in Visual Studio
2. Uncomment the OnTriggerEntry method

```
50      //// When this game object intersects a collider with 'is trigger' checked,
51      //// store a reference to that collider in a variable named 'other'..
52      //void OnTriggerEnter(Collider other)
53      //{
54      //  // ..and if the game object we intersect has the tag 'Pick Up' assigned to it..
55      //  if (other.gameObject.CompareTag ("Pick Up"))
56      //  {
57      //      // Make the other game object (the pick up) inactive, to make it disappear
58      //      other.gameObject.SetActive (false);
59
60      //      // Add one to the score variable 'count'
61      //      count = count + 1;
62
63      //      //// Run the 'SetCountText()' function (see below)
64      //      //SetCountText ();
65      //  }
66      //}
```

3. Play Game!
4. Edit the compare tag parameter on line 55. What happens?
5. On line 58 change the SetActive parameter to true? What happens?
6. Add a break point to the script and debug the project.

Unity Roll-a-ball Game - Lesson 4 - Displaying the Score and Text

Reference

Unity Tutorial - https://unity3d.com/learn/tutorials/projects/roll-ball-tutorial/displaying-score-and-text?playlist=17141

Update PlayerController.cs script

1. Uncomment countText and winText public variable

```
11      public float speed;
12      //public Text countText;
13      //public Text winText;
```

2. Uncomment SetCountText method

```
68  //// Create a standalone function that can update the 'countText' UI and check if the required amount to win has been achieved
69  //void SetCountText()
70  //{
71  //  // Update the text field of our 'countText' variable
72  //  countText.text = "Count: " + count.ToString ();
73
74  //  // Check if our 'count' is equal to or exceeded 12
75  //  if (count >= 12)
76  //  {
77  //      // Set the text value of our 'winText'
78  //      winText.text = "You Win!";
79  //  }
80  //}
81  }
```

3. Uncomment the use of SetCountText in OnTriggerEntry method

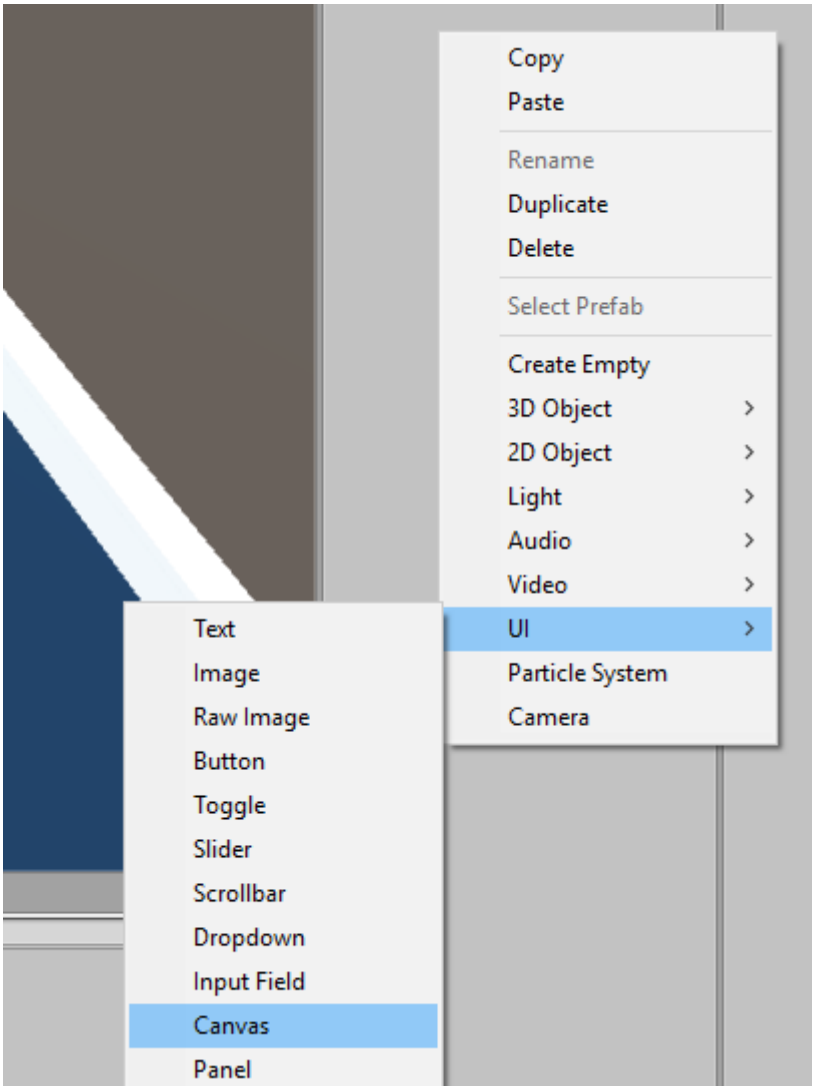
```
62
63              //// Run the 'SetCountText()' function (see below)
64              //SetCountText ();
65          }
```

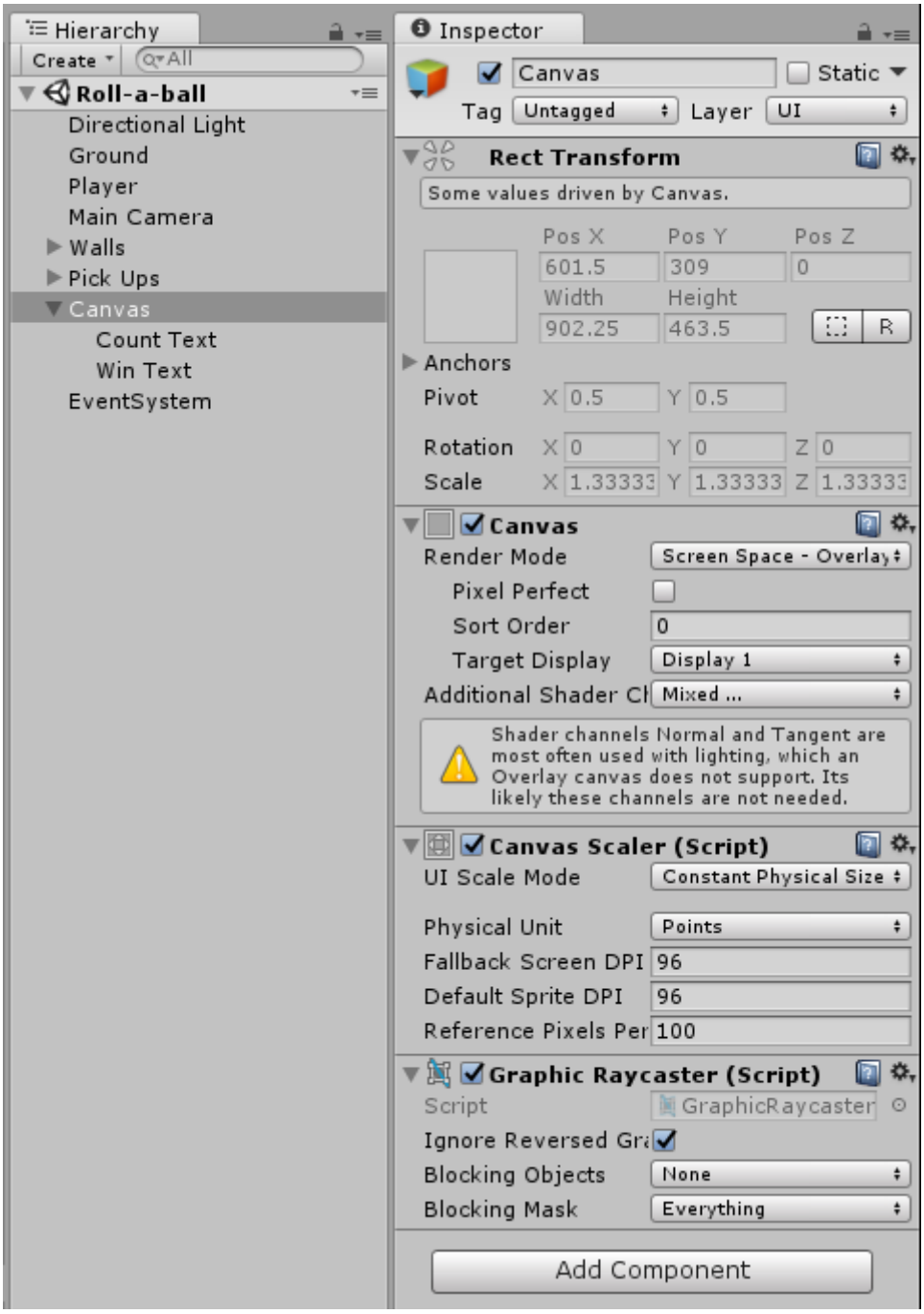
4. In Start method uncomment the use of SetCountText and setting of winText.text

```
28      //// Run the SetCountText function to update the UI (see below)
29      //SetCountText ();
30
31      // Set the text property of our Win Text UI to an empty string, making the 'You Win' (game over message) blank
32      //winText.text = "";
33  }
```

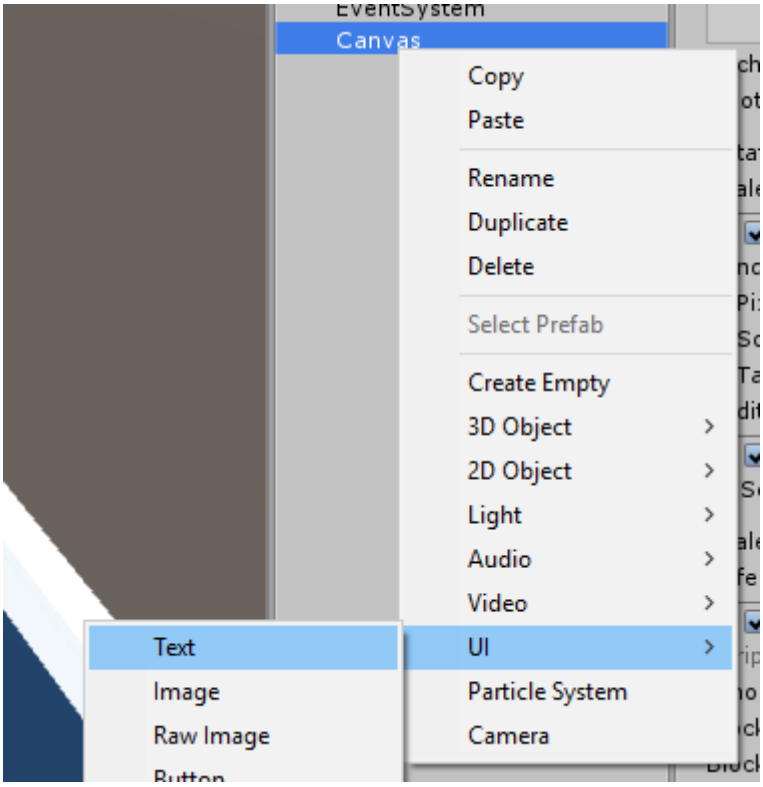
Add Canvas, Count Text, and Win Text

1. Create Canvas

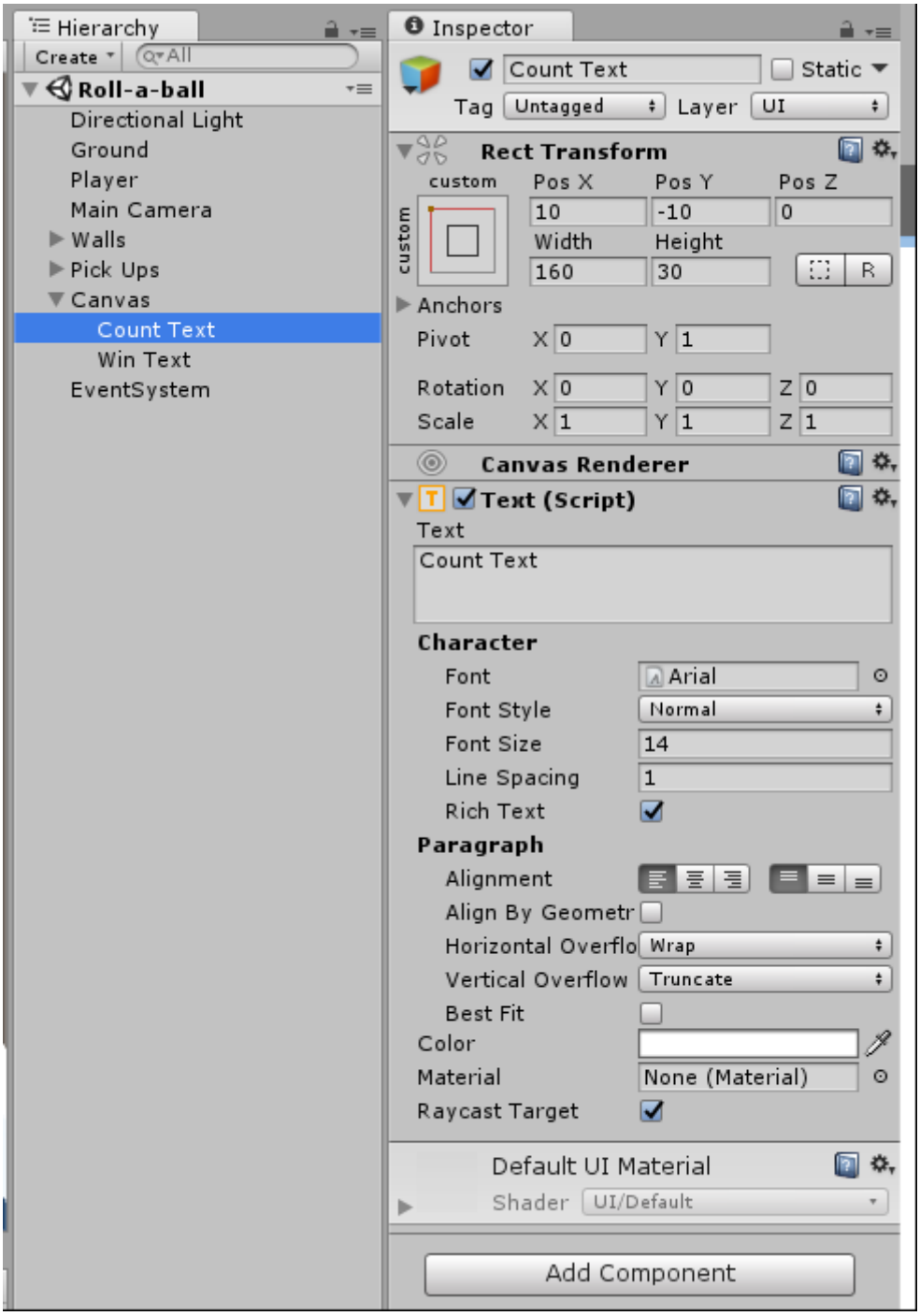

Copy
Paste
Rename
Duplicate
Delete
Select Prefab
Create Empty
3D Object
2D Object
Light
Audio
Video
UI
Particle System
Camera
Text
Image
Raw Image
Button
Toggle
Slider
Scrollbar
Dropdown
Input Field
Canvas
Panel



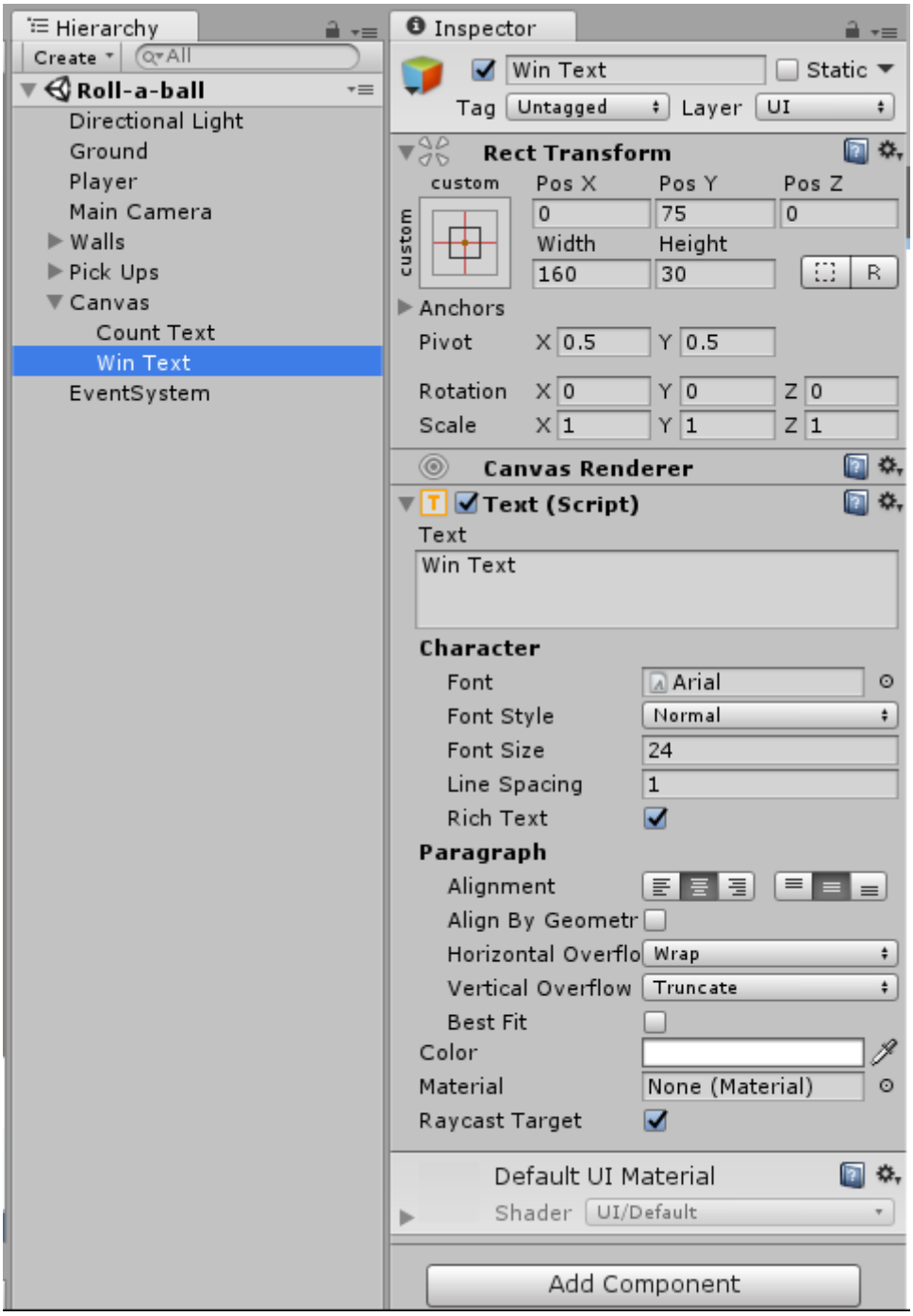
2. Add Text to Canvas



3. Add Count Text



4. Add Win Text



Play Final Game!

Unity Simple Ball Rolling Game

These lessons are designed to help mentors present a 'Show it. Try it.' curriculum. Each step in this lesson can quickly be presented then should be given the opportunity to do what they learned.

These lessons are a simplified version of the Roll-a-ball tutorial on the unity3d.com site.

Lessons

- Unity Simple Ball Rolling Game: Lesson 1 - Setup Sphere and Terrain
- Unity Simple Ball Rolling Game: Lesson 2 - Setup Camera
- Unity Simple Ball Rolling Game: Lesson 3 - Add Detail to Terrain
- Unity Simple Ball Rolling Game: Lesson 3 - Adding Terrain Textures

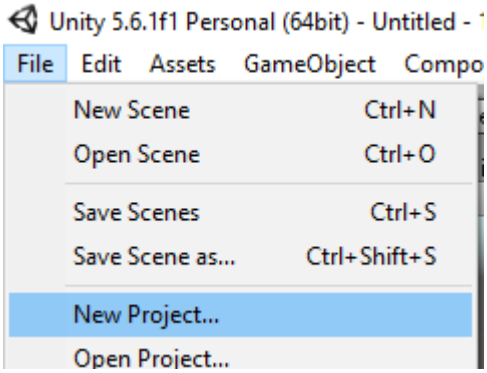
Unity Simple Ball Rolling Game: Lesson 1 - Setup Sphere and Terrain

Notes

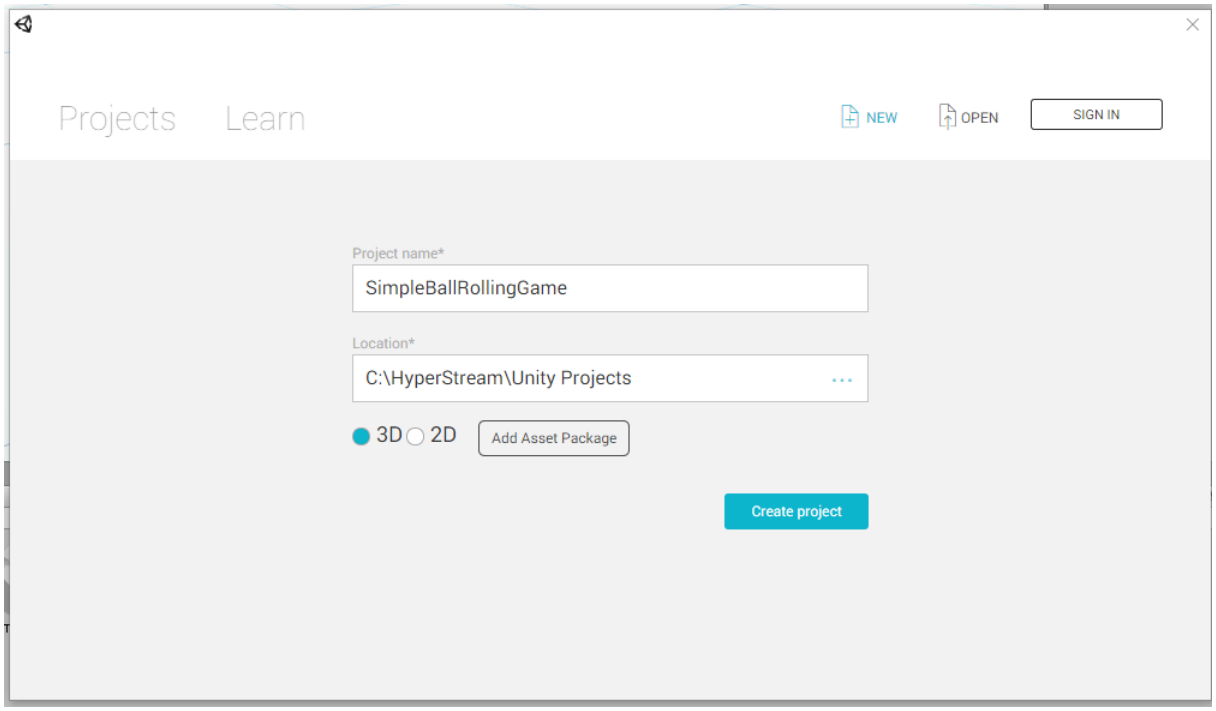
These lessons are intended to help mentors quickly get up to speed with Unity. The hope is the lessons can be used in a 'show then do' format giving students flexibility to change the game as they prefer.

Lesson 1 - Setup Sphere and Terrain

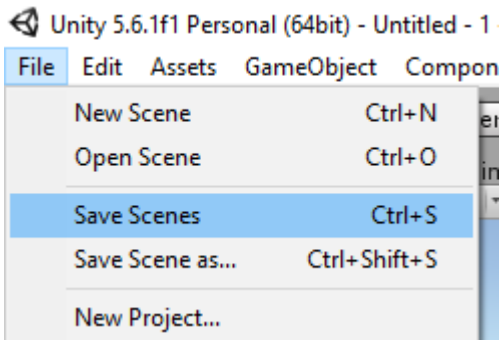
1. Open Unity
2. Select File -> New Project



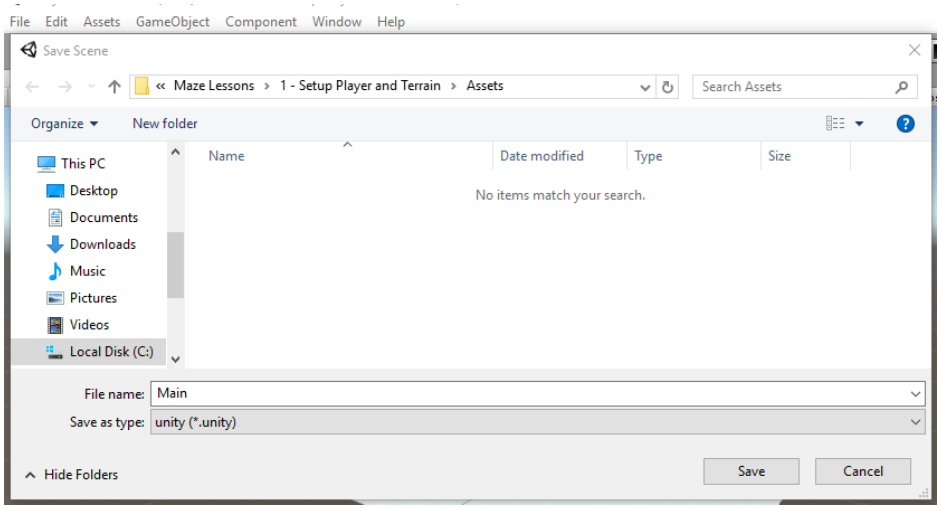
3. Complete new project form:
    a. Name project SimpleBallRollingGame
    b. Set project location
    c. Check 3D



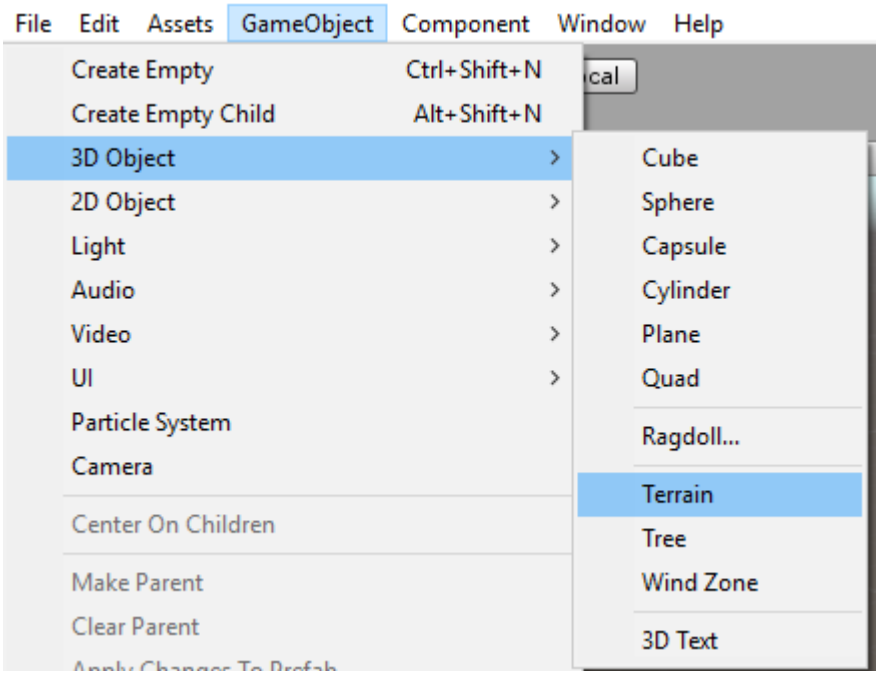
4. Save Scene
    a. Select Save Scenes from File menu.



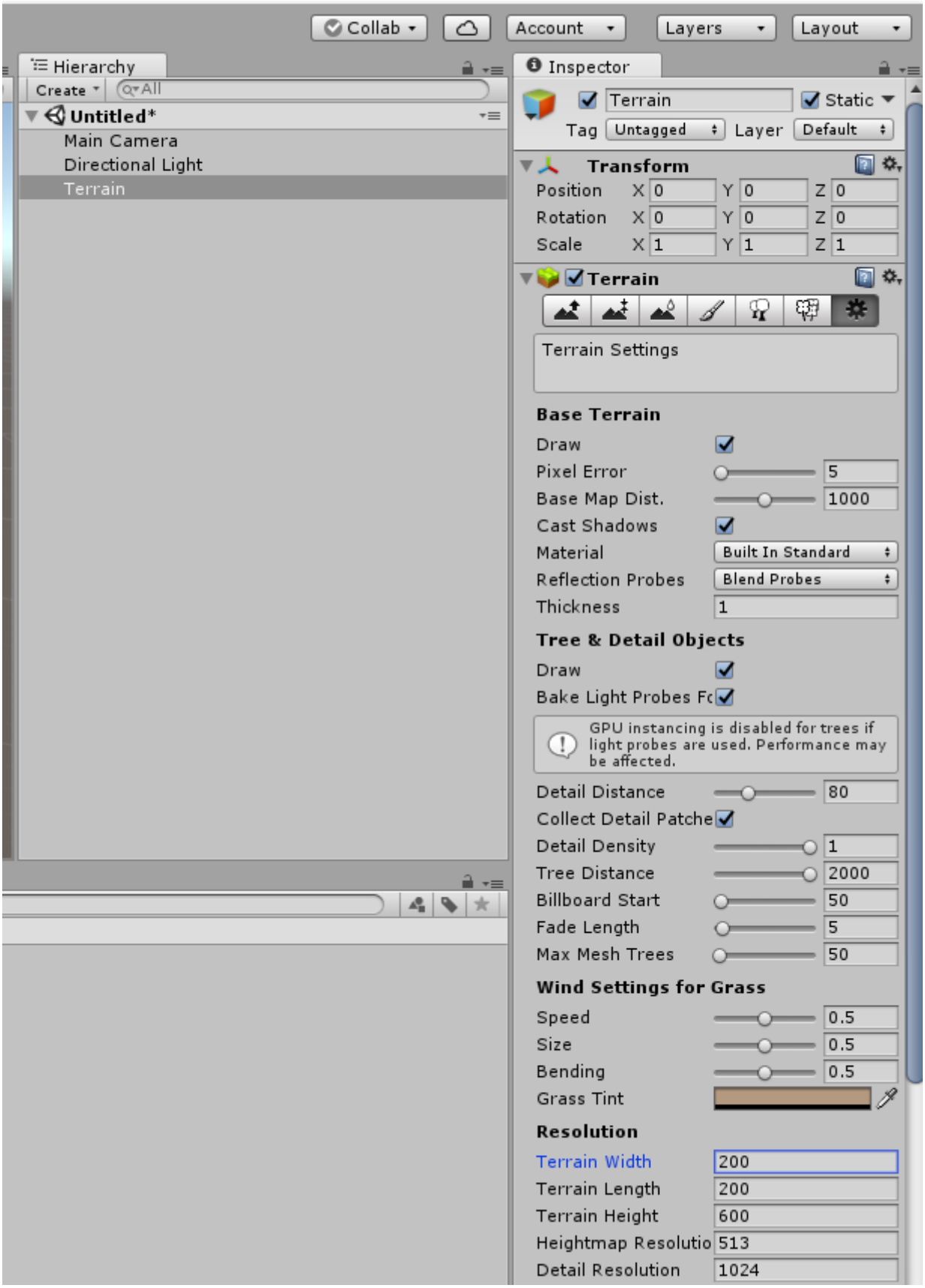
    b. Name scene Main



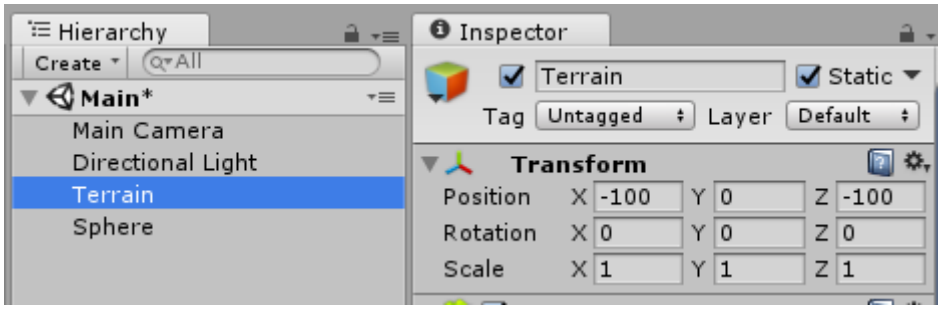
5. Create a Terrain



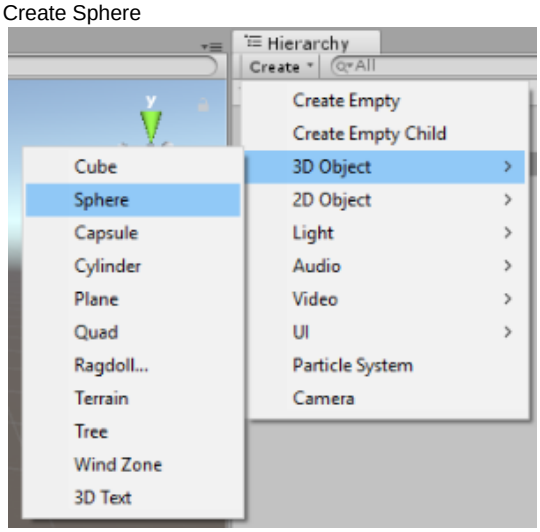
6. Resize Terrain



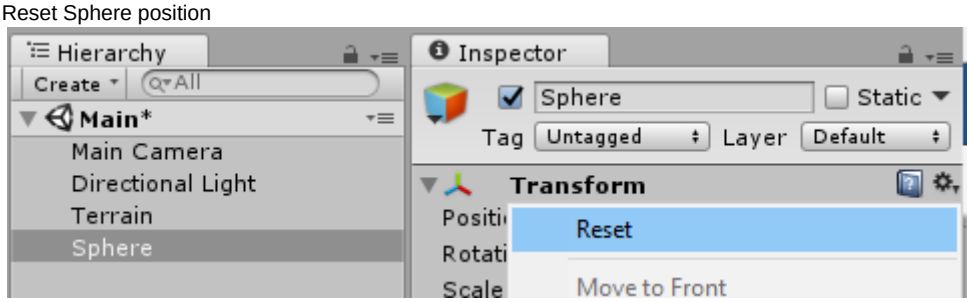
7. Set Terrain position
    a.



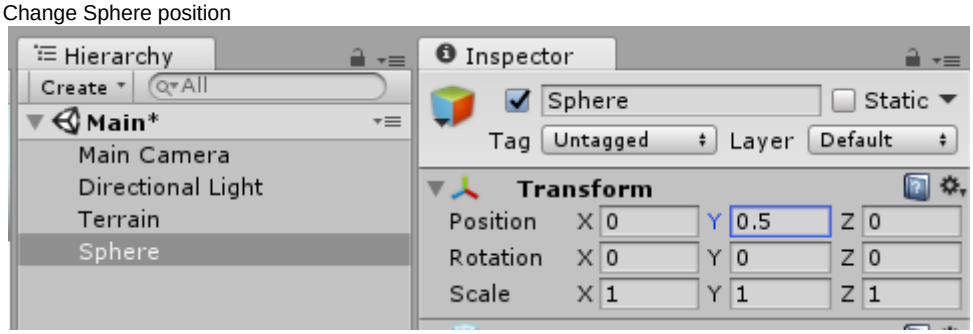
8. Create Sphere



9. Reset Sphere position

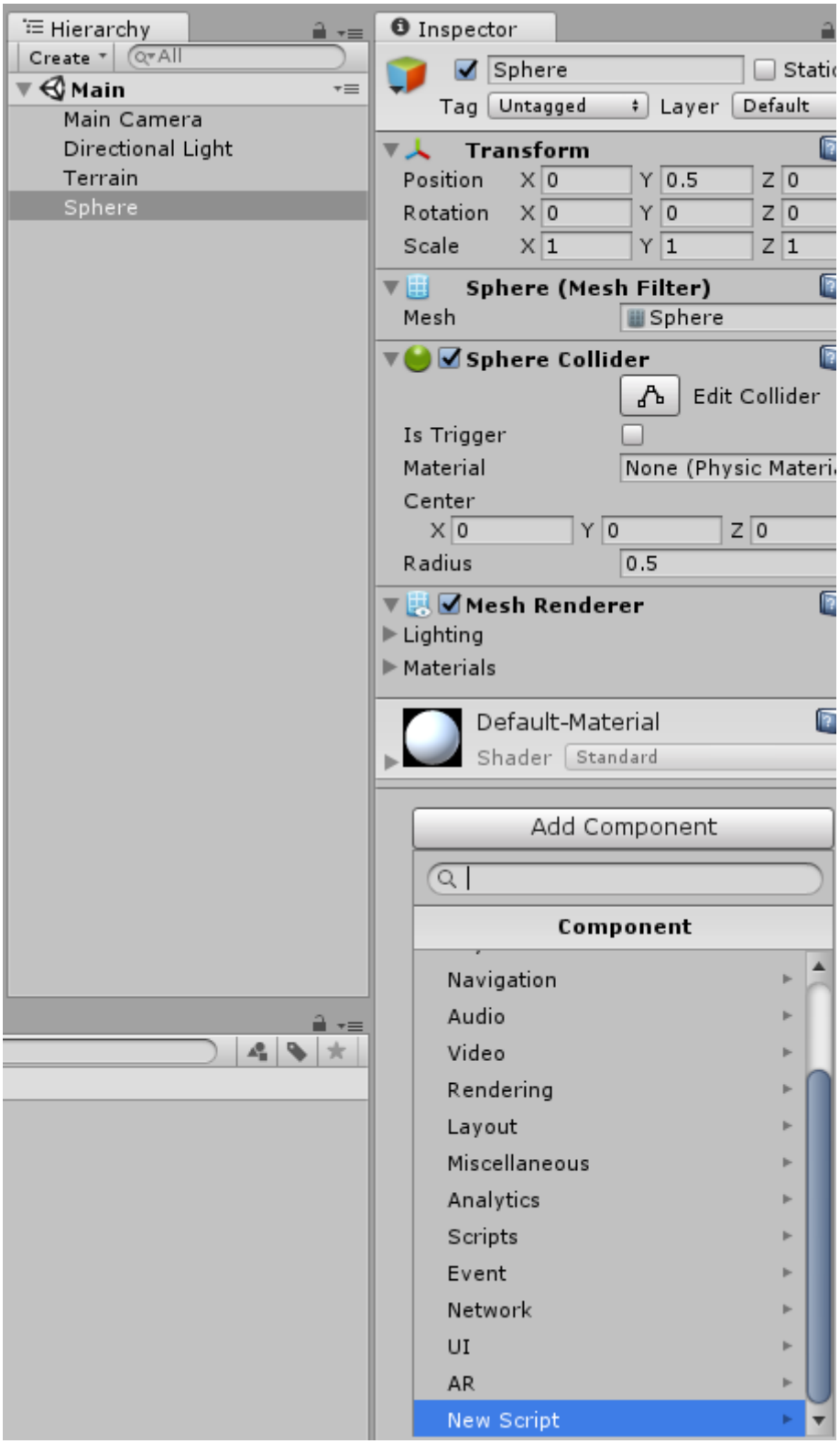


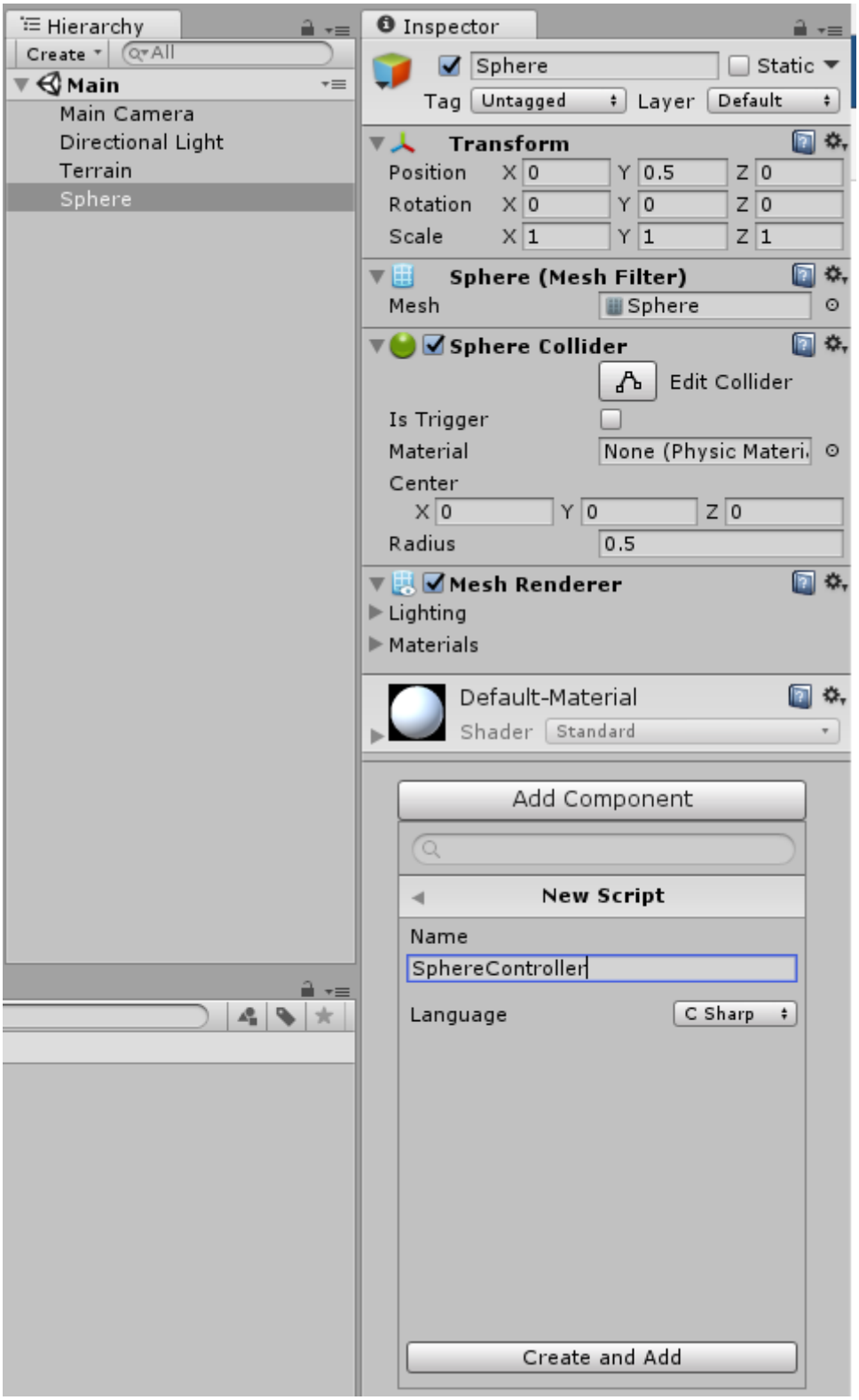
10. Change Sphere position



11. Create SphereController script

12.


Hierarchy
Create
All
Main
Main Camera
Directional Light
Terrain
Sphere
Inspector
Sphere
Tag
Untagged
Layer
Default
Transform
Position
Rotation
Scale
Sphere (Mesh Filter)
Mesh
Sphere
Sphere Collider
Edit Collider
Is Trigger
Material
None (Physic Materi
Center
Radius
0.5
Mesh Renderer
Lighting
Materials
Default-Material
Shader
Standard
Add Component
Component
Navigation
Audio
Video
Rendering
Layout
Miscellaneous
Analytics
Scripts
Event
Network
UI
AR
New Script

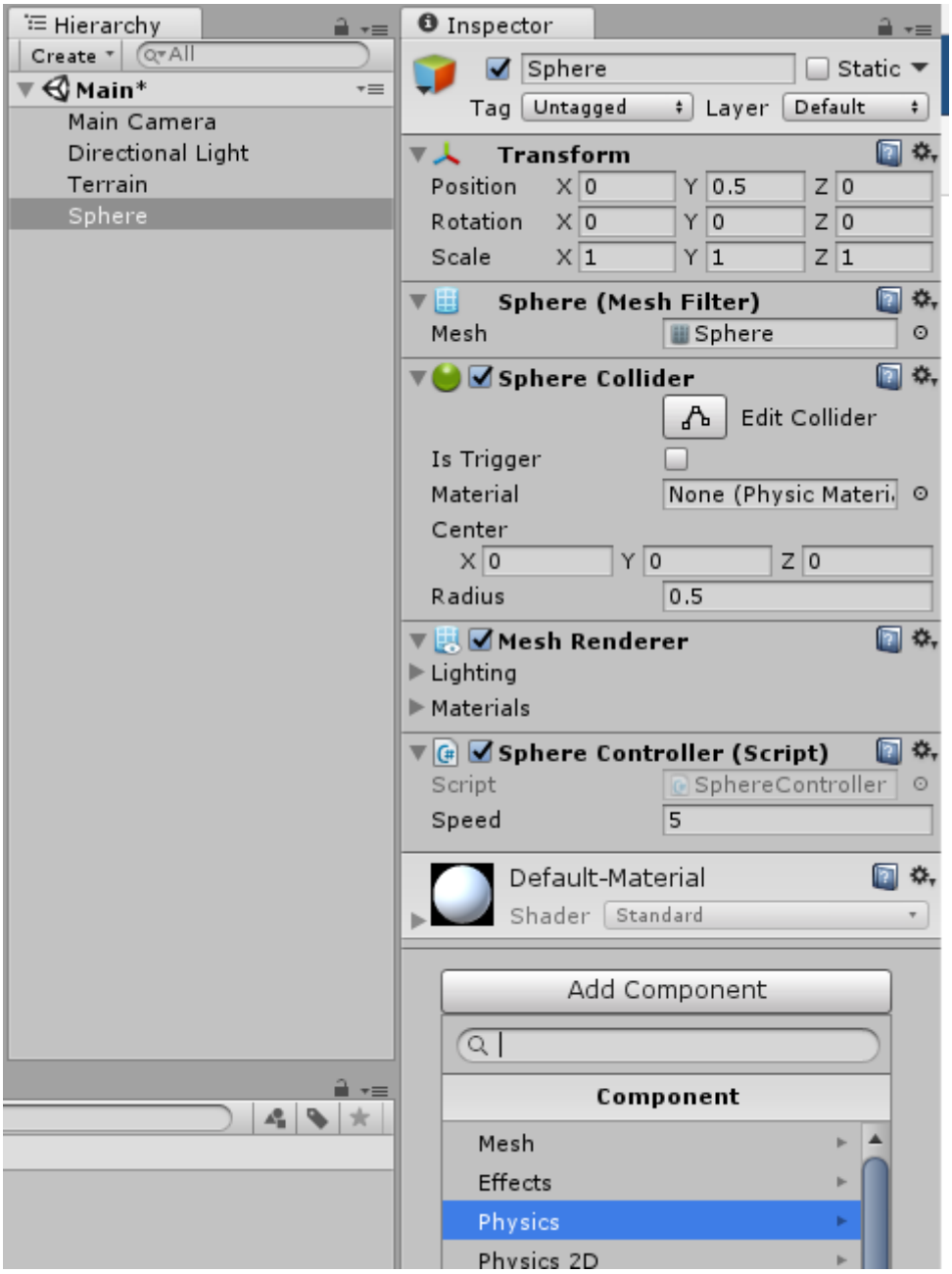


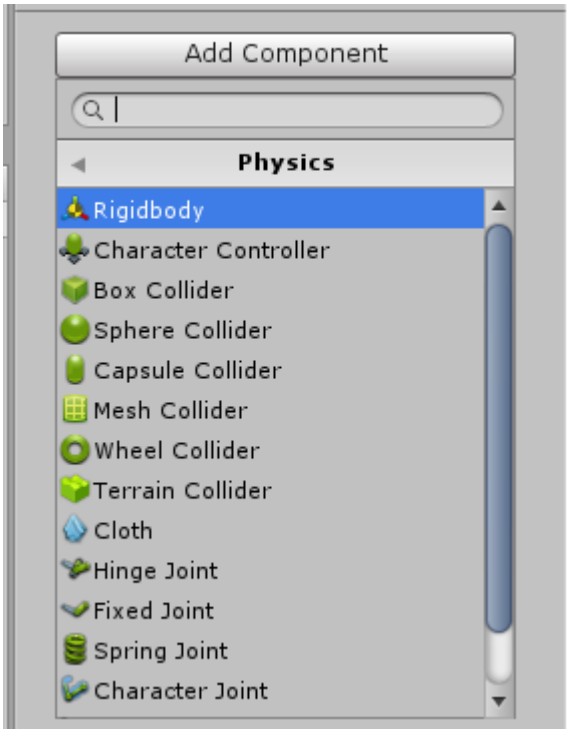
13. Edit SphereController script

SphereController

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class SphereController : MonoBehaviour
{
    public float Speed;
    private Rigidbody _rb;

    void Start()
    {
        _rb = GetComponent<Rigidbody>();
    }

    void FixedUpdate()
    {
        var moveHorizontal = Input.GetAxis("Horizontal");
        var moveVertical = Input.GetAxis("Vertical");
        var movement = new Vector3(moveHorizontal, 0.0f,
moveVertical);

        _rb.AddForce(movement * Speed);
    }
}
```

14. Add Rigidbody component to Sphere

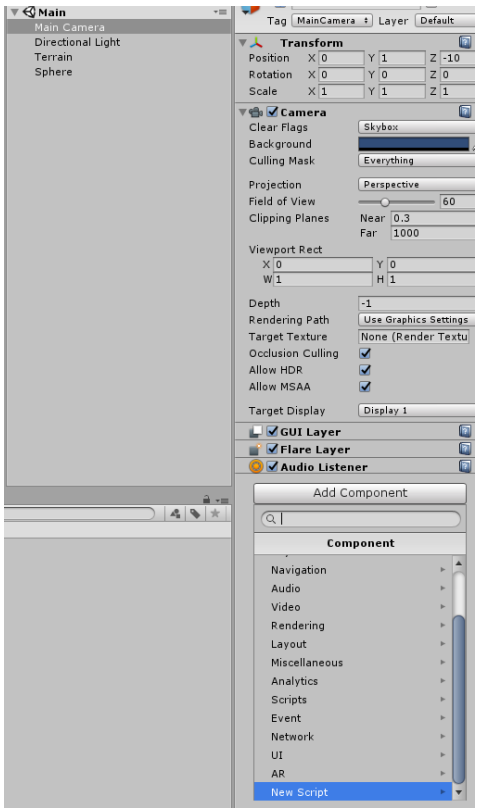

Hierarchy
Create
All
Main*
Main Camera
Directional Light
Terrain
Sphere
Inspector
Sphere
Static
Tag
Untagged
Layer
Default
Transform
Position
X 0
Y 0.5
Z 0
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1
Sphere (Mesh Filter)
Mesh
Sphere
Sphere Collider
Edit Collider
Is Trigger
Material
None (Physic Materi
Center
X 0
Y 0
Z 0
Radius
0.5
Mesh Renderer
Lighting
Materials
Sphere Controller (Script)
Script
SphereController
Speed
5
Default-Material
Shader
Standard
Add Component
Component
Mesh
Effects
Physics
Physics 2D



15. Play

Unity Simple Ball Rolling Game: Lesson 2 - Setup Camera

1. Open Unity project started with Unity Simple Ball Rolling Game: Lesson 1 - Setup Sphere and Terrain
2. Add a New Script Component to the Main Camera
    a. Select the Main Camera from the Hierarchy window.
    b. Click on Add Component button and select New Script.



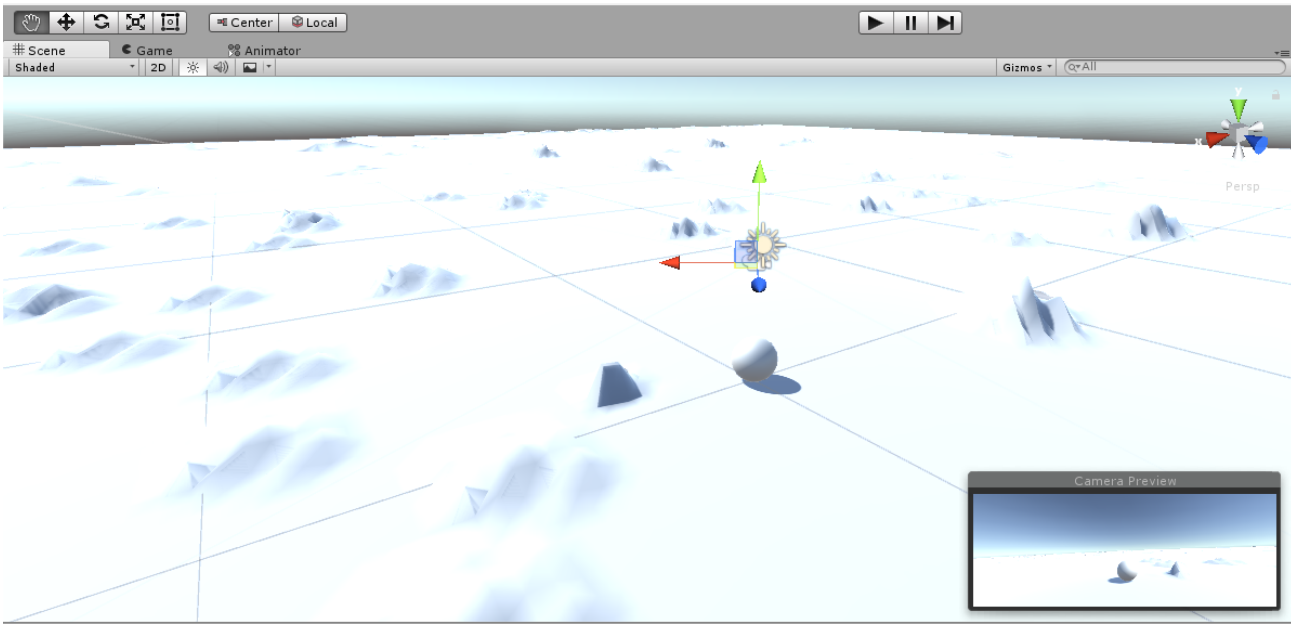
c. Open new CameraController script located in the Asset window with Visual Studio Express.
d. Edit CameraController script.

SphereController

```
using UnityEngine;
using System.Collections;

public class CameraController : MonoBehaviour
{
    public GameObject Player;
    private Vector3 _offset;

    void Start()
    {
        _offset = transform.position -
Player.transform.position;
    }

    void LateUpdate()
    {
        transform.position = Player.transform.position +
_offset;
    }
}
```

e. Associate Sphere to Player field in Camera Controller
From the Hierarchy window select the Sphere. Drag the selected Sphere into the Player field of the Camera Controller Script.

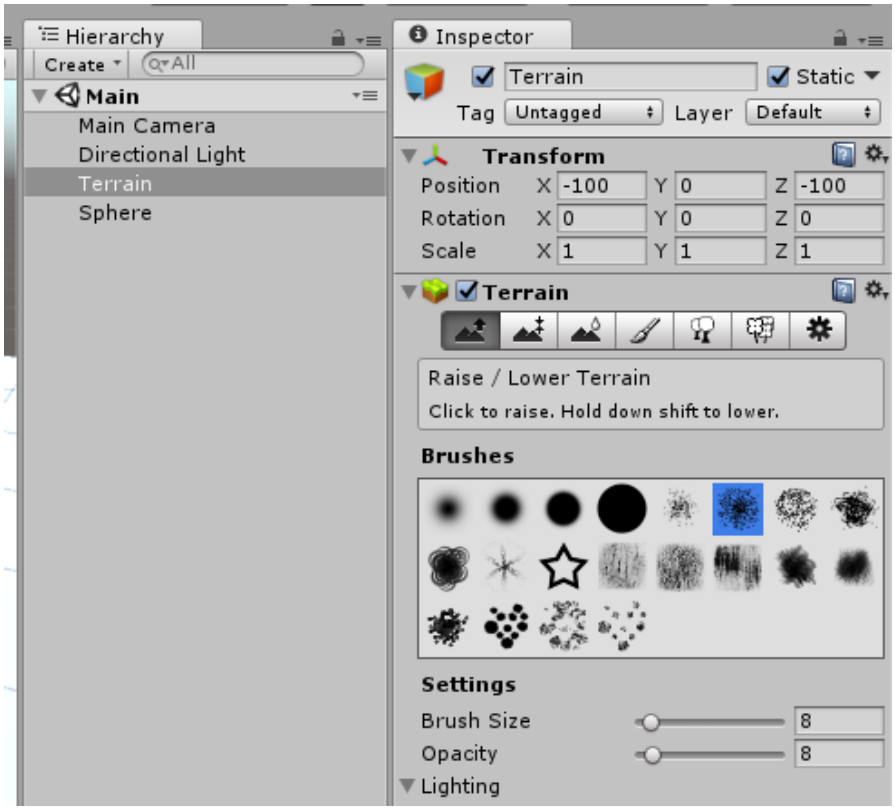
3. Using Scene drag camera so it shows the Sphere in the Camera Preview window.

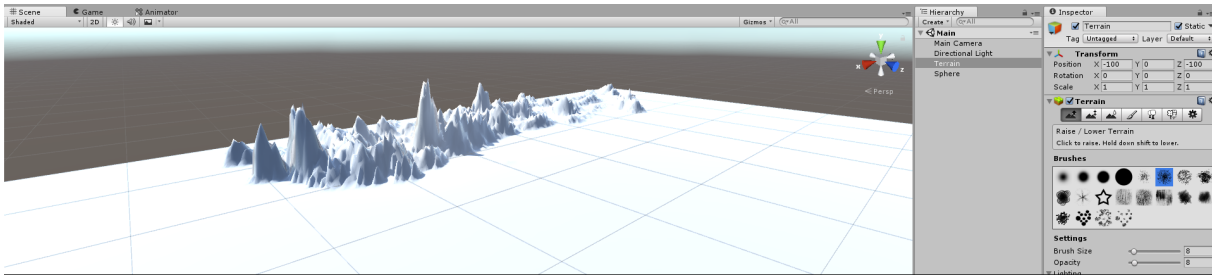


4. Run project

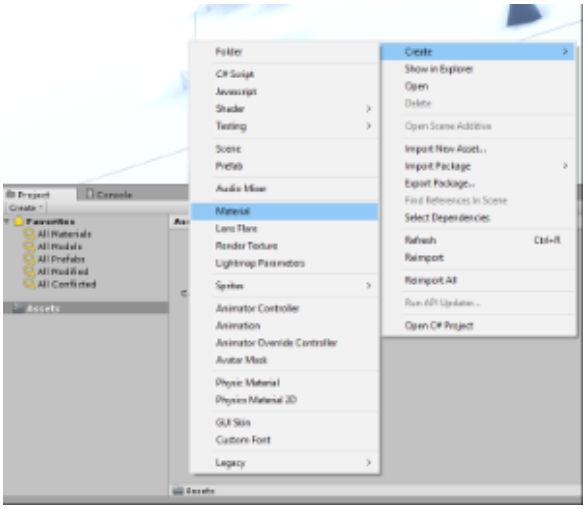
Unity Simple Ball Rolling Game: Lesson 3 - Add Detail to Terrain

1. Create Mountains
    a. Select Terrain from Main scene
    b. In Inspector window select Raise / Lower Terrain.
    c. Select a brush
    d. Set the brush size and opacity.

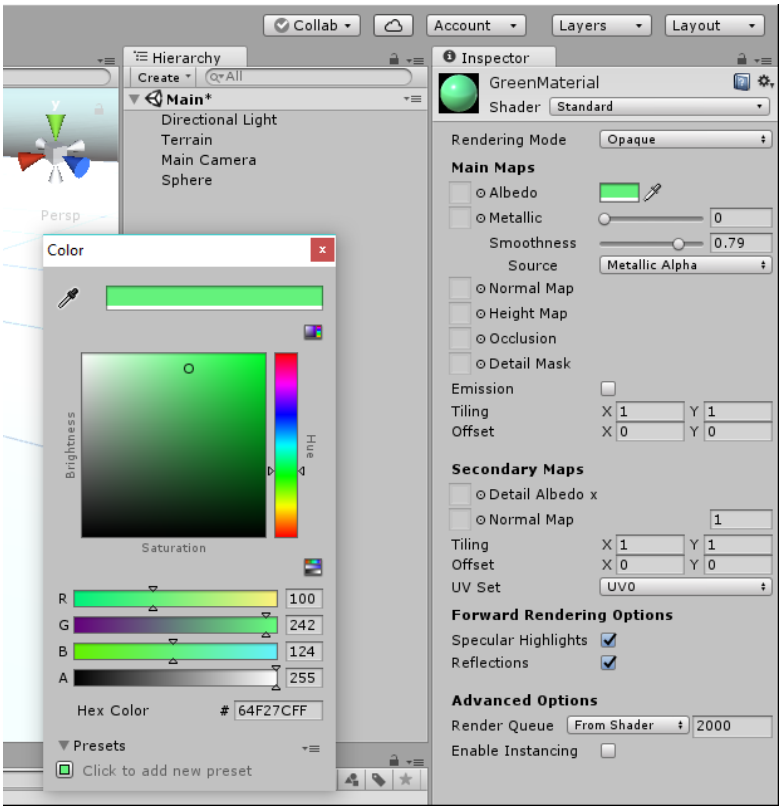




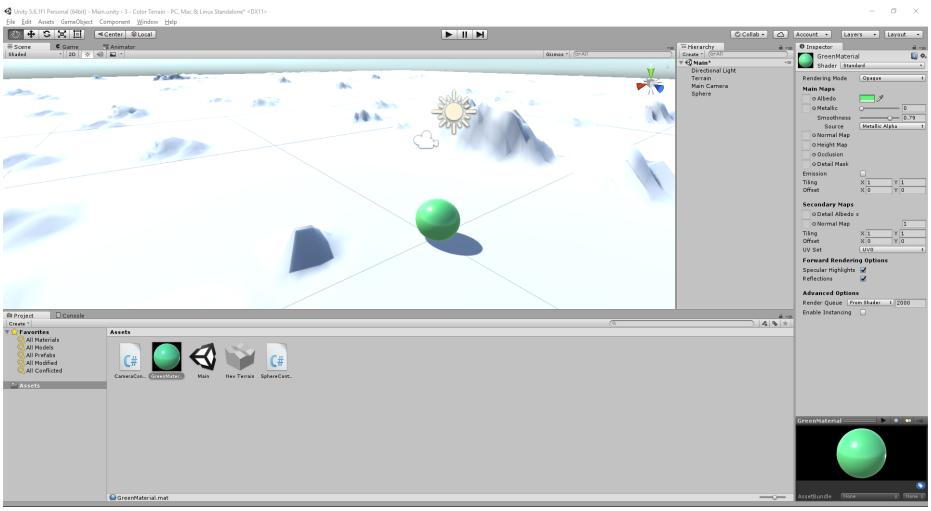
2. Color the Sphere
3. Create new Material Asset
4. Rename material to GreenMaterial



5. Change color to a shade of green



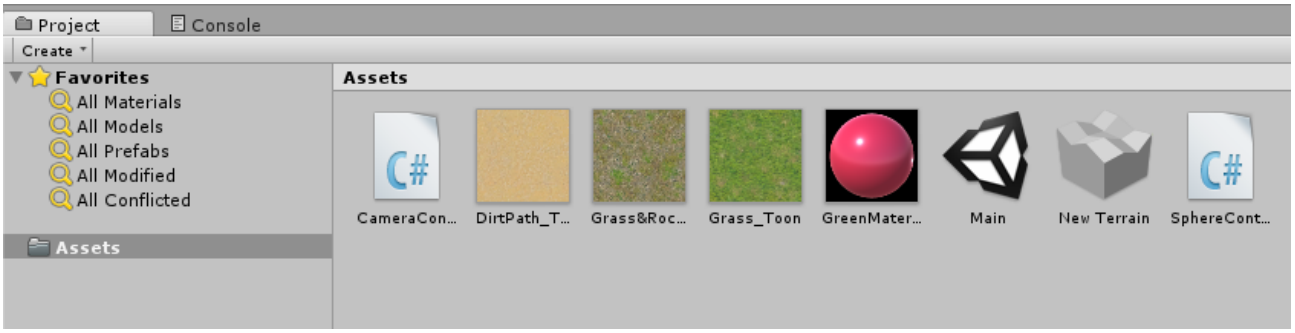
6. Drag GreenMaterial from Asset window onto Sphere in the Main Hierarchy windows.



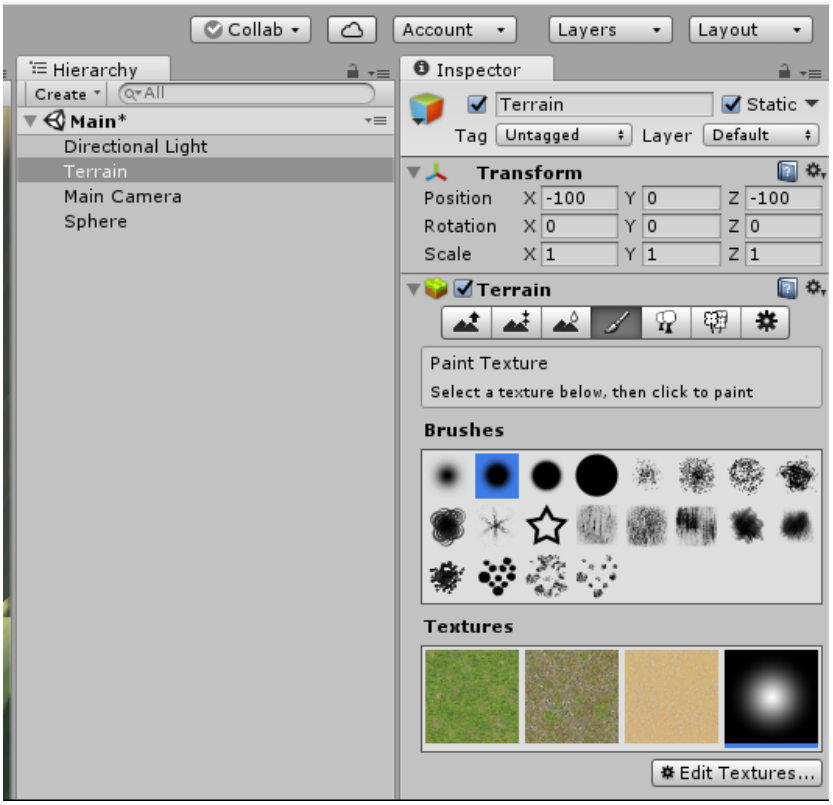
7. Copy texture files to asset folder.

Name	Date modified	Type	Size
CameraController.cs	5/31/2017 6:53 PM	Visual C# Source F...	1 KB
CameraController.cs.meta	5/24/2017 7:20 PM	META File	1 KB
DirtPath_Toon.psd	5/13/2017 11:01 AM	PSD File	1,561 KB
DirtPath_Toon.psd.meta	5/13/2017 11:01 AM	META File	1 KB
Grass&Rock_Toon.psd	5/13/2017 11:01 AM	PSD File	1,568 KB
Grass&Rock_Toon.psd.meta	5/13/2017 11:01 AM	META File	1 KB
Grass_Toon.psd	5/13/2017 11:01 AM	PSD File	1,567 KB
Grass_Toon.psd.meta	5/13/2017 11:01 AM	META File	1 KB
GreenMaterial.mat	5/31/2017 6:58 PM	Microsoft Access ...	5 KB
GreenMaterial.mat.meta	5/31/2017 6:58 PM	META File	1 KB

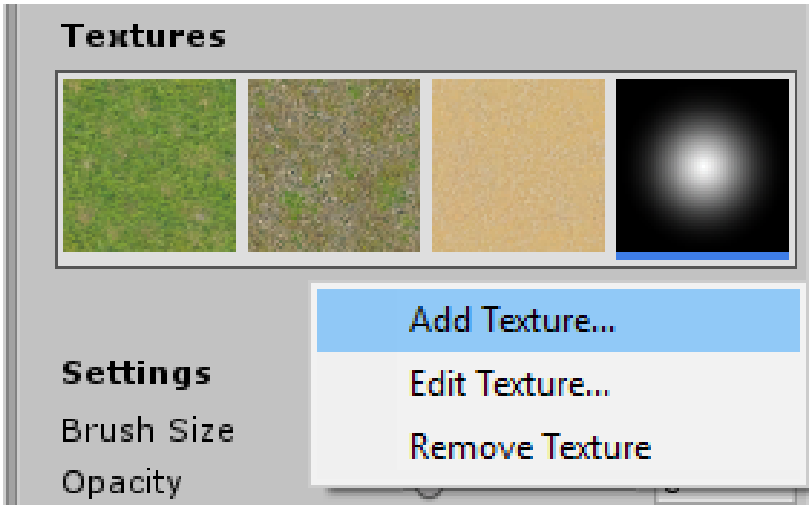
8. Copies terrains will appear in Unity Asset window.

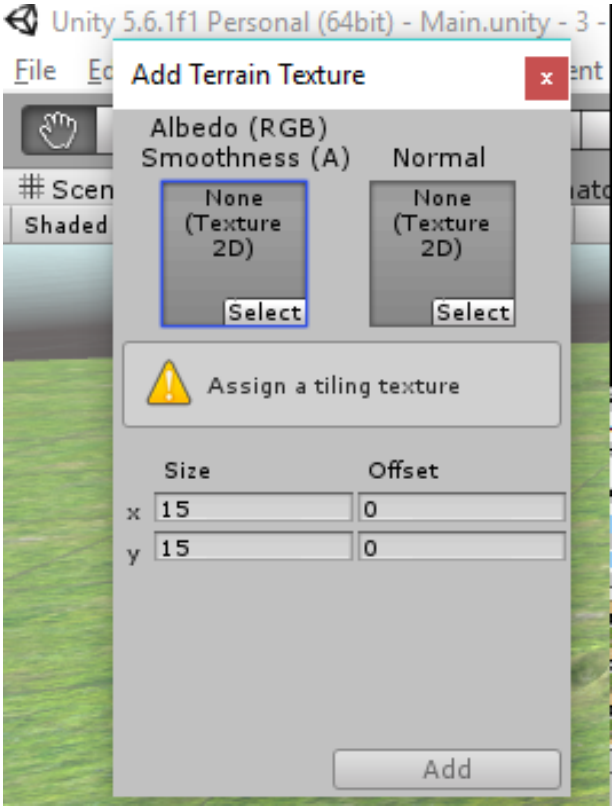


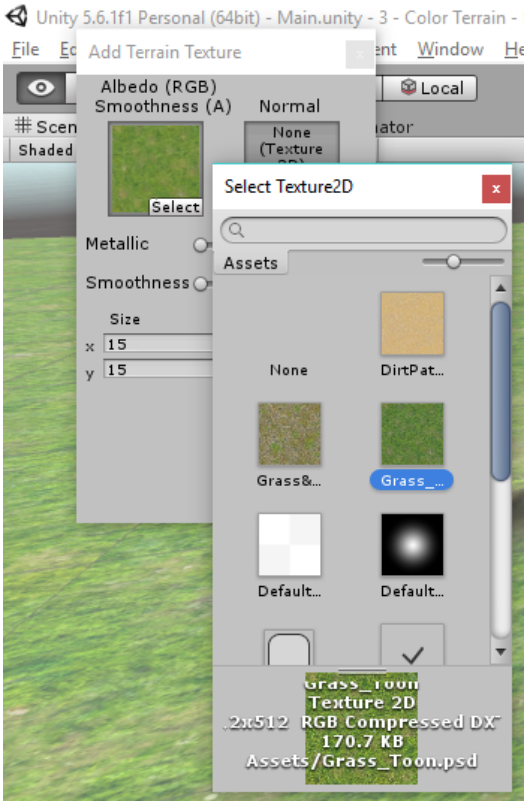
9. Click on Edit Textures button



10. Select Add Texture



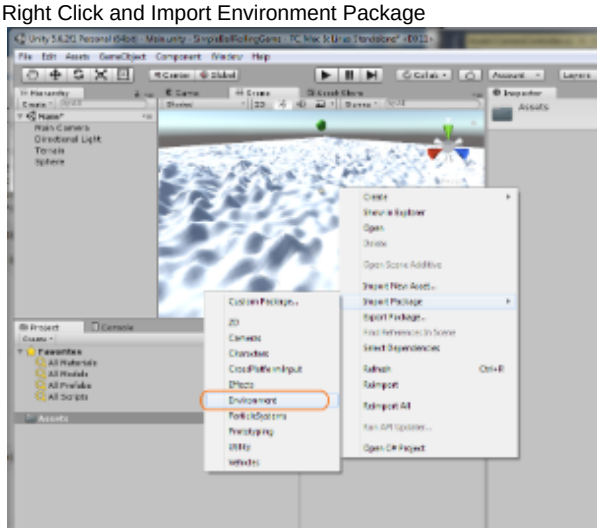

Unity 5.6.1f1 Personal (64bit) - Main.unity - 3 -
File
Add Terrain Texture
Albedo (RGB)
Smoothness (A)
Normal
None
(Texture
2D)
Select
None
(Texture
2D)
Select
Assign a tiling texture
Size
Offset
x
15
0
y
15
0
Add


Unity 5.6.1f1 Personal (64bit) - Main.unity - 3 - Color Terrain -
File
Window
Add Terrain Texture
Albedo (RGB)
Smoothness (A)
Normal
Select
Metallic
Smoothness
Size
x
15
y
15
Local
Select Texture2D
Assets
None
DirtPat...
Grass&...
Grass_...
Default...
Default...
Texture 2D
RGB Compressed DX
170.7 KB
Assets/Grass_Toon.psd

11.

Unity Simple Ball Rolling Game: Lesson 3 - Adding Terrain Textures

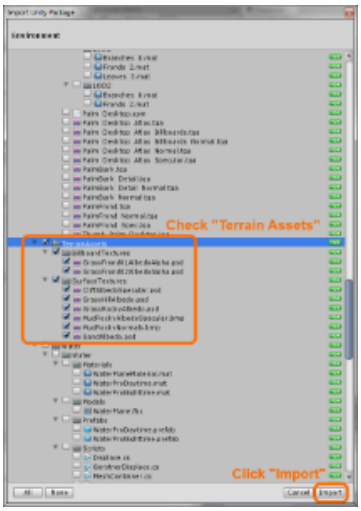
1. Right Click and Import Environment Package



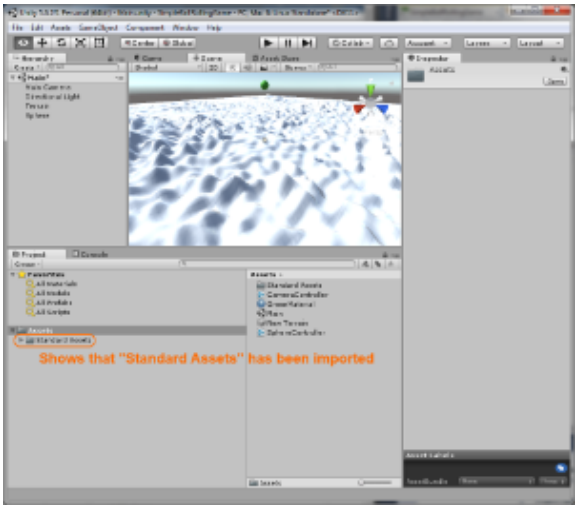
2. Uncheck Standard Assets so we don't get ALL packages

3. Showing Standard Assets is now unchecked

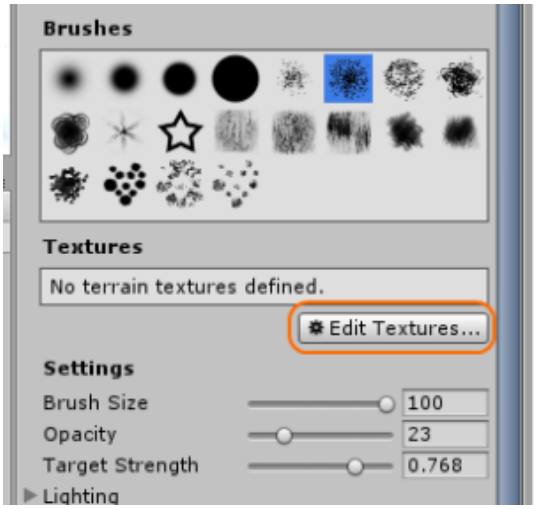
4. Find TerrianAssets and check it



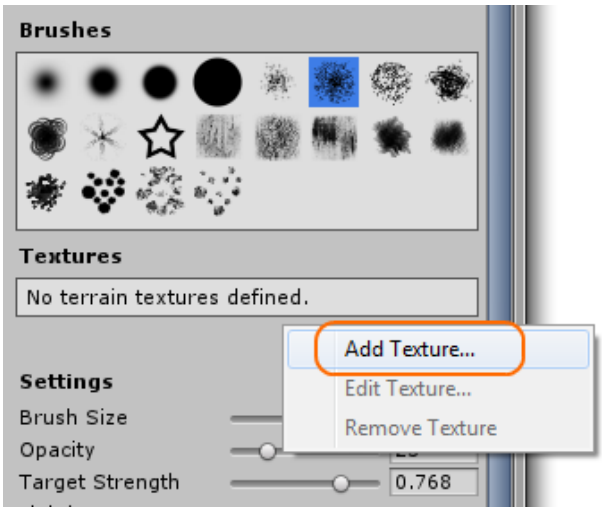
5. Verify Standard Assets has been imported



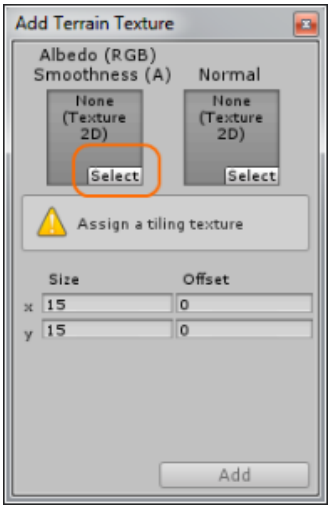
6. Click EDIT TEXTURE



7. Click ADD TEXTURE

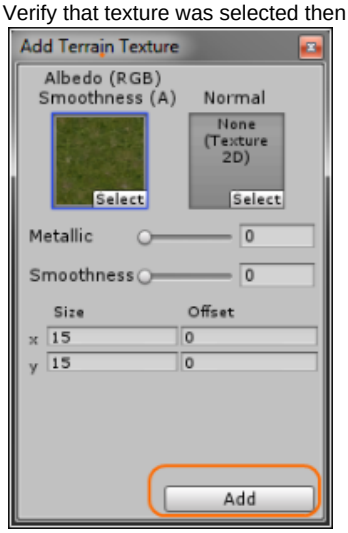


8. Press Select

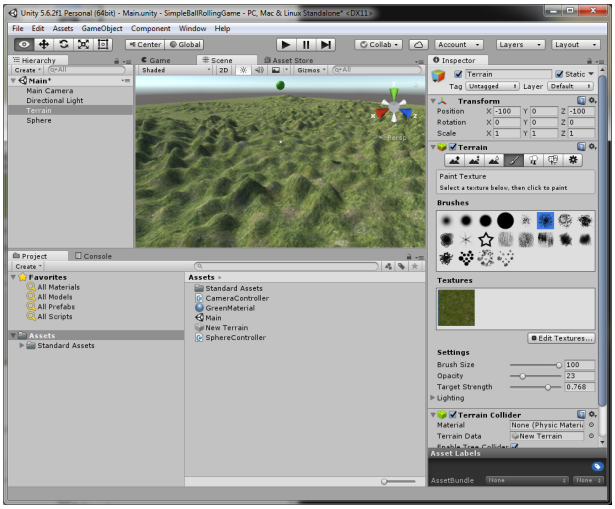


9. Choose a grass Texture

10. Verify that texture was selected then press ADD



11. Verify that texture was added to your Terrain



Unity Space Shooter Game

Unity Tutorial - https://unity3d.com/learn/tutorials/projects/space-shooter-tutorial

Source Code - https://github.com/jasonweibel/HyperStream-Unity

- Unity Space Shooter - Lesson 1 - Set Background and Moving of the Player
- Unity Space Shooter - Lesson 2 - Shooting shots
- Unity Space Shooter - Lesson 3 - Spawning Waves

Unity Space Shooter - Lesson 1 - Set Background and Moving of the Player

Reference

Unity Tutorial - https://unity3d.com/earn/tutorials/projects/space-shooter/moving-the-player?playlist=17147

Set Background

1. Open BGScroller script in Visual Studio
2. Uncomment scrollSpeed and tileSizeZ variables.

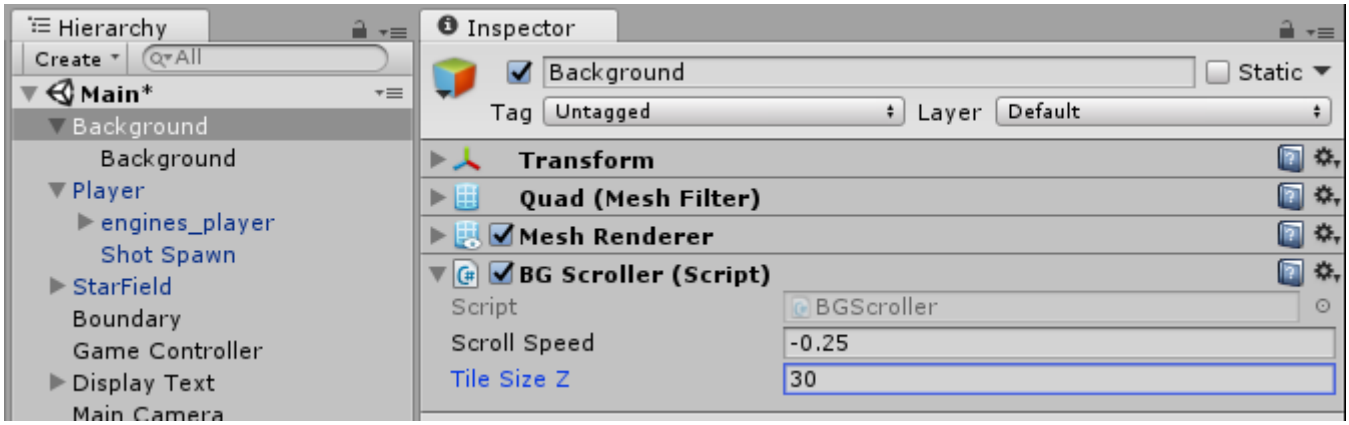
```
4  public class BGScroller : MonoBehaviour
5  {
6      //public float scrollSpeed;
7      //public float tileSizeZ;
```

3. Uncomment Update method

```
16     //void Update ()
17     //{
18     //  float newPosition = Mathf.Repeat(Time.time * scrollSpeed, tileSizeZ);
19     //  transform.position = startPosition + Vector3.forward * newPosition;
20     //}
```

4. Open Unity project
5. Set Scroll Speed and Tile Size Z values.
   Scroll Speed = -0.25
   Tile Size Z = 30
6. Let students play game and mess with these values.

Final Background - BG Scroller script values



Setup Player Movement

1. Open PlayerController script in Visual Studio
2. Uncomment speed and tilt variables

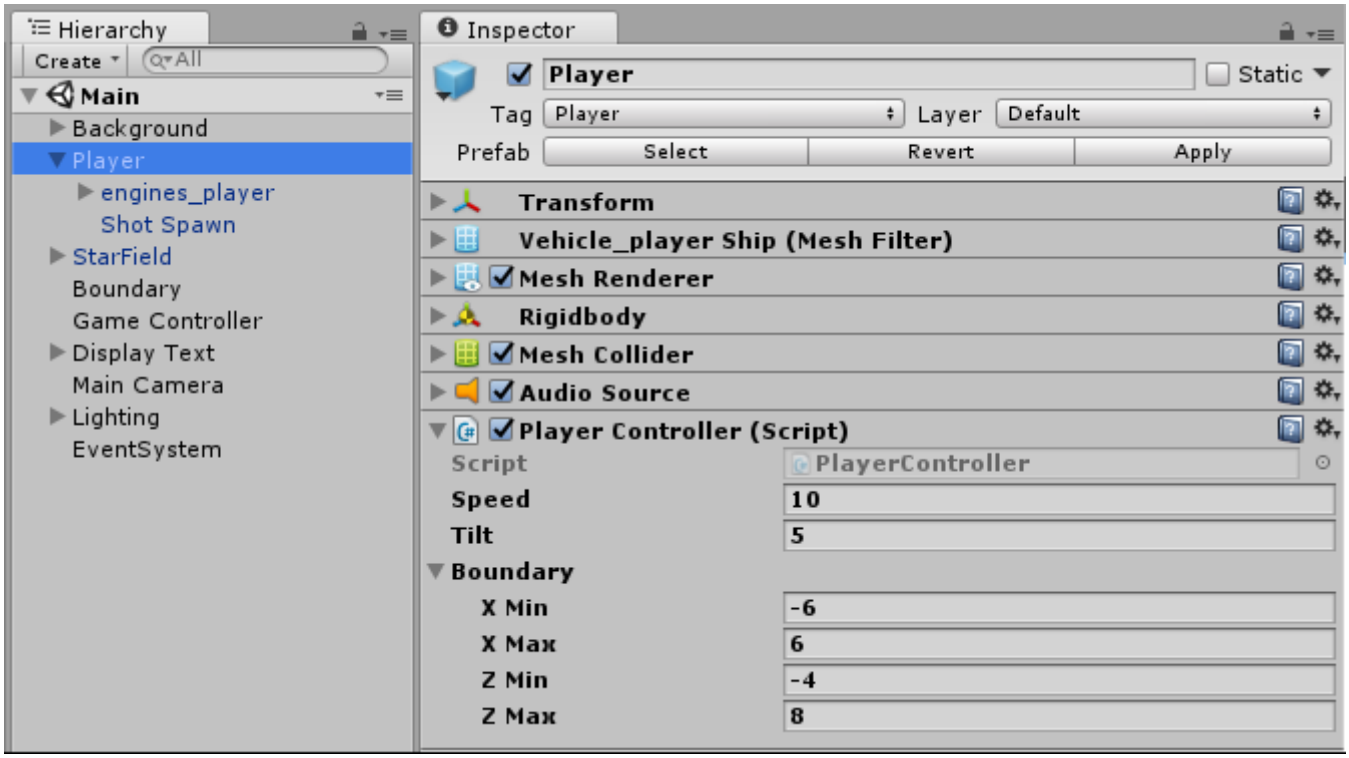
```
10  public class PlayerController : MonoBehaviour
11  {
12      public float speed;
13      public float tilt;
```

3. Uncomment FixedUpdate method

```
32  //   void FixedUpdate ()
33      //{
34      //  float moveHorizontal = Input.GetAxis ("Horizontal");
35      //  float moveVertical = Input.GetAxis ("Vertical");
36
37      //  Vector3 movement = new Vector3 (moveHorizontal, 0.0f, moveVertical);
38      //  GetComponent<Rigidbody>().velocity = movement * speed;
39
40      //  GetComponent<Rigidbody>().position = new Vector3
41      //  (
42      //      Mathf.Clamp (GetComponent<Rigidbody>().position.x, boundary.xMin, boundary.xMax),
43      //      0.0f,
44      //      Mathf.Clamp (GetComponent<Rigidbody>().position.z, boundary.zMin, boundary.zMax)
45      //  );
46
47      //  GetComponent<Rigidbody>().rotation = Quaternion.Euler (0.0f, 0.0f, GetComponent<Rigidbody>().velocity.x * -tilt);
48      //}
```

4. Open Unity Project
5. Select Player in Hierarchy window and set the Speed and Tilt.
    a. Tilt = 5
    b. Speed = 10
6. Let students play game and mess with these values.

Final Player - Player Controller Script Values



Unity Space Shooter - Lesson 2 - Shooting shots

Reference

Unity Tutorial - https://unity3d.com/learn/tutorials/projects/space-shooter/shooting-shots?playlist=17147

Add Movement to the Player Object

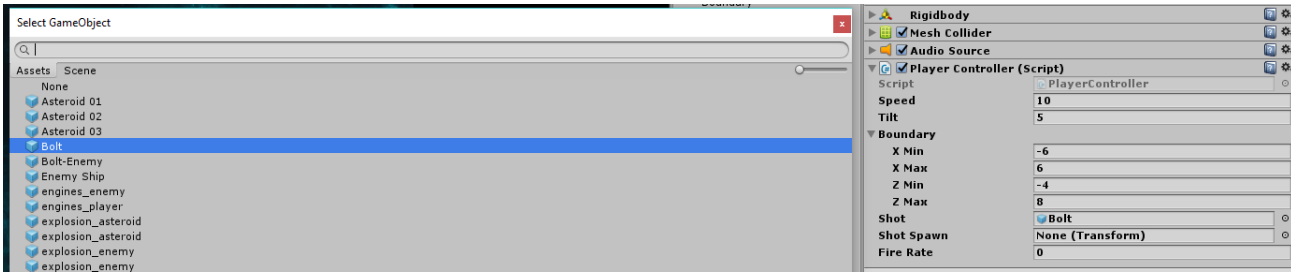
1. Open PlayerController script in Visual Studio
2. Uncomment shot, shotSpawn, and fireRate variables.

```
10  public class PlayerController : MonoBehaviour
11  {
12      public float speed;
13      public float tilt;
14      public Boundary boundary;
15
16      //public GameObject shot;
17      //public Transform shotSpawn;
18      //public float fireRate;
```

3. Uncomment Update method

```
22      //void Update()
23      //{
24      //    if (Input.GetButton("Fire1") && Time.time > nextFire)
25      //    {
26      //        nextFire = Time.time + fireRate;
27      //        Instantiate(shot, shotSpawn.position, shotSpawn.rotation);
28      //        GetComponent<AudioSource>().Play();
29      //    }
30      //}
```

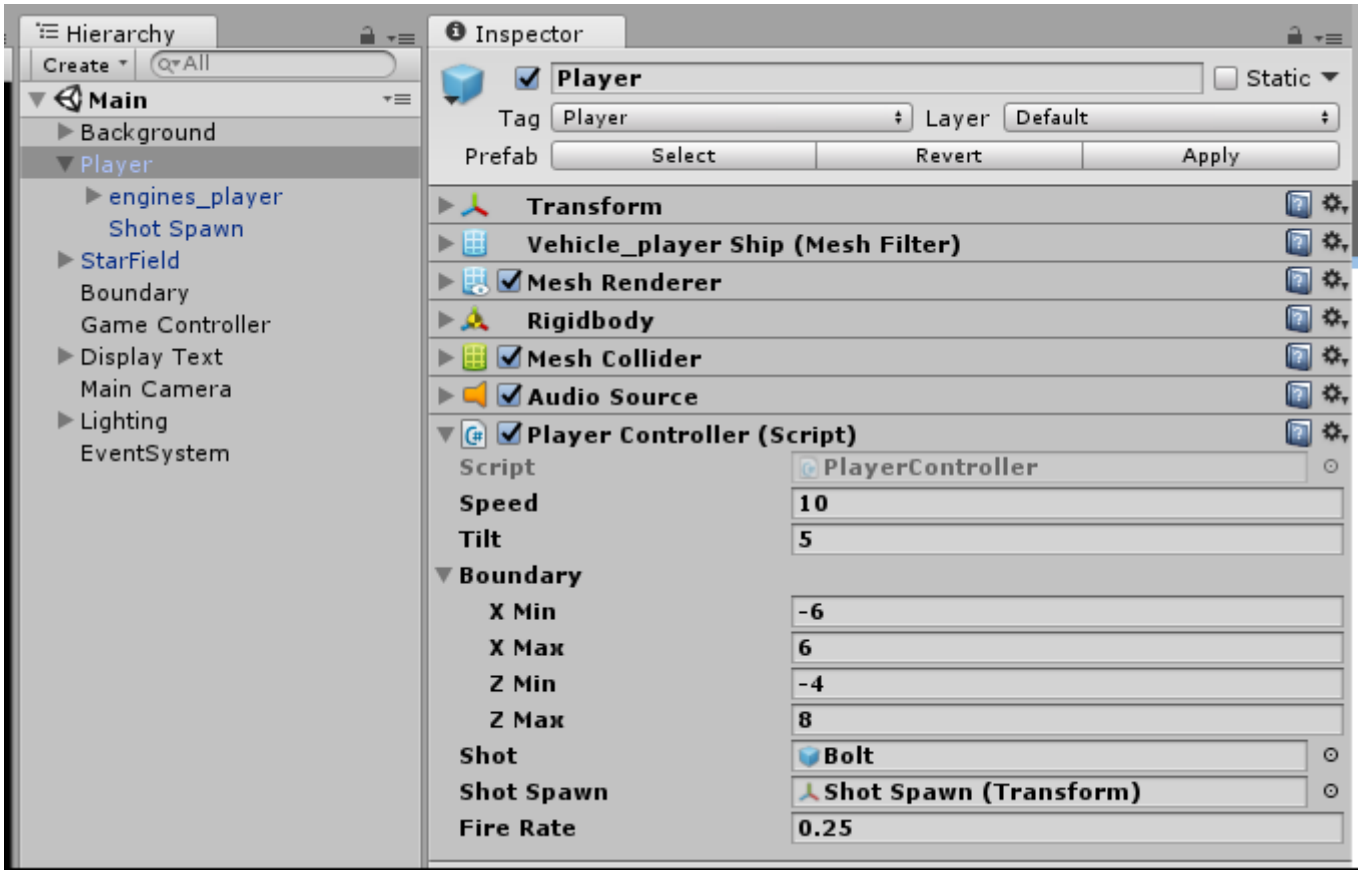
4. Switch to Unity project
5. Update Player - Player Controller script values.
6. Set Shot to Bolt



7. Set Shot Spawn to Shot Spawn
8. Set Fire Rate to 0.25

Have students run the game and press the Ctrl button to fire.

Finished Values



Unity Space Shooter - Lesson 3 - Spawning Waves

Reference

Unity Tutorial - https://unity3d.com/learn/tutorials/projects/space-shooter/spawning-waves?playlist=17147

Spawning Waves

1. Open GameController script in Visual Studio
2. Uncomment public hazards variable

```
5  public class GameController : MonoBehaviour
6  {
7      //public GameObject[] hazards;
8      public Vector3 spawnValues;
9      public int hazardCount;
10     public float spawnWait;
11     public float startWait;
12     public float waveWait;
```

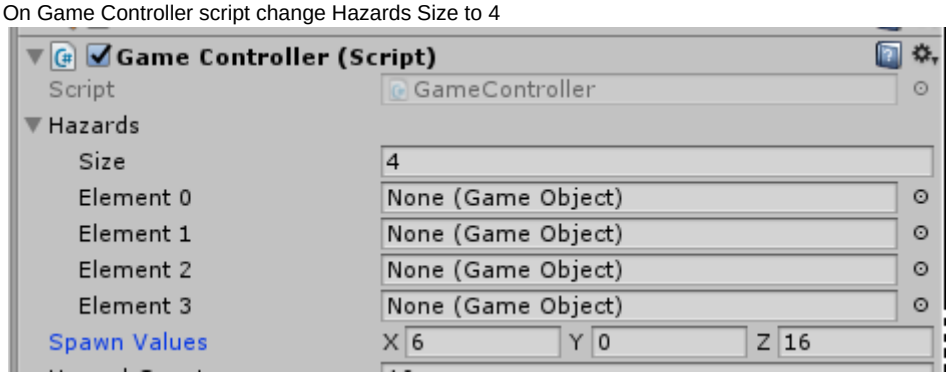
3. Uncomment SpawnWaves method

```
44  //IEnumerator SpawnWaves ()
45  //{
46  //  yield return new WaitForSeconds (startWait);
47  //  while (true)
48  //  {
49  //      for (int i = 0; i < hazardCount; i++)
50  //      {
51  //          GameObject hazard = hazards [Random.Range (0, hazards.Length)];
52  //          Vector3 spawnPosition = new Vector3 (Random.Range (-spawnValues.x, spawnValues.x), spawnValues.y, spawnValues.z);
53  //          Quaternion spawnRotation = Quaternion.identity;
54  //          Instantiate (hazard, spawnPosition, spawnRotation);
55  //          yield return new WaitForSeconds (spawnWait);
56  //      }
57  //      yield return new WaitForSeconds (waveWait);
58
59  //      if (gameOver)
60  //      {
61  //          restartText.text = "Press 'R' for Restart";
62  //          restart = true;
63  //          break;
64  //      }
65  //  }
66  //}
```

4. Uncomment StartCoroutine statement in Start method.

```
22  void Start ()
23  {
24      gameOver = false;
25      restart = false;
26      restartText.text = "";
27      gameOverText.text = "";
28      score = 0;
29      UpdateScore ();
30      //StartCoroutine (SpawnWaves ());
31  }
```

5. Switch to Unity project and locate Game Controller in Hierarchy window.
6. On Game Controller script change Hazards Size to 4



7. Change Element 0 to Asteroid 01
8. Change Element 1 to Asteroid 02
9. Change Element 2 to Asteroid 3
10. Change Element 3 to Enemy Ship
11. Play Game!

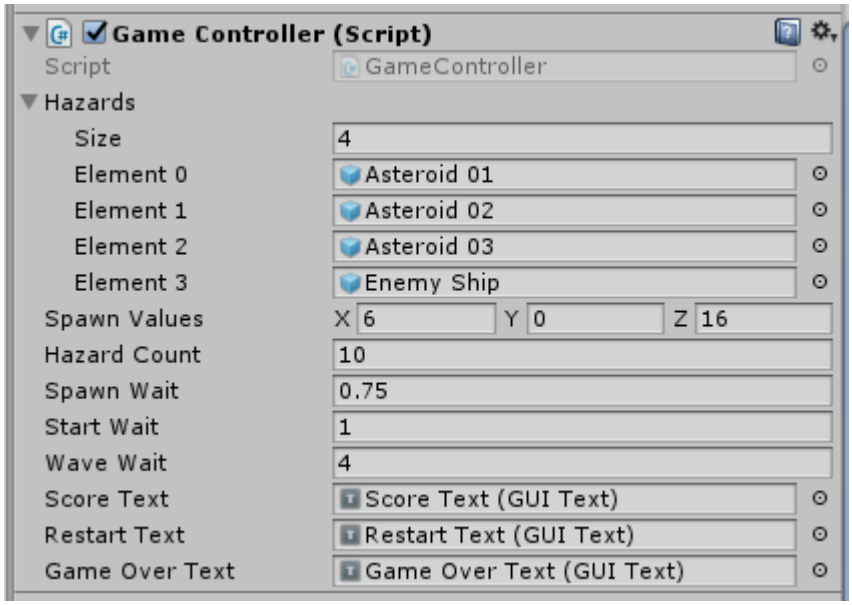
Have students change the game:

1. Modify Game Controller values
    a. Change Spawn Wait to 0.1

b. Change Hazard Count to 100
c. Change Hazard Count to 10000

2. Modify Player Controller Script values
    a. Change Speed to 1
    b. Change Speed to 100
    c. Change Fire Rate to 1
    d. Change Fire Rate to 0

Finished Game Controller Script Inspector Window



Unity Survival Shooter Game

These lessons are based on the Survival Shooter tutorials provided by Unity. They are a simplified version of those lessons so students see a working game sooner. Deviate from the lessons as you see appropriate.

Survival Shooter - https://unity3d.com/learn/tutorials/projects/survival-shooter-tutorial

Source Code - https://github.com/jasonweibel/HyperStream-Unity

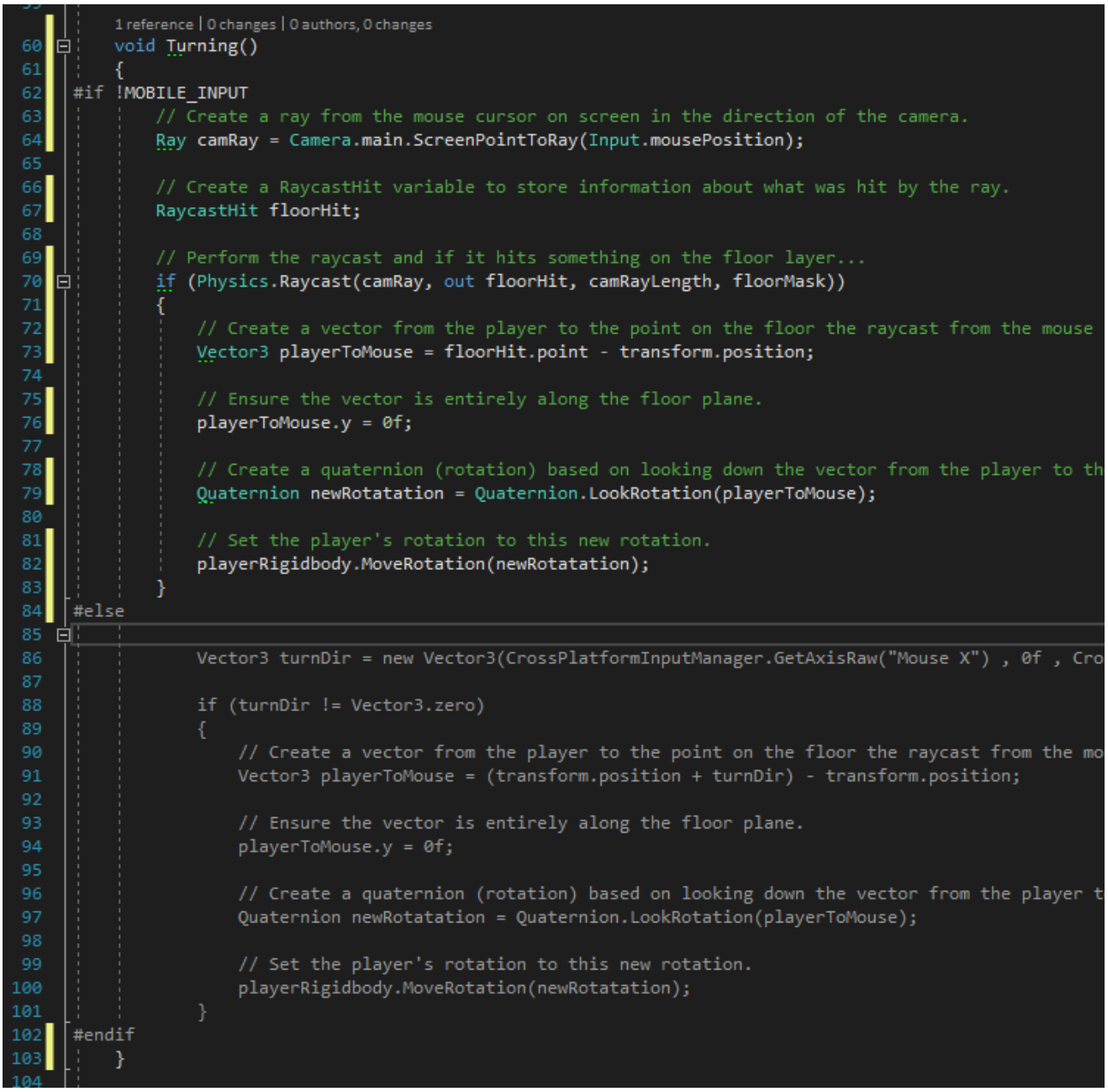
- Unity Survival Shooter Game - Lesson 1 - Player Character
- Unity Survival Shooter Game - Lesson 2 - Camera Setup
- Unity Survival Shooter Game - Lesson 3 - Spawning Enemies
- Unity Survival Shooter Game - Lesson 4 - Player Health
- Unity Survival Shooter Game - Lesson 5 - Scoring Points & Game Over

Unity Survival Shooter Game - Lesson 1 - Player Character

1. Open Player Movement script in Visual Studio
2. Locate Move method and uncomment it.

```
46
    1 reference | 0 changes | 0 authors, 0 changes
47  void Move(float h, float v)
48  {
49      // Set the movement vector based on the axis input.
50      movement.Set(h, 0f, v);
51
52      // Normalise the movement vector and make it proportional to the speed per second.
53      movement = movement.normalized * speed * Time.deltaTime;
54
55      // Move the player to it's current position plus the movement.
56      playerRigidbody.MovePosition(transform.position + movement);
57  }
58
```

3. Locate Turning Method and uncomment it.



```
    1 reference | 0 changes | 0 authors, 0 changes
    void Turning()
    {
#if !MOBILE_INPUT
        // Create a ray from the mouse cursor on screen in the direction of the camera.
        Ray camRay = Camera.main.ScreenPointToRay(Input.mousePosition);

        // Create a RaycastHit variable to store information about what was hit by the ray.
        RaycastHit floorHit;

        // Perform the raycast and if it hits something on the floor layer...
        if (Physics.Raycast(camRay, out floorHit, camRayLength, floorMask))
        {
            // Create a vector from the player to the point on the floor the raycast from the mouse
            Vector3 playerToMouse = floorHit.point - transform.position;

            // Ensure the vector is entirely along the floor plane.
            playerToMouse.y = 0f;

            // Create a quaternion (rotation) based on looking down the vector from the player to th
            Quaternion newRotatation = Quaternion.LookRotation(playerToMouse);

            // Set the player's rotation to this new rotation.
            playerRigidbody.MoveRotation(newRotatation);
        }
#else

            Vector3 turnDir = new Vector3(CrossPlatformInputManager.GetAxisRaw("Mouse X") , 0f , Cro

            if (turnDir != Vector3.zero)
            {
                // Create a vector from the player to the point on the floor the raycast from the mo
                Vector3 playerToMouse = (transform.position + turnDir) - transform.position;

                // Ensure the vector is entirely along the floor plane.
                playerToMouse.y = 0f;

                // Create a quaternion (rotation) based on looking down the vector from the player t
                Quaternion newRotatation = Quaternion.LookRotation(playerToMouse);

                // Set the player's rotation to this new rotation.
                playerRigidbody.MoveRotation(newRotatation);
            }
#endif
    }
```

4. In FixedUpdate method uncomment call of Move and Turning methods.

```
0 references | 0 changes | 0 authors, 0 changes
void FixedUpdate()
{
    // Store the input axes.
    float h = CrossPlatformInputManager.GetAxisRaw("Horizontal");
    float v = CrossPlatformInputManager.GetAxisRaw("Vertical");

    // Move the player around the scene.
    Move(h, v);

    // Turn the player to face the mouse cursor.
    Turning();

    // Animate the player.
    Animating(h, v);
}
```

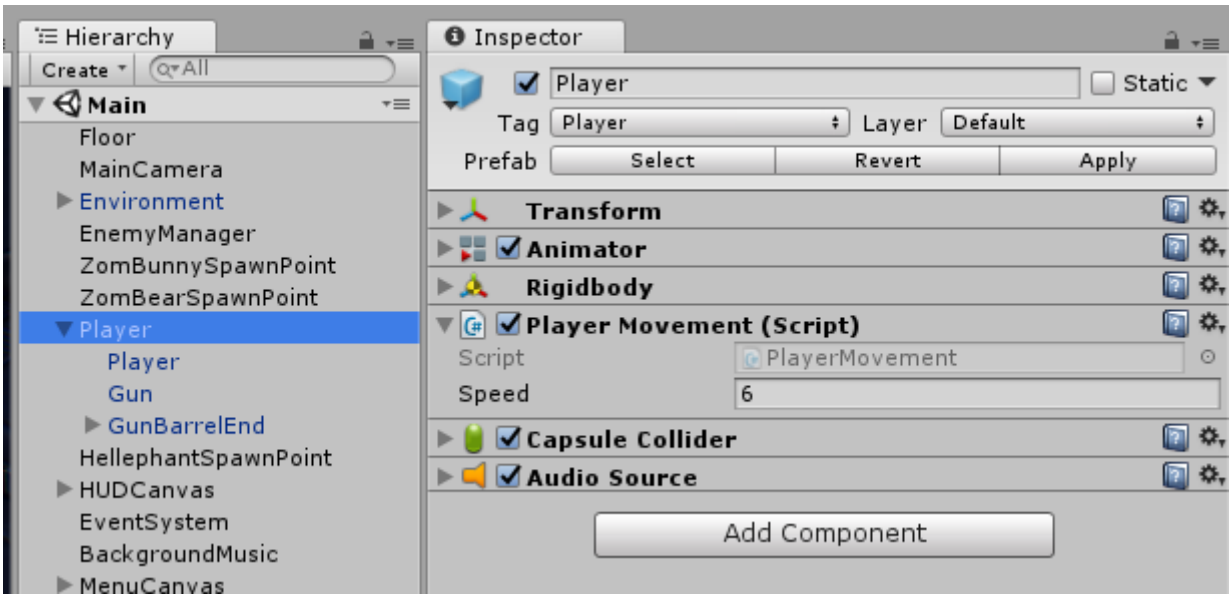
5. Open PlayerShooting script in Visual Studio
6. Uncomment Shoot method

```
void Shoot ()
{
    // Reset the timer.
    timer = 0f;

    // Play the gun shot audioclip.
    gunAudio.Play ();

    // Enable the lights.
    gunLight.enabled = true;
    faceLight.enabled = true;

    // Stop the particles from playing if they were, then start the particles.
    gunParticles.Stop ();
    gunParticles.Play ();

    // Enable the line renderer and set it's first position to be the end of the gun.
    gunLine.enabled = true;
    gunLine.SetPosition (0, transform.position);

    // Set the shootRay so that it starts at the end of the gun and points forward from the barrel.
    shootRay.origin = transform.position;
    shootRay.direction = transform.forward;

    // Perform the raycast against gameobjects on the shootable layer and if it hits something...
    if(Physics.Raycast (shootRay, out shootHit, range, shootableMask))
    {
        // Try and find an EnemyHealth script on the gameobject hit.
        EnemyHealth enemyHealth = shootHit.collider.GetComponent <EnemyHealth> ();

        // If the EnemyHealth component exist...
        if(enemyHealth != null)
        {
            // ... the enemy should take damage.
            enemyHealth.TakeDamage (damagePerShot, shootHit.point);
        }

        // Set the second position of the line renderer to the point the raycast hit.
        gunLine.SetPosition (1, shootHit.point);
    }
    // If the raycast didn't hit anything on the shootable layer...
    else
    {
        // ... set the second position of the line renderer to the fullest extent of the gun's range.
        gunLine.SetPosition (1, shootRay.origin + shootRay.direction * range);
    }
}
```

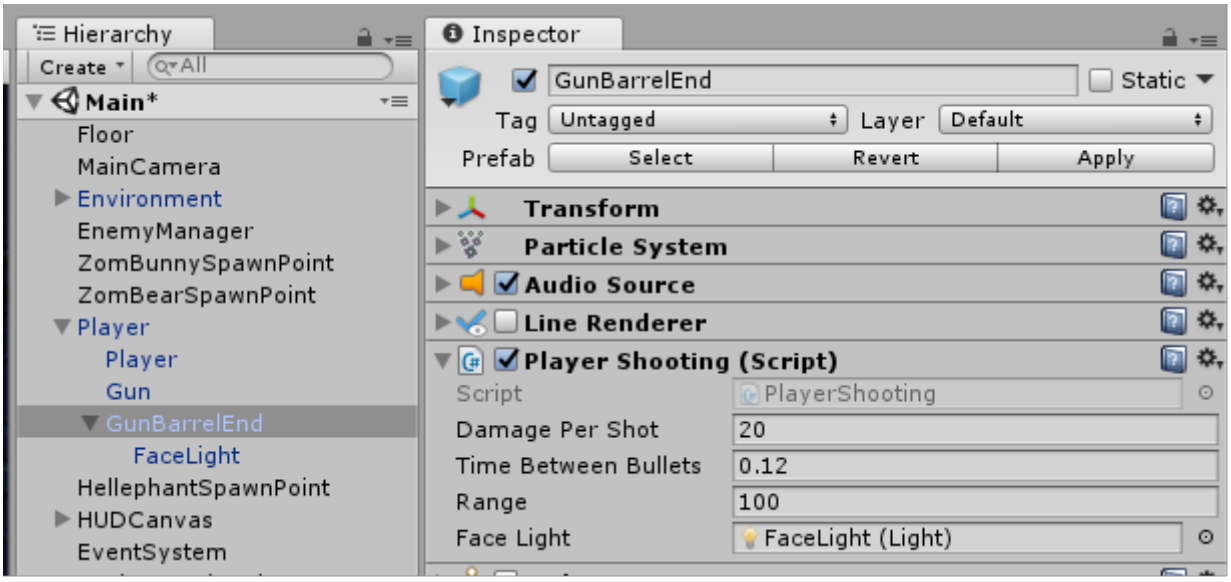
7. In Update method uncomment Shoot method call on line 47.

```
43          // If the Fire1 button is being press and it's time to fire...
44          if(Input.GetButton ("Fire1") && timer >= timeBetweenBullets && Time.timeScale != 0)
45          {
46              // ... shoot the gun.
47              Shoot ();
48          }
```

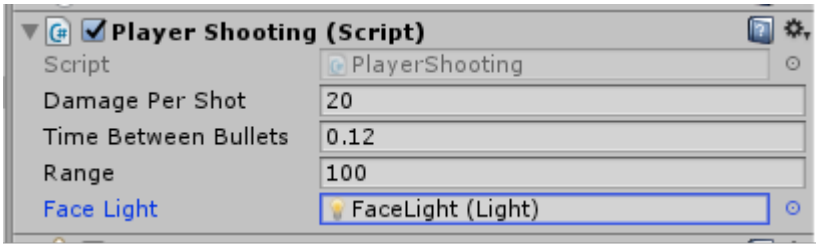
8. Add PlayerMovement script component to Player game object
9. Set speed to 6
10.



11. Add Player Shooting script to the GunBarrelEnd game object.



12. Set Damage Per Shot = 20, Time Between Bullets = 0.12, Range = 100, Face Light = FaceLight light

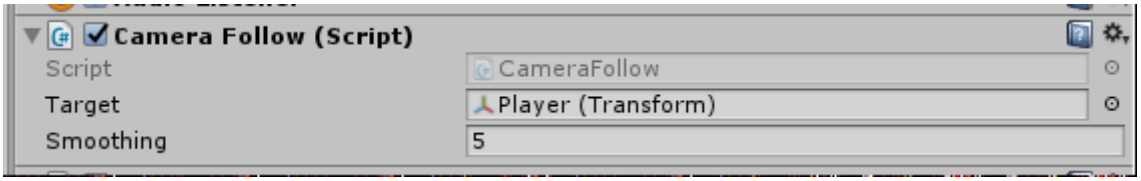


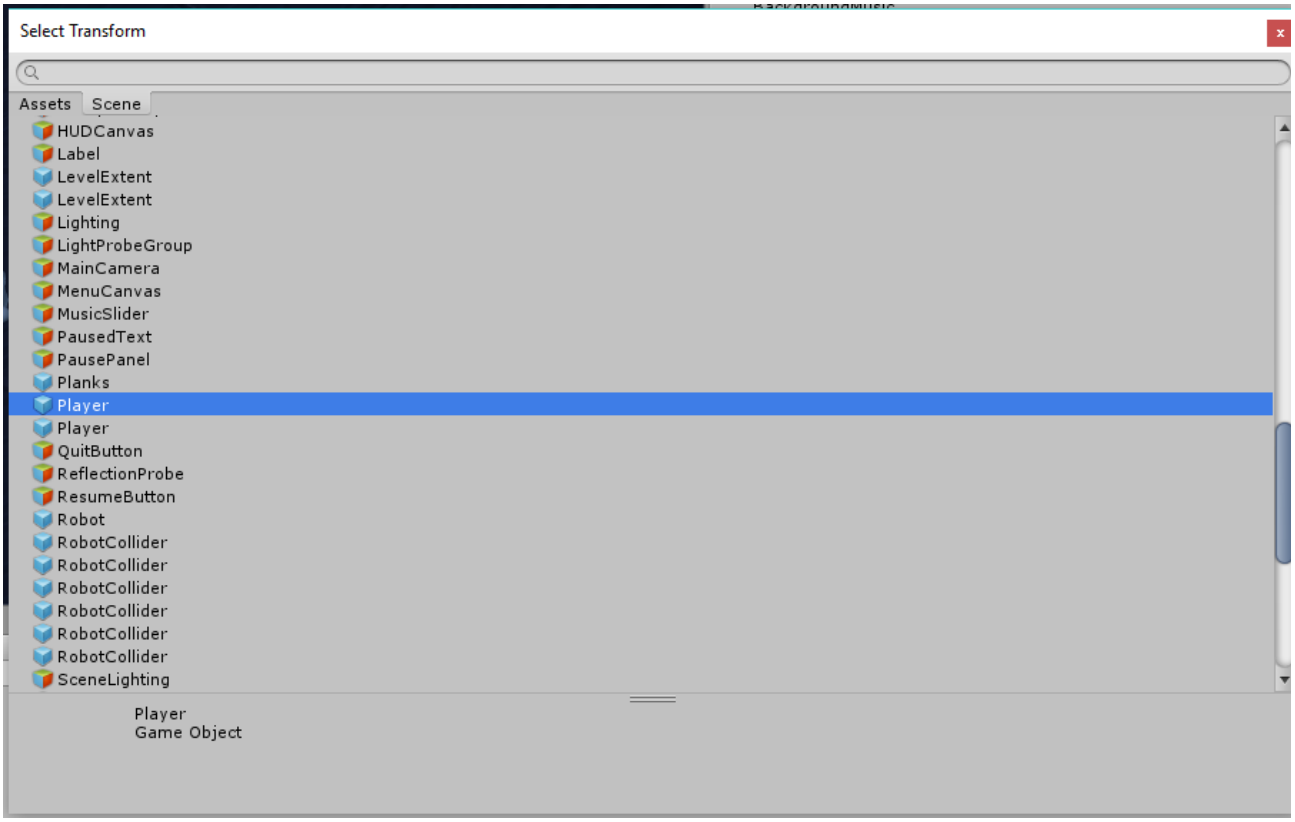
Unity Survival Shooter Game - Lesson 2 - Camera Setup

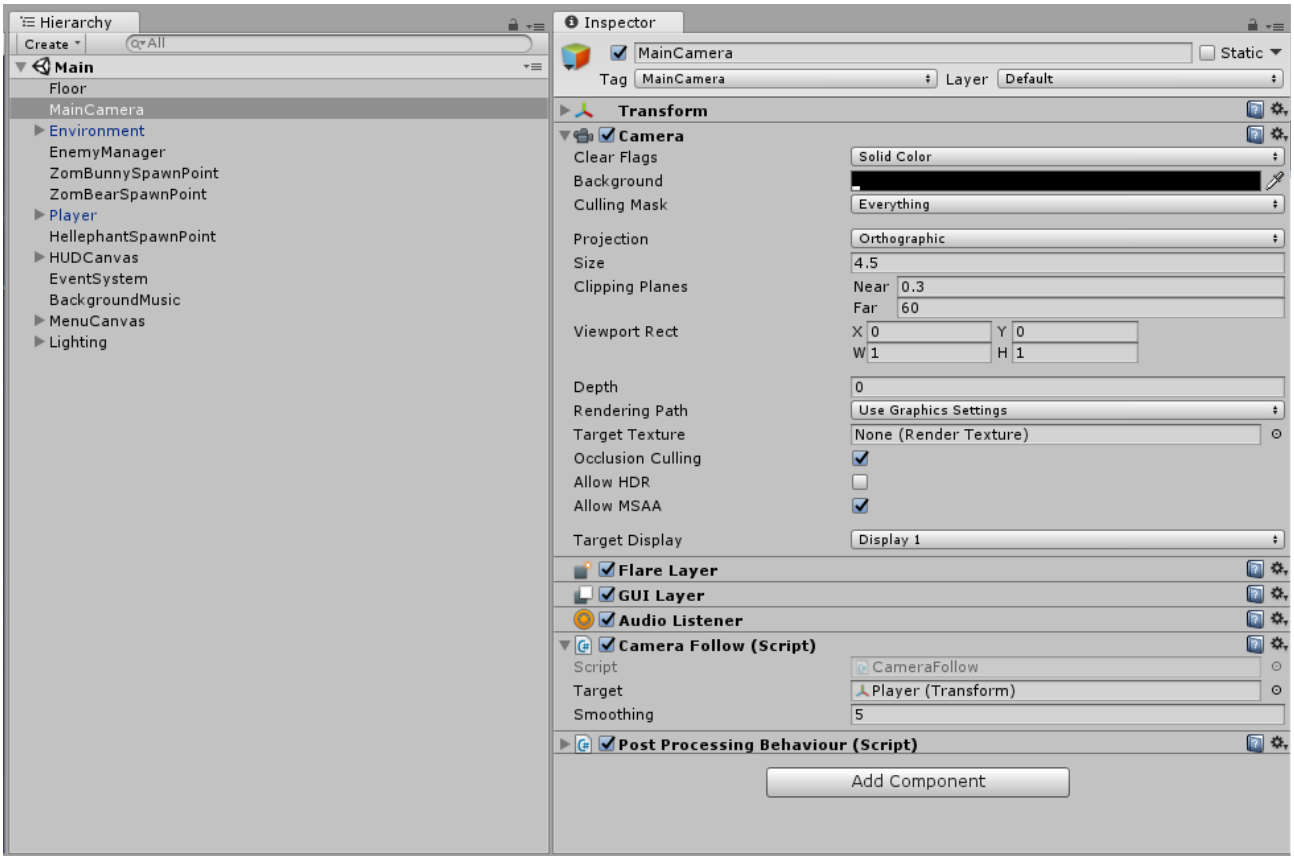
1. Open CameraFollow script in Visual Studio
2. Uncomment FixedUpdate Method

```
0 references | 0 changes | 0 authors, 0 changes
void FixedUpdate ()
{
    // Create a postion the camera is aiming for based on the offset from the target.
    Vector3 targetCamPos = target.position + offset;

    // Smoothly interpolate between the camera's current position and it's target position.
    transform.position = Vector3.Lerp (transform.position, targetCamPos, smoothing * Time.deltaTime);
}
```

3. Select MainCamera in Main scene.
4. Add CameraFollow Script to MainCamera game object.
5. Set Target to Player Transform and Smoothing = 5







6.

Unity Survival Shooter Game - Lesson 3 - Spawning Enemies

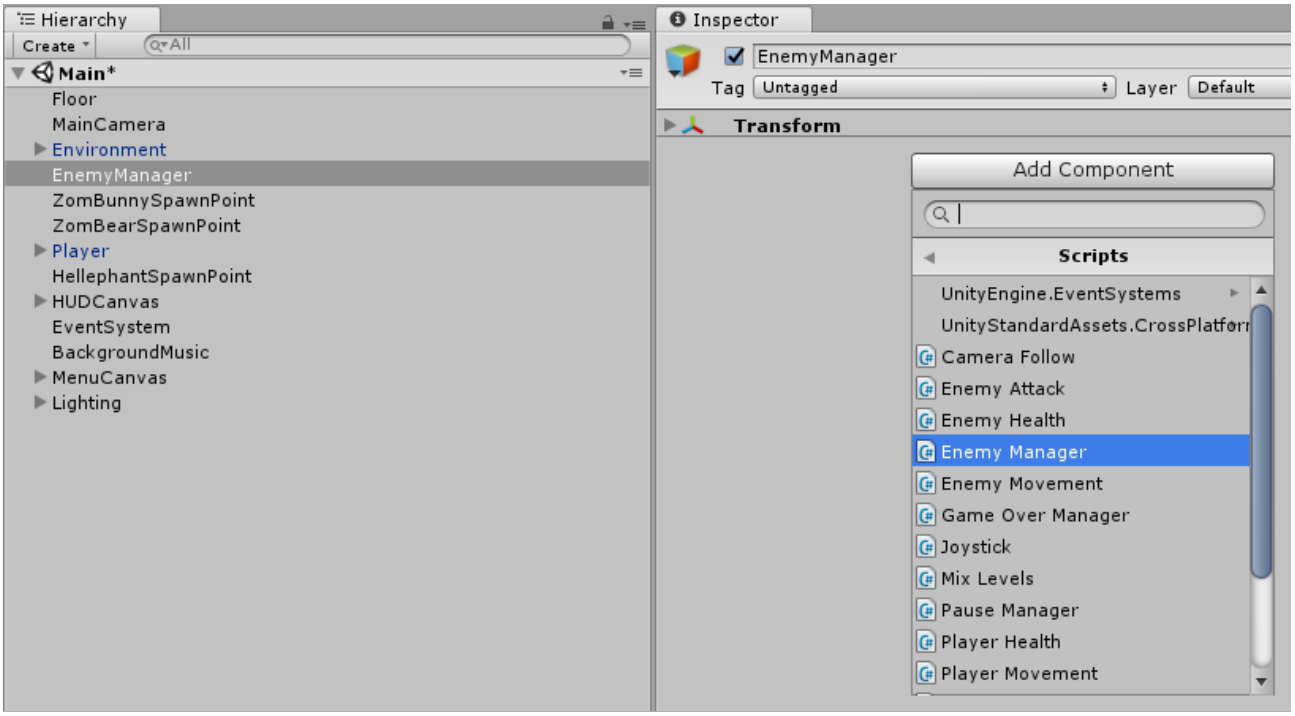
1. Open EnemyManager script in Visual Studio
2. Uncomment Spawn method.

```
0 references | 0 changes | 0 authors, 0 changes
void Spawn ()
{
    // If the player has no health left...
    //if(playerHealth.currentHealth <= 0f)
    //{
    //    // ... exit the function.
    //    return;
    //}

    // Find a random index between zero and one less than the number of spawn points.
    int spawnPointIndex = Random.Range (0, spawnPoints.Length);

    // Create an instance of the enemy prefab at the randomly selected spawn point's position and rotation.
    Instantiate (enemy, spawnPoints[spawnPointIndex].position, spawnPoints[spawnPointIndex].rotation);
}
```

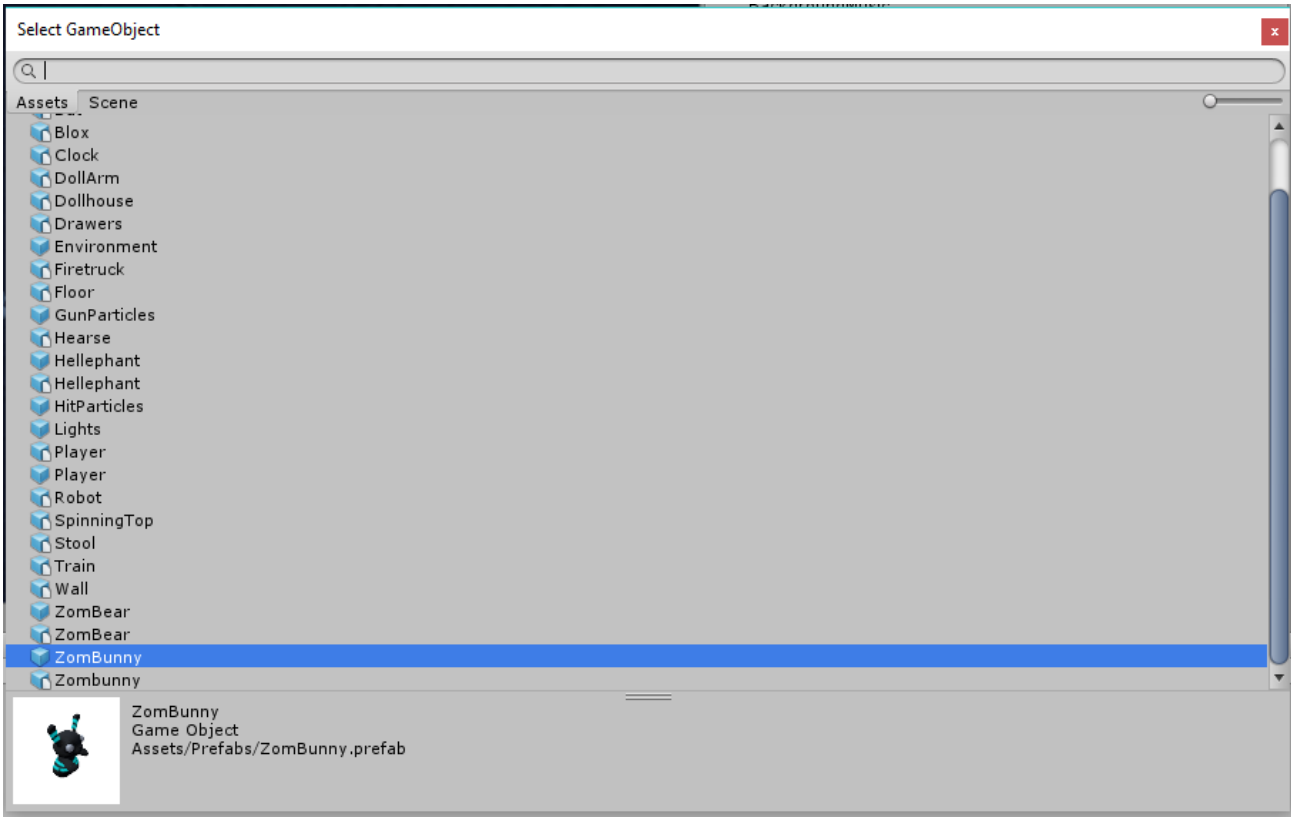
3. Uncomment InvokeRepeating in Start method.

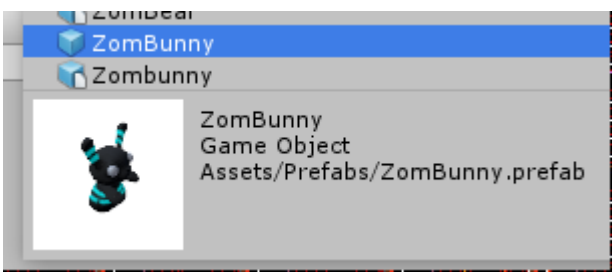
```
0 references | 0 changes | 0 authors, 0 changes
void Start ()
{
    // Call the Spawn function after a delay of the spawnTime and then continue to call after the same amount of time.
    InvokeRepeating ("Spawn", spawnTime, spawnTime);
}
```

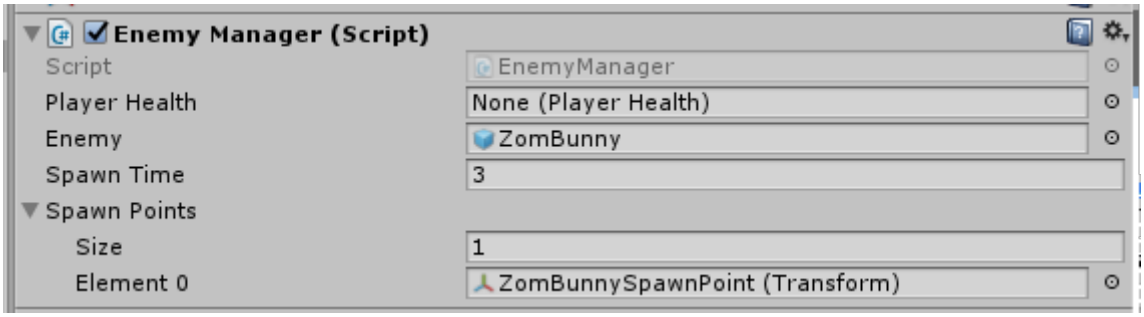
4. Switch to Unity and locate Enemy Manager game object on Main scene.
5. Add 3 Enemy Manager scripts to the Enemy Manager game object.



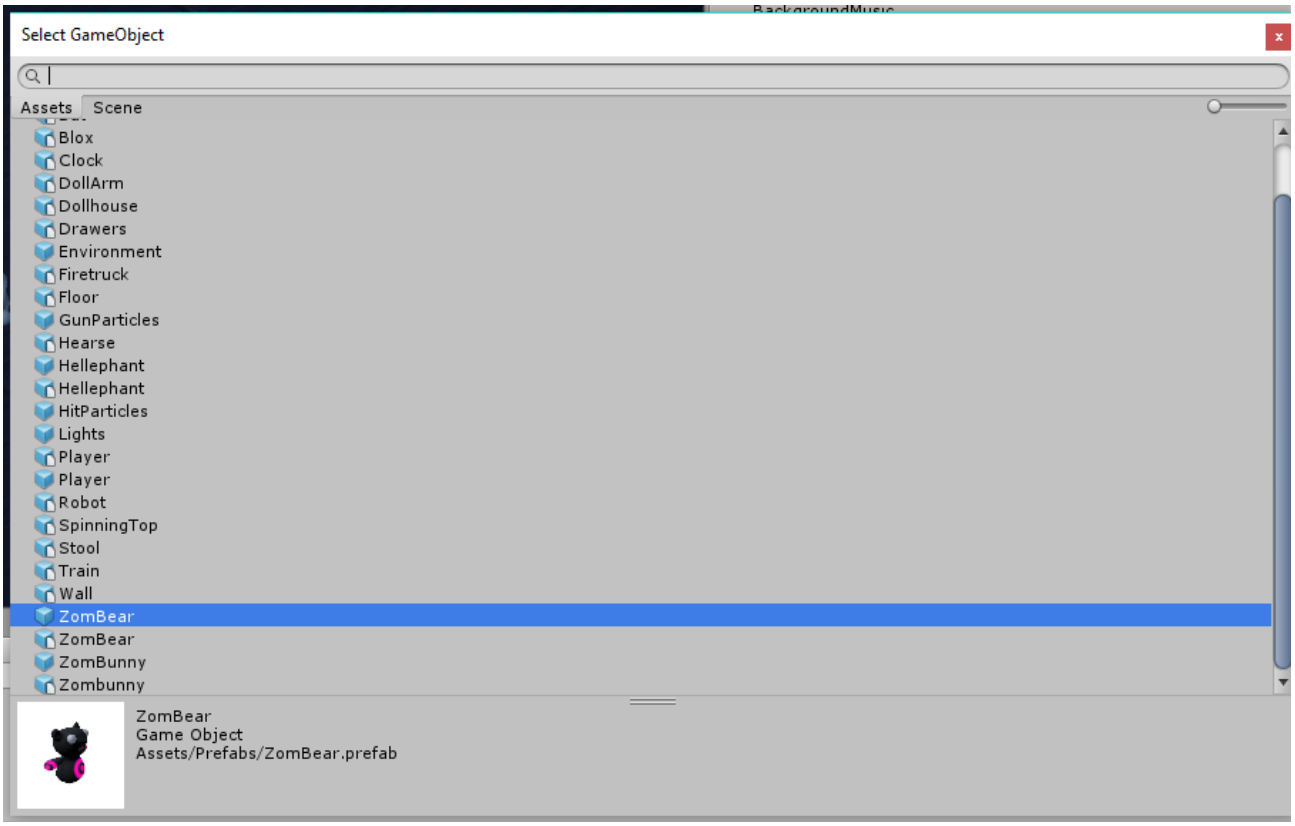
6. Set the enemy of the first Enemy Manager script to ZomBunny, spawn time = 3, spawn point size = 1, element 0 = ZomBunnySpawnPoint

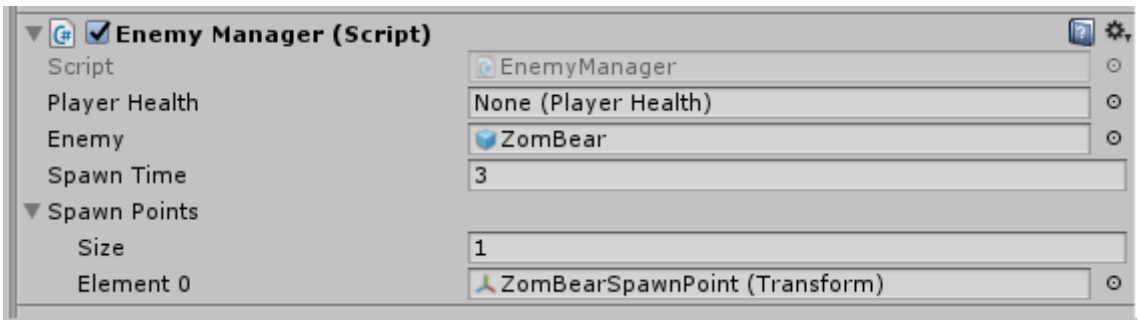




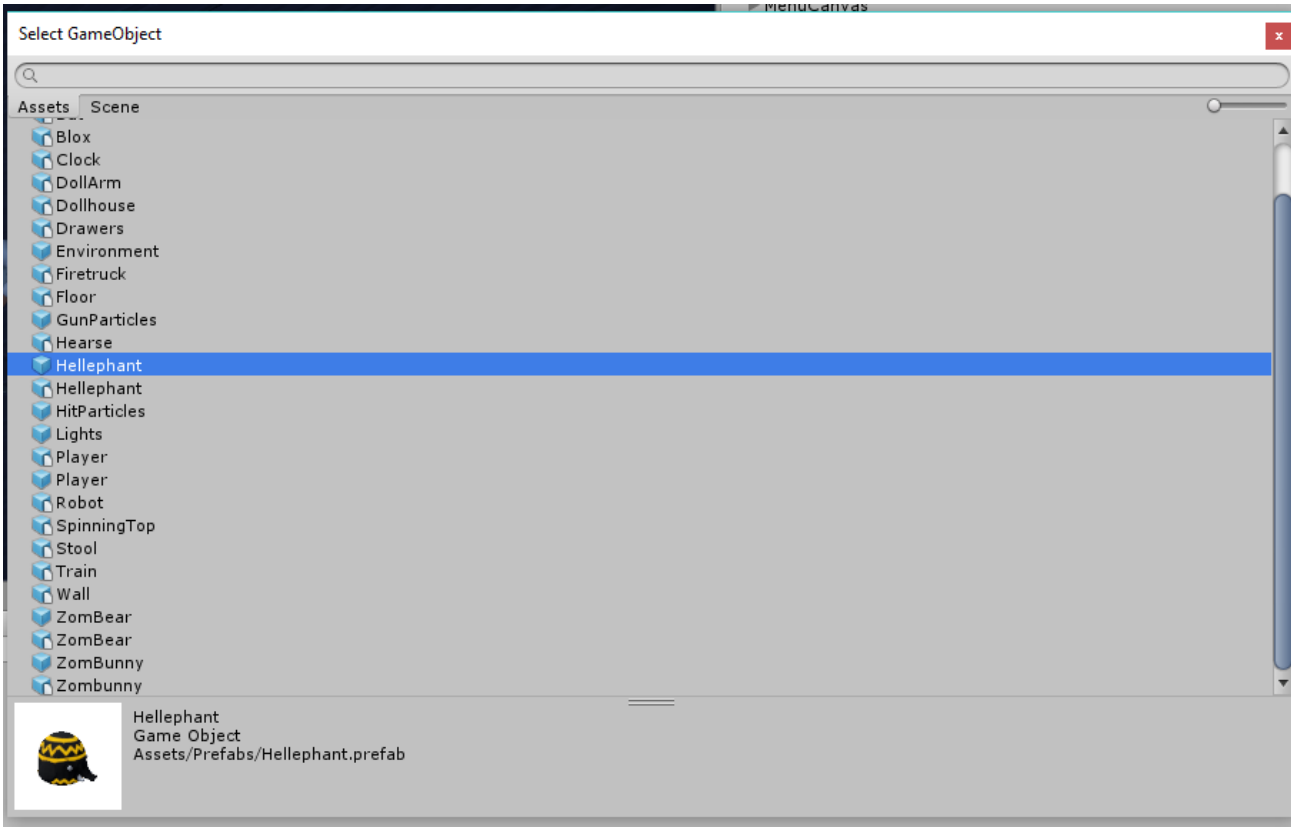


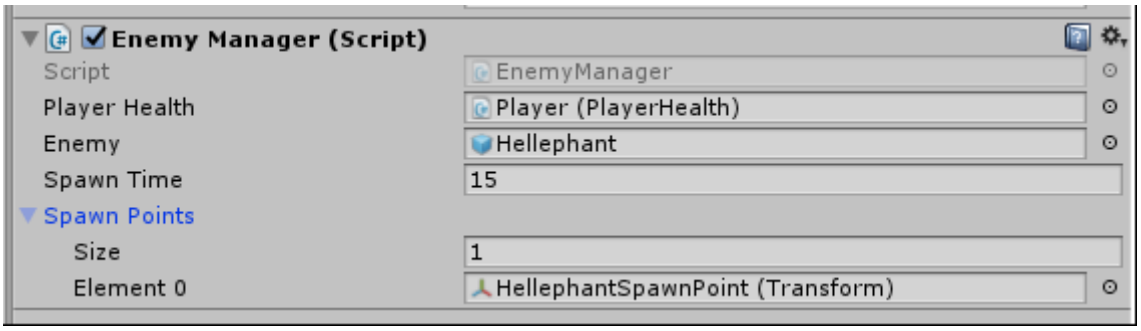
7. Set the enemy of the second Enemy Manager script to ZomBear, spawn time = 3, spawn point size = 1, element 0 = ZomBearSpawnPoint



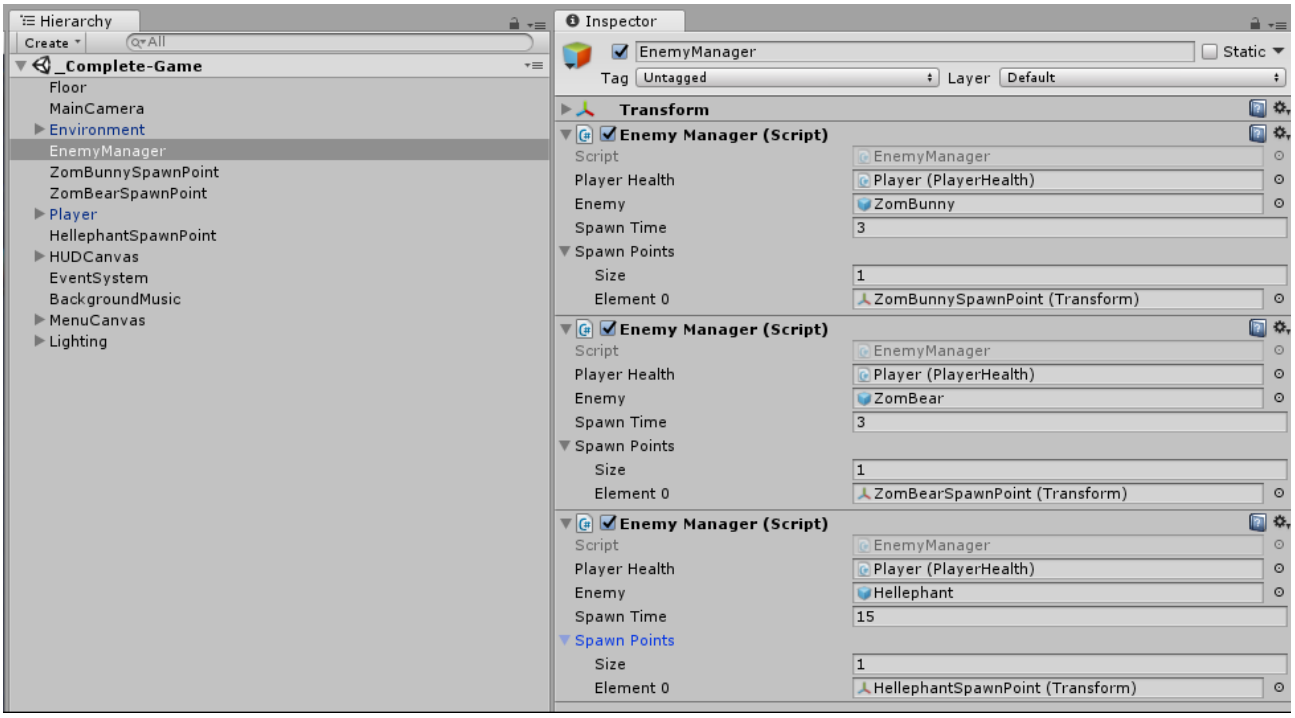


8. Set the enemy of the second Enemy Manager script to Hellephant, spawn time = 15, spawn point size = 1, element 0 = HellephantSpawnPoint





9. Completed Enemy Manager Game Object



10.

Unity Survival Shooter Game - Lesson 4 - Player Health

1. Open PlayerHealth script in Visual Studio
2. Uncomment TakeDamage method

```
1 reference | 0 changes | 0 authors, 0 changes
58  public void TakeDamage (int amount)
59  {
60      // Set the damaged flag so the screen will flash.
61      damaged = true;
62
63      // Reduce the current health by the damage amount.
64      currentHealth -= amount;
65
66      // Set the health bar's value to the current health.
67      healthSlider.value = currentHealth;
68
69      // Play the hurt sound effect.
70      playerAudio.Play ();
71
72      // If the player has lost all it's health and the death flag hasn't been set yet...
73      if(currentHealth <= 0 && !isDead)
74      {
75          // ... it should die.
76          Death ();
77      }
78  }
```

3. Uncomment Death method

```
0 references | 0 changes | 0 authors, 0 changes
81  void Death ()
82  {
83      // Set the death flag so this function won't be called again.
84      isDead = true;
85
86      // Turn off any remaining shooting effects.
87      playerShooting.DisableEffects ();
88
89      // Tell the animator that the player is dead.
90      anim.SetTrigger ("Die");
91
92      // Set the audiosource to play the death clip and play it (this will stop the hurt sound from playing).
93      playerAudio.clip = deathClip;
94      playerAudio.Play ();
95
96      // Turn off the movement and shooting scripts.
97      playerMovement.enabled = false;
98      playerShooting.enabled = false;
99  }
```

4. Open EnemyAttack script in Visual Studio and uncomment player health take damage method call, line 80.

```
1 reference | 0 changes | 0 authors, 0 changes
71  void Attack ()
72  {
73      // Reset the timer.
74      timer = 0f;
75
76      // If the player has health to lose...
77      if(playerHealth.currentHealth > 0)
78      {
79          // ... damage the player.
80          playerHealth.TakeDamage (attackDamage);
81      }
82  }
```

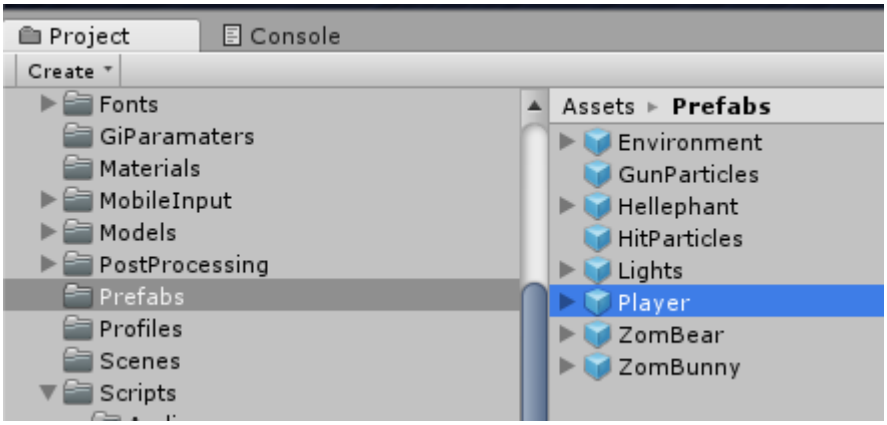
5. Open EnemyManager script and uncomment player health greater than 0 if condition.

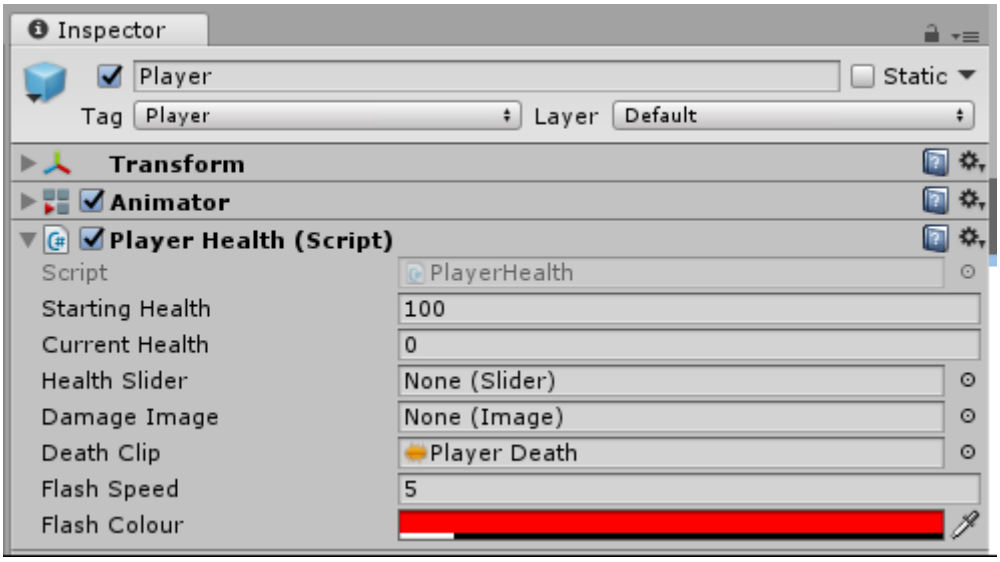
```
0 references | 0 changes | 0 authors, 0 changes
18  void Spawn ()
19  {
20      // If the player has no health left...
21      if(playerHealth.currentHealth <= 0f)
22      {
23          // ... exit the function.
24          return;
25      }
26
27      // Find a random index between zero and one less than the number of spawn points.
28      int spawnPointIndex = Random.Range (0, spawnPoints.Length);
29
30      // Create an instance of the enemy prefab at the randomly selected spawn point's position and rotation.
31      Instantiate (enemy, spawnPoints[spawnPointIndex].position, spawnPoints[spawnPointIndex].rotation);
32  }
```

6. Open EnemyMovement script and uncomment update method if condition.

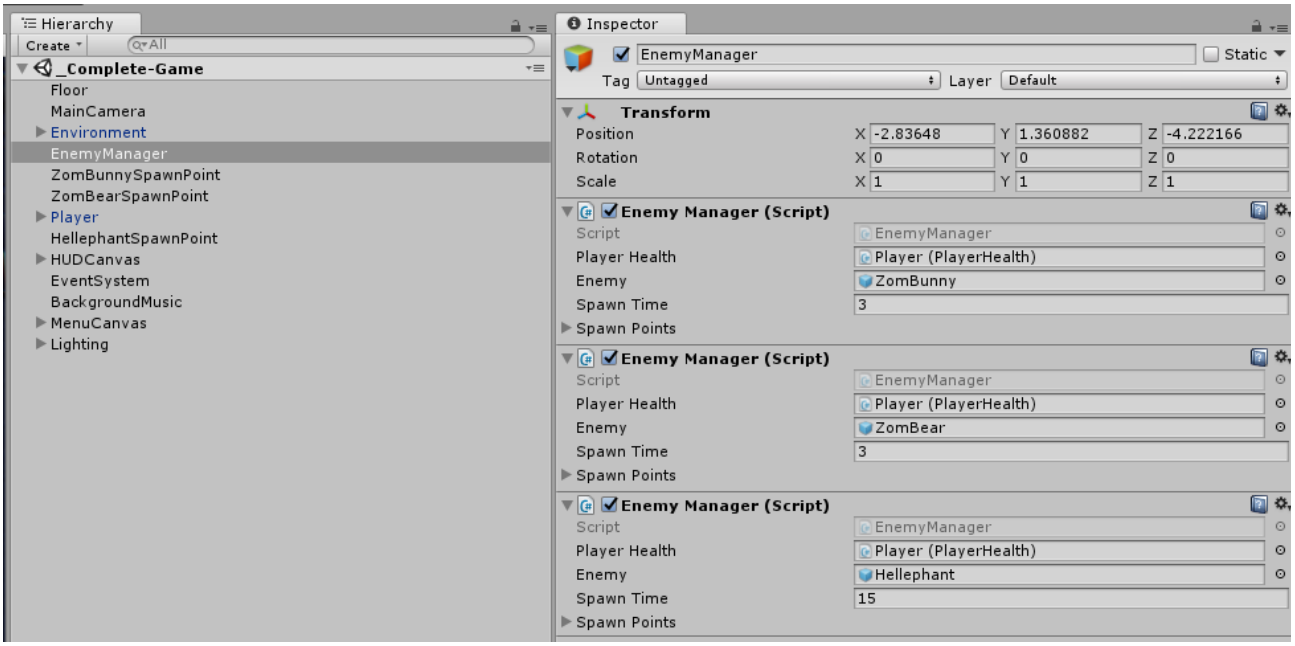
```
0 references | 0 changes | 0 authors, 0 changes
22  void Update ()
23  {
24      // If the enemy and the player have health left...
25      if(enemyHealth.currentHealth > 0 && playerHealth.currentHealth > 0)
26      {
27          // ... set the destination of the nav mesh agent to the player.
28          nav.SetDestination (player.position);
29      }
30      // Otherwise...
31      else
32      {
33          // ... disable the nav mesh agent.
34          nav.enabled = false;
35      }
36  }
```

7. Open Player Prefab and set add Player Health script component.





8. Update EnemyManager Player Health



9.

Unity Survival Shooter Game - Lesson 5 - Scoring Points & Game Over

Reference

Scoring points - https://unity3d.com/learn/tutorials/projects/survival-shooter-tutorial/scoring-points?playlist=17144

Lesson

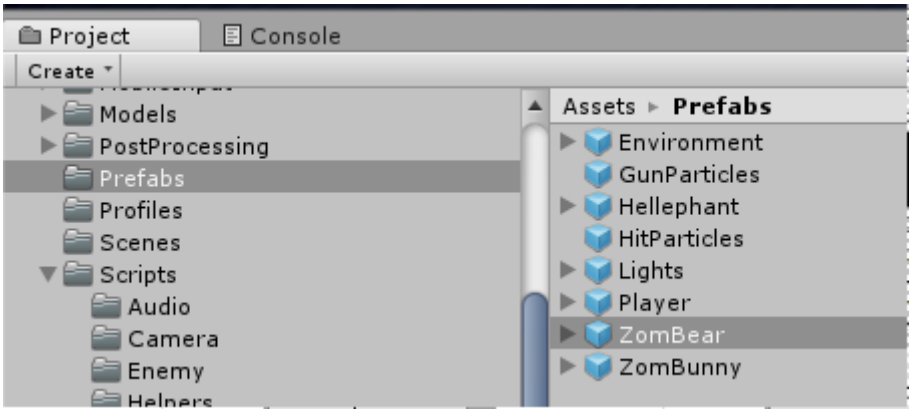
1. Open ScoreManager.cs file in Visual Studio
2. Uncomment text object text property assignment, line 26.

```
0 references | 0 changes | 0 authors, 0 changes
23      void Update ()
24      {
25          // Set the displayed text to be the word "Score" followed by the score value.
26          text.text = "Score: " + score;
27      }
28  }
```

3. Open EnemyHealth.cs file in Visual Studio
4. Uncomment ScoreManager object score property assignment, line 101

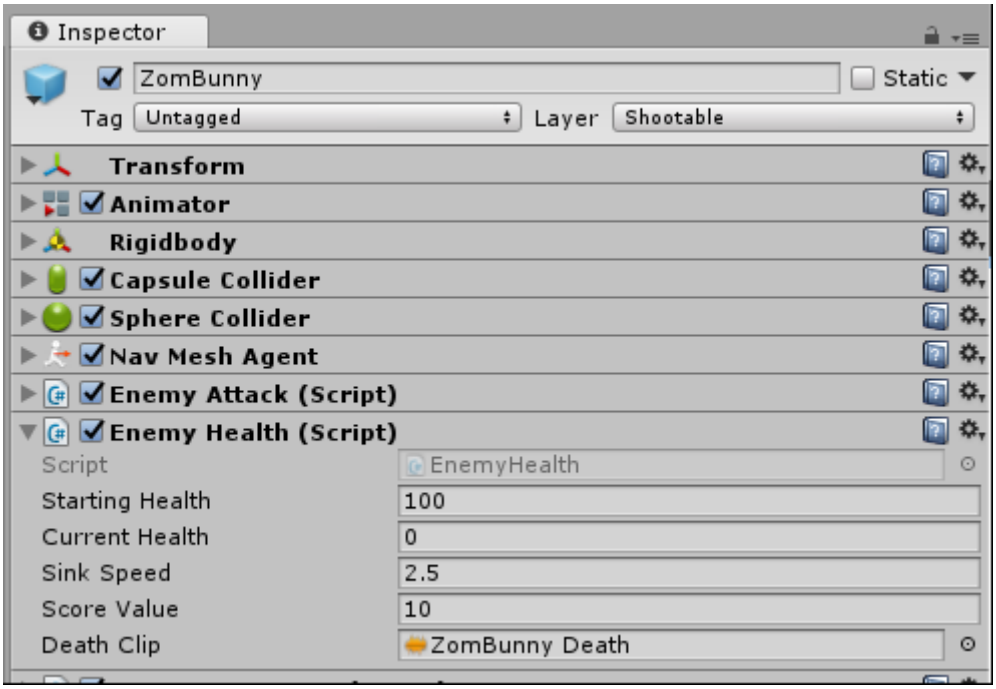
```
0 references | 0 changes | 0 authors, 0 changes
89      public void StartSinking ()
90      {
91          // Find and disable the Nav Mesh Agent.
92          GetComponent <UnityEngine.AI.NavMeshAgent> ().enabled = false;
93
94          // Find the rigidbody component and make it kinematic (since we use Translate to sink the enemy).
95          GetComponent <Rigidbody> ().isKinematic = true;
96
97          // The enemy should no sink.
98          isSinking = true;
99
100         // Increase the score by the enemy's score value.
101         ScoreManager.score += scoreValue;
102
103         // After 2 seconds destory the enemy.
104         Destroy (gameObject, 2f);
105     }
```

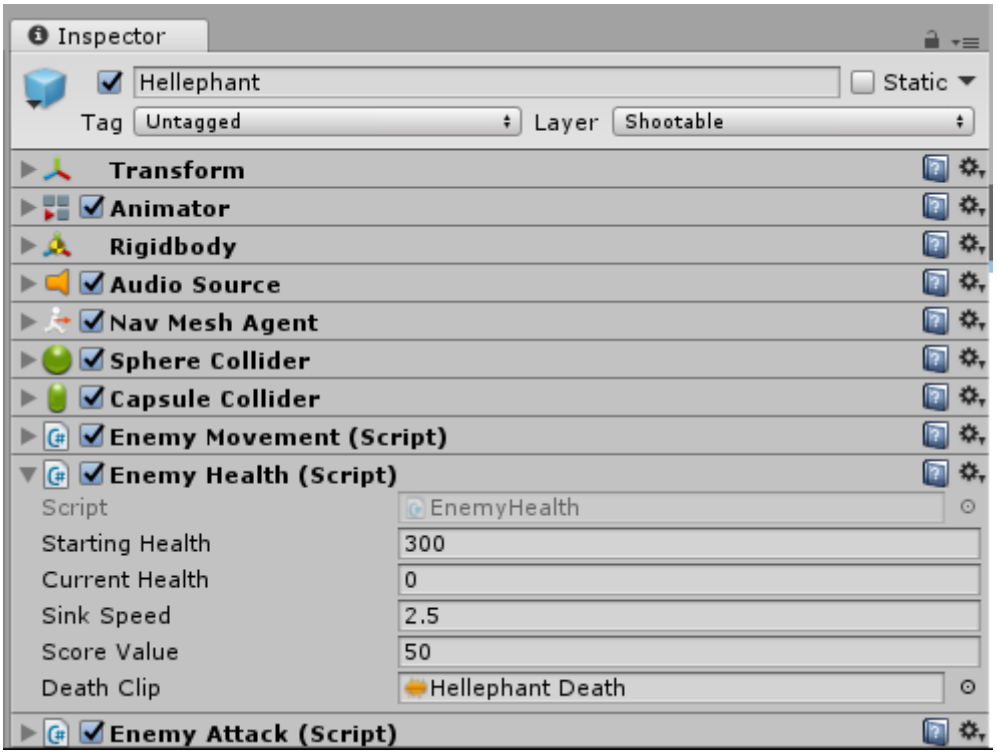
5. Set ZomBear Prefab Score Value
6. In Unity select ZomBear Prefab



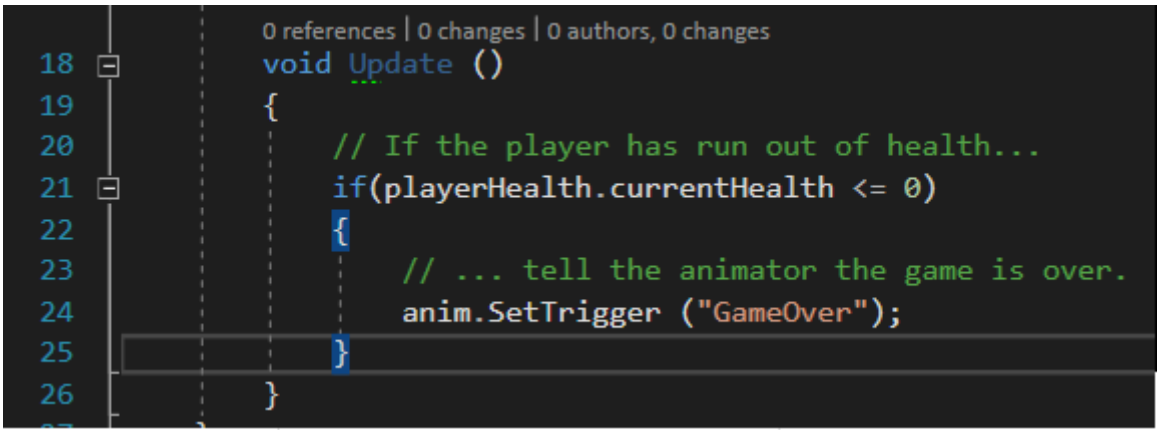
7. Set Score Value property

8. Repeat with ZomBunny and Hellephant





9. Open GameOverManager.cs file in Visual Studio
10. Uncomment player health if statement.



11.

Unity Tanks Game

Unity Tutorial - https://unity3d.com/learn/tutorials/projects/tanks-tutorial

Source Code - https://github.com/jasonweibel/HyperStream-Unity

- Unity Tanks Game - Lesson 1 - Add Tank Movement
- Unity Tanks Game - Lesson 2 - Add Shell Creation and Firing
- Unity Tanks Game - Lesson 3 - Add Tank Health
- Unity Tanks Game - Lesson 4 - Add Audio Mixing

Unity Tanks Game - Lesson 1 - Add Tank Movement

Update TankMovement.cs File

1. Open TankMovement.cs file in Visual Studio
2. Uncomment Move method

```
1 reference | 0 changes | 0 authors, 0 changes
119  private void Move ()
120  {
121      // Create a vector in the direction the tank is facing with a magnitude based on the input, speed and the time between frames.
122      Vector3 movement = transform.forward * m_MovementInputValue * m_Speed * Time.deltaTime;
123
124      // Apply this movement to the rigidbody's position.
125      m_Rigidbody.MovePosition(m_Rigidbody.position + movement);
126  }
```

3. Uncomment Turn method

```
1 reference | 0 changes | 0 authors, 0 changes
129  private void Turn ()
130  {
131      // Determine the number of degrees to be turned based on the input, speed and time between frames.
132      float turn = m_TurnInputValue * m_TurnSpeed * Time.deltaTime;
133
134      // Make this into a rotation in the y axis.
135      Quaternion turnRotation = Quaternion.Euler (0f, turn, 0f);
136
137      // Apply this rotation to the rigidbody's rotation.
138      m_Rigidbody.MoveRotation (m_Rigidbody.rotation * turnRotation);
139  }
```

4. Uncomment FixedUpdate method

```
0 references | 0 changes | 0 authors, 0 changes
111  private void FixedUpdate ()
112  {
113      // Adjust the rigidbodies position and orientation in FixedUpdate.
114      Move ();
115      Turn ();
116  }
```

5. Uncomment movement.enabled = false in disable control method, line 56.

```
1 reference | 0 changes | 0 authors, 0 changes
54  public void DisableControl ()
55  {
56      m_Movement.enabled = false;
57      //m_Shooting.enabled = false;
58
59      m_CanvasGameObject.SetActive (false);
60  }
61
```

Update TankManager.cs File

1. Open TankManager script in Visual Studio
2. uncomment setting of player number for movement, line 35.

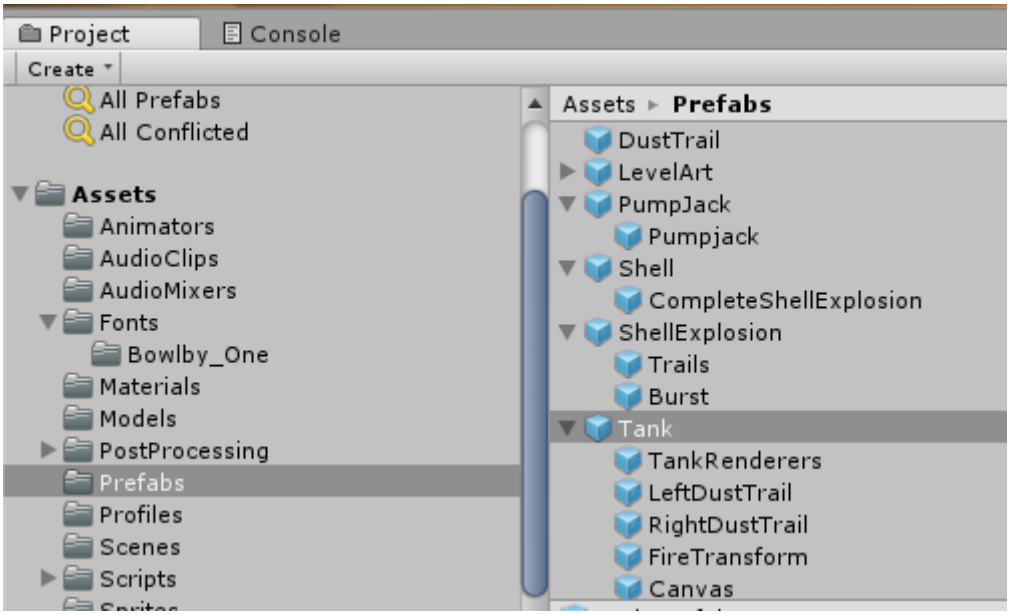
```
34      // Set the player numbers to be consistent across the scripts.
35      m_Movement.m_PlayerNumber = m_PlayerNumber;
36      //m_Shooting.m_PlayerNumber = m_PlayerNumber;
```

3. uncomment movement.enbled in enable controller method, line 66.

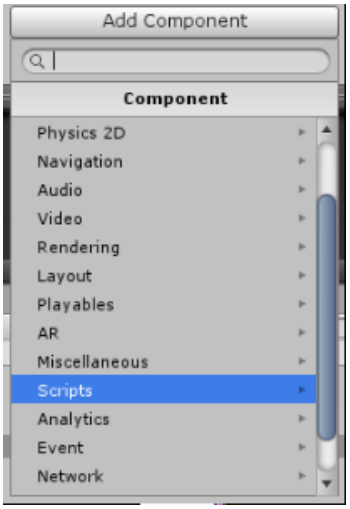
```
64      public void EnableControl ()
65      {
66          m_Movement.enabled = true;
67          //m_Shooting.enabled = true;
68
69          m_CanvasGameObject.SetActive (true);
70      }
```

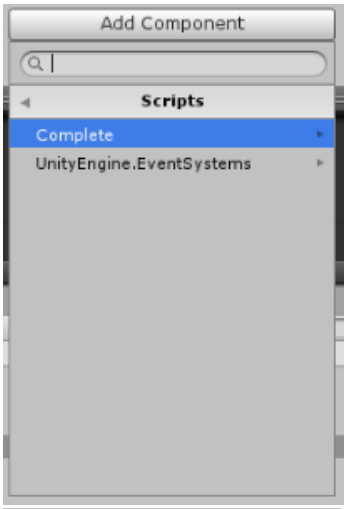
Add Tank Movement Script Component

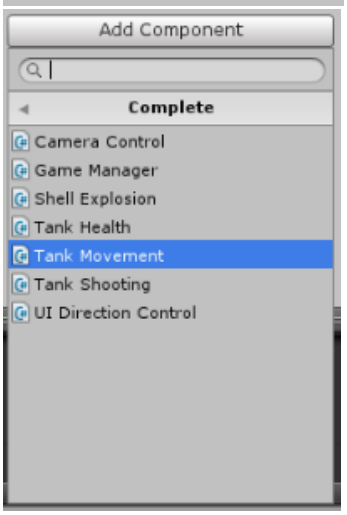
1. Select Tank Prefab



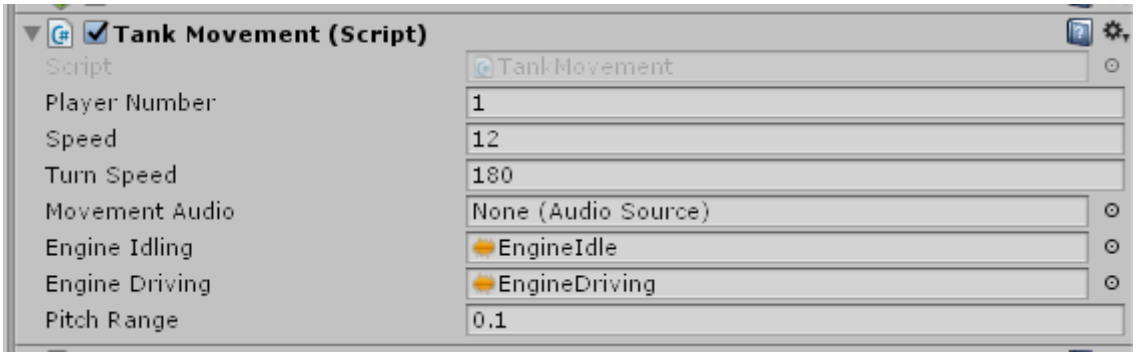
2. Add TankMovement Script Component







3. Set Properties



Unity Tanks Game - Lesson 2 - Add Shell Creation and Firing

Update TankShooting.cs File

1. Open TankShooting.cs file in Visual Studio
2. Uncomment public and private properties.

```
2 references | 0 changes | 0 authors, 0 changes
public class TankShooting : MonoBehaviour
{
    public int m_PlayerNumber = 1;              // Used to identify the different players.
    public Rigidbody m_Shell;                   // Prefab of the shell.
    public Transform m_FireTransform;           // A child of the tank where the shells are spawned.
    public Slider m_AimSlider;                  // A child of the tank that displays the current launch force.
    public AudioSource m_ShootingAudio;         // Reference to the audio source used to play the shooting audio. NB: different to the movement audio source.
    public AudioClip m_ChargingClip;            // Audio that plays when each shot is charging up.
    public AudioClip m_FireClip;                // Audio that plays when each shot is fired.
    public float m_MinLaunchForce = 15f;        // The force given to the shell if the fire button is not held.
    public float m_MaxLaunchForce = 30f;        // The force given to the shell if the fire button is held for the max charge time.
    public float m_MaxChargeTime = 0.75f;       // How long the shell can charge for before it is fired at max force.


    private string m_FireButton;                // The input axis that is used for launching shells.
    private float m_CurrentLaunchForce;         // The force that will be given to the shell when the fire button is released.
    private float m_ChargeSpeed;                // How fast the launch force increases, based on the max charge time.
    private bool m_Fired;                       // Whether or not the shell has been launched with this button press.
```

3. Uncomment OnEnable method

```
0 references | 0 changes | 0 authors, 0 changes
private void OnEnable()
{
    // When the tank is turned on, reset the launch force and the UI
    m_CurrentLaunchForce = m_MinLaunchForce;
    m_AimSlider.value = m_MinLaunchForce;
}
```

4. Uncomment Start method

```
0 references | 0 changes | 0 authors, 0 changes
private void Start ()
{
    // The fire axis is based on the player number.
    m_FireButton = "Fire" + m_PlayerNumber;

    // The rate that the launch force charges up is the range of possible forces by the max charge time.
    m_ChargeSpeed = (m_MaxLaunchForce - m_MinLaunchForce) / m_MaxChargeTime;
}
```

5. Uncomment Fire method

```
2 references | 0 changes | 0 authors, 0 changes
private void Fire ()
{
    // Set the fired flag so only Fire is only called once.
    m_Fired = true;

    // Create an instance of the shell and store a reference to it's rigidbody.
    Rigidbody shellInstance =
        Instantiate (m_Shell, m_FireTransform.position, m_FireTransform.rotation) as Rigidbody;

    // Set the shell's velocity to the launch force in the fire position's forward direction.
    shellInstance.velocity = m_CurrentLaunchForce * m_FireTransform.forward;

    //// Change the clip to the firing clip and play it.
    //m_ShootingAudio.clip = m_FireClip;
    //m_ShootingAudio.Play ();

    // Reset the launch force.  This is a precaution in case of missing button events.
    m_CurrentLaunchForce = m_MinLaunchForce;
}
```

6. Uncomment Update method

```
0 references | 0 changes | 0 authors, 0 changes
44  private void Update ()
45  {
46      // The slider should have a default value of the minimum launch force.
47      m_AimSlider.value = m_MinLaunchForce;
48
49      // If the max force has been exceeded and the shell hasn't yet been launched...
50      if (m_CurrentLaunchForce >= m_MaxLaunchForce && !m_Fired)
51      {
52          // ... use the max force and launch the shell.
53          m_CurrentLaunchForce = m_MaxLaunchForce;
54          Fire ();
55      }
56      // Otherwise, if the fire button has just started being pressed...
57      else if (Input.GetButtonDown (m_FireButton))
58      {
59          // ... reset the fired flag and reset the launch force.
60          m_Fired = false;
61          m_CurrentLaunchForce = m_MinLaunchForce;
62
63          //// Change the clip to the charging clip and start it playing.
64          //m_ShootingAudio.clip = m_ChargingClip;
65          //m_ShootingAudio.Play ();
66      }
67      // Otherwise, if the fire button is being held and the shell hasn't been launched yet...
68      else if (Input.GetButton (m_FireButton) && !m_Fired)
69      {
70          // Increment the launch force and update the slider.
71          m_CurrentLaunchForce += m_ChargeSpeed * Time.deltaTime;
72
73          m_AimSlider.value = m_CurrentLaunchForce;
74      }
75      // Otherwise, if the fire button is released and the shell hasn't been launched yet...
76      else if (Input.GetButtonUp (m_FireButton) && !m_Fired)
77      {
78          // ... launch the shell.
79          Fire ();
80      }
81  }
```

Update TankManager.cs File

1. Open TankManager.cs file in Visual Studio
2. Uncomment line 36.

```
34          // Set the player numbers to be consistent across the scripts.
35          m_Movement.m_PlayerNumber = m_PlayerNumber;
36          m_Shooting.m_PlayerNumber = m_PlayerNumber;
```

3. Uncomment line 57

```
53      // Used during the phases of the game where the player shouldn't be able to control their tank.
        1 reference | 0 changes | 0 authors, 0 changes
54      public void DisableControl ()
55      {
56          m_Movement.enabled = false;
57          m_Shooting.enabled = false;
```

4. Uncomment line 67

```
63          // Used during the phases of the game where the player should be able to control their tank.
            1 reference | 0 changes | 0 authors, 0 changes
64          public void EnableControl ()
65          {
66              m_Movement.enabled = true;
67              m_Shooting.enabled = true;
68
69              m_CanvasGameObject.SetActive (true);
70          }
```

Update ShellExplosion.cs File

1. Open ShellExplosion.cs file in Visual Studio
2. Uncomment public properties

```
            0 references | 0 changes | 0 authors, 0 changes
5           public class ShellExplosion : MonoBehaviour
6           {
7               public LayerMask m_TankMask;                        // Used to filter what the explosion affects, this should be set to "Players".
8               public ParticleSystem m_ExplosionParticles;         // Reference to the particles that will play on explosion.
9               public AudioSource m_ExplosionAudio;                // Reference to the audio that will play on explosion.
10              public float m_MaxDamage = 100f;                    // The amount of damage done if the explosion is centred on a tank.
11              public float m_ExplosionForce = 1000f;              // The amount of force added to a tank at the centre of the explosion.
12              public float m_MaxLifeTime = 2f;                    // The time in seconds before the shell is removed.
13              public float m_ExplosionRadius = 5f;                // The maximum distance away from the explosion tanks can be and are still affected.
```

3. Uncomment Start method

```
                0 references | 0 changes | 0 authors, 0 changes
16              private void Start ()
17              {
18                  // If it isn't destroyed by then, destroy the shell after it's lifetime.
19                  Destroy (gameObject, m_MaxLifeTime);
20              }
21
```

4. Uncomment CalculateDamage method

```
                1 reference | 0 changes | 0 authors, 0 changes
73              private float CalculateDamage (Vector3 targetPosition)
74              {
75                  // Create a vector from the shell to the target.
76                  Vector3 explosionToTarget = targetPosition - transform.position;
77
78                  // Calculate the distance from the shell to the target.
79                  float explosionDistance = explosionToTarget.magnitude;
80
81                  // Calculate the proportion of the maximum distance (the explosionRadius) the target is away.
82                  float relativeDistance = (m_ExplosionRadius - explosionDistance) / m_ExplosionRadius;
83
84                  // Calculate damage as this proportion of the maximum possible damage.
85                  float damage = relativeDistance * m_MaxDamage;
86
87                  // Make sure that the minimum damage is always 0.
88                  damage = Mathf.Max (0f, damage);
89
90                  return damage;
91              }
```

5. Uncomment OnTriggerEnter method

```
0 references | 0 changes | 0 authors, 0 changes
23  private void OnTriggerEnter (Collider other)
24  {
25      // Collect all the colliders in a sphere from the shell's current position to a radius of the explosion radius.
26      Collider[] colliders = Physics.OverlapSphere (transform.position, m_ExplosionRadius, m_TankMask);
27
28      // Go through all the colliders...
29      for (int i = 0; i < colliders.Length; i++)
30      {
31          // ... and find their rigidbody.
32          Rigidbody targetRigidbody = colliders[i].GetComponent<Rigidbody> ();
33
34          // If they don't have a rigidbody, go on to the next collider.
35          if (!targetRigidbody)
36              continue;
37
38          // Add an explosion force.
39          targetRigidbody.AddExplosionForce (m_ExplosionForce, transform.position, m_ExplosionRadius);
40
41          // Find the TankHealth script associated with the rigidbody.
42          TankHealth targetHealth = targetRigidbody.GetComponent<TankHealth> ();
43
44          // If there is no TankHealth script attached to the gameobject, go on to the next collider.
45          if (!targetHealth)
46              continue;
47
48          // Calculate the amount of damage the target should take based on it's distance from the shell.
49          float damage = CalculateDamage (targetRigidbody.position);
50
51          // Deal this damage to the tank.
52          //targetHealth.TakeDamage (damage);
53      }
54
55      // Unparent the particles from the shell.
56      m_ExplosionParticles.transform.parent = null;
57
58      // Play the particle system.
59      m_ExplosionParticles.Play();
60
61      // Play the explosion sound effect.
62      //m_ExplosionAudio.Play();
63
64      // Once the particles have finished, destroy the gameobject they are on.
65      ParticleSystem.MainModule mainModule = m_ExplosionParticles.main;
66      Destroy (m_ExplosionParticles.gameObject, mainModule.duration);
67
68      // Destroy the shell.
69      Destroy (gameObject);
70  }
```

Update TankShooting.cs File

1. Open TankShooting.cs File in Visual Studio
2. In Fire method uncomment lines 97 and 98.

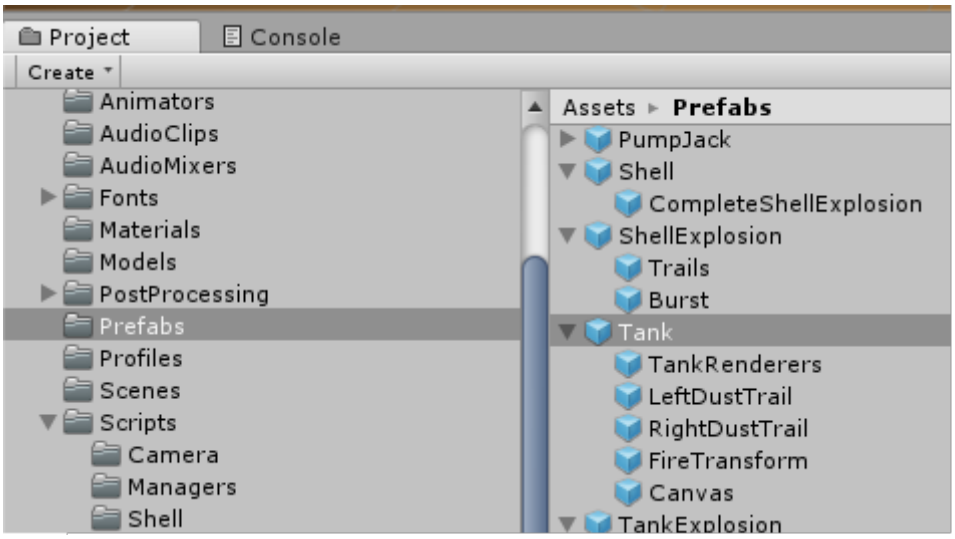
```
95
96          //// Change the clip to the firing clip and play it.
97          //m_ShootingAudio.clip = m_FireClip;
98          //m_ShootingAudio.Play ();
```

3. In Update method uncomment lines 64 and 65.

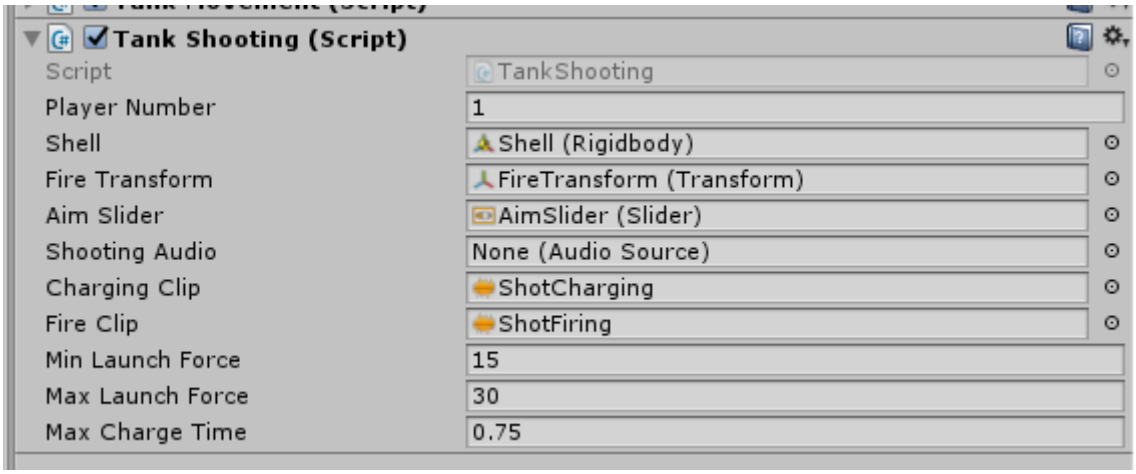
```
62
63          //// Change the clip to the charging clip and start it playing.
64          //m_ShootingAudio.clip = m_ChargingClip;
65          //m_ShootingAudio.Play ();
66      }
```

Add Tank Shooting Script to Tank Prefab

1. Select Tank Prefab

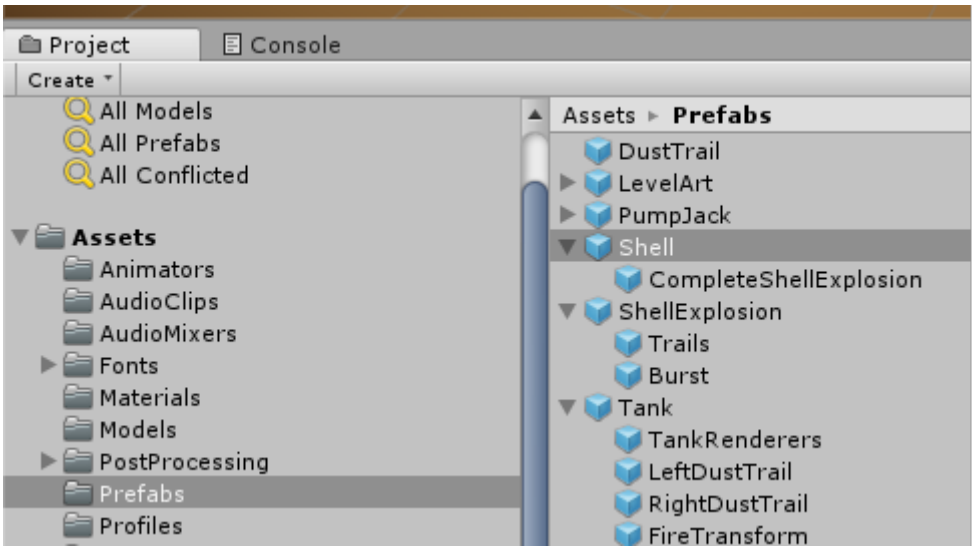


2. Add Script TankShooting
3. Set Script Properties
   Todo - figure out how AimSlider gets assigned.

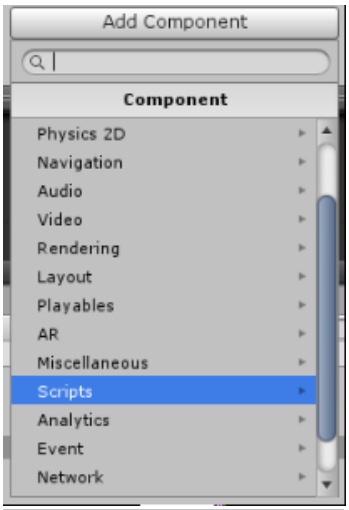


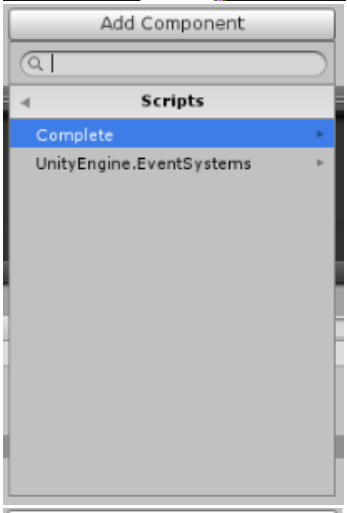
Add Shell Explosion Script to Shell Prefab

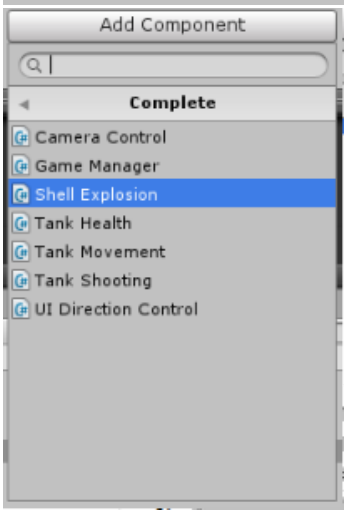
1. Select Shell Prefab



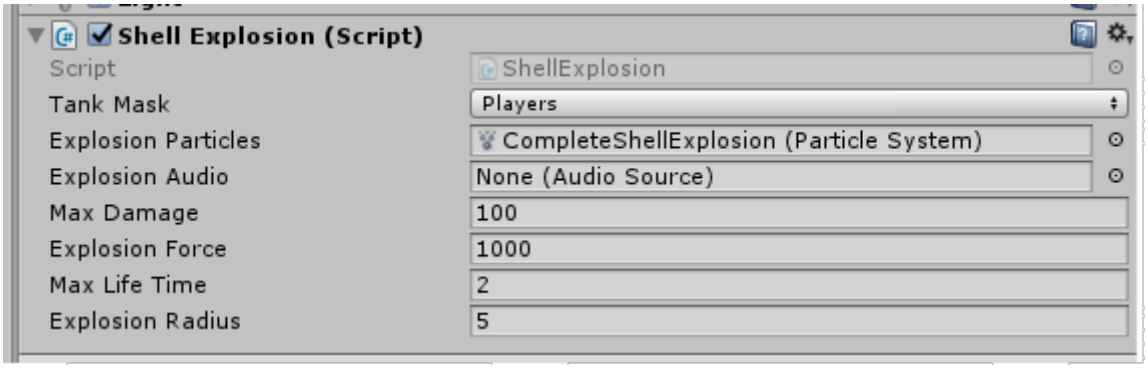
2. Add Shell Explosion Script Component







3. Set Properties
   Drag the CompleteShellExplosion prefab onto the Explosion Particles property.



4.

Unity Tanks Game - Lesson 3 - Add Tank Health

Update TankHealth Script

1. Open TankHealth.cs script in Visual Studio.
2. Uncomment public and private properties.

```
2 references | 0 changes | 0 authors, 0 changes
public class TankHealth : MonoBehaviour
{
    public float m_StartingHealth = 100f;               // The amount of health each tank starts with.
    public Slider m_Slider;                             // The slider to represent how much health the tank currently has.
    public Image m_FillImage;                           // The image component of the slider.
    public Color m_FullHealthColor = Color.green;       // The color the health bar will be when on full health.
    public Color m_ZeroHealthColor = Color.red;         // The color the health bar will be when on no health.
    public GameObject m_ExplosionPrefab;                // A prefab that will be instantiated in Awake, then used whenever the tank dies.


    private AudioSource m_ExplosionAudio;               // The audio source to play when the tank explodes.
    private ParticleSystem m_ExplosionParticles;        // The particle system the will play when the tank is destroyed.
    private float m_CurrentHealth;                      // How much health the tank currently has.
    private bool m_Dead;                                // Has the tank been reduced beyond zero health yet?
```

3. Uncomment Awake Method

```
0 references | 0 changes | 0 authors, 0 changes
private void Awake ()
{
    // Instantiate the explosion prefab and get a reference to the particle system on it.
    m_ExplosionParticles = Instantiate (m_ExplosionPrefab).GetComponent<ParticleSystem> ();

    // Get a reference to the audio source on the instantiated prefab.
    m_ExplosionAudio = m_ExplosionParticles.GetComponent<AudioSource> ();

    // Disable the prefab so it can be activated when it's required.
    m_ExplosionParticles.gameObject.SetActive (false);
}
```

4. Uncomment SetHealthUI method

```
2 references | 0 changes | 0 authors, 0 changes
private void SetHealthUI ()
{
    // Set the slider's value appropriately.
    m_Slider.value = m_CurrentHealth;

    // Interpolate the color of the bar between the choosen colours based on the current percentage of the starting health.
    m_FillImage.color = Color.Lerp (m_ZeroHealthColor, m_FullHealthColor, m_CurrentHealth / m_StartingHealth);
}
```

5. Uncomment OnDeath method

```
1 reference | 0 changes | 0 authors, 0 changes
private void OnDeath ()
{
    // Set the flag so that this function is only called once.
    m_Dead = true;

    // Move the instantiated explosion prefab to the tank's position and turn it on.
    m_ExplosionParticles.transform.position = transform.position;
    m_ExplosionParticles.gameObject.SetActive (true);

    // Play the particle system of the tank exploding.
    m_ExplosionParticles.Play ();

    // Play the tank explosion sound effect.
    m_ExplosionAudio.Play();

    // Turn the tank off.
    gameObject.SetActive (false);
}
```

6. Uncomment OnEnable method

```
0 references | 0 changes | 0 authors, 0 changes
private void OnEnable()
{
    // When the tank is enabled, reset the tank's health and whether or not it's dead.
    m_CurrentHealth = m_StartingHealth;
    m_Dead = false;

    // Update the health slider's value and color.
    SetHealthUI();
}
```

7. Uncomment Take Damage method

```
1 reference | 0 changes | 0 authors, 0 changes
public void TakeDamage (float amount)
{
    // Reduce current health by the amount of damage done.
    m_CurrentHealth -= amount;

    // Change the UI elements appropriately.
    SetHealthUI ();

    // If the current health is at or below zero and it has not yet been registered, call OnDeath.
    if (m_CurrentHealth <= 0f && !m_Dead)
    {
        OnDeath ();
    }
}
```

Update ShellExplosion.cs Script

1. Open ShellExplosion.cs file in Visual Studio

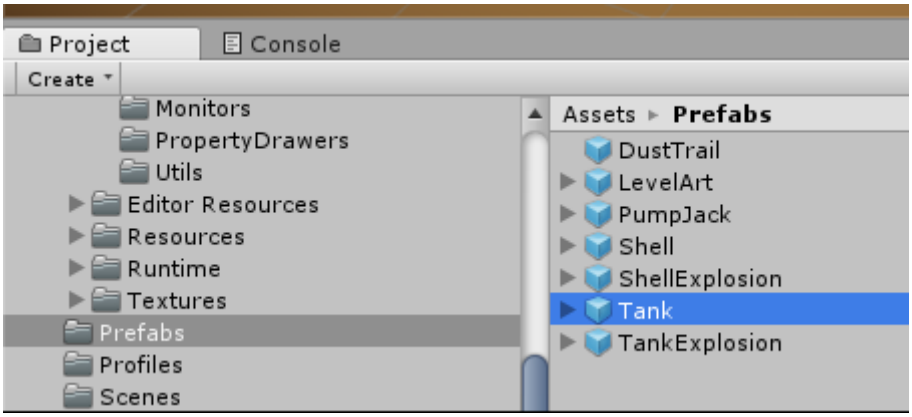
2. Uncomment line 52.

```
51              // Deal this damage to the tank.
52              targetHealth.TakeDamage (damage);
53          }
```

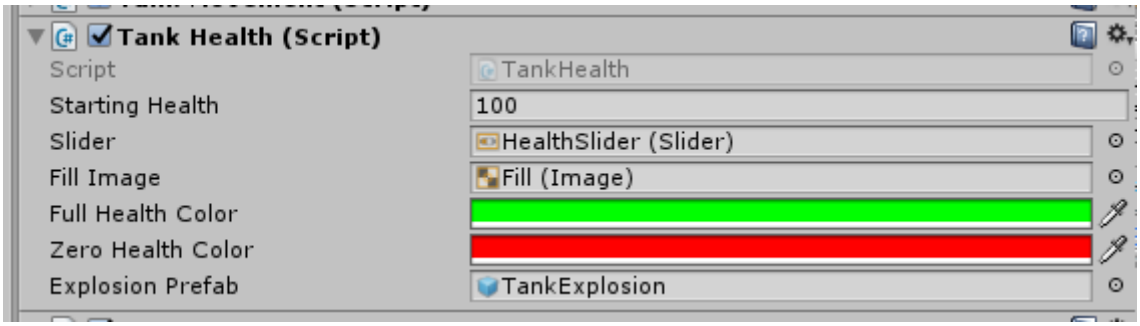
3.

Add Tank Health Script to Tank Prefab

1. Select Tanks Prefab



2. Add a Scripts --> Complete --> Tank Health
3. Set Tank Health Script properties



Unity Tanks Game - Lesson 4 - Add Audio Mixing

Enable Engine Audio Code

1. Open TankMovement.cs file in Visual Studio
2. Uncomment EngineAudio method on linew 83 - 108.

```
1 reference | 0 changes | 0 authors, 0 changes
83  private void EngineAudio ()
84  {
85      // If there is no input (the tank is stationary)...
86      if (Mathf.Abs (m_MovementInputValue) < 0.1f && Mathf.Abs (m_TurnInputValue) < 0.1f)
87      {
88          // ... and if the audio source is currently playing the driving clip...
89          if (m_MovementAudio.clip == m_EngineDriving)
90          {
91              // ... change the clip to idling and play it.
92              m_MovementAudio.clip = m_EngineIdling;
93              m_MovementAudio.pitch = Random.Range (m_OriginalPitch - m_PitchRange, m_OriginalPitch + m_PitchRange);
94              m_MovementAudio.Play ();
95          }
96      }
97      else
98      {
99          // Otherwise if the tank is moving and if the idling clip is currently playing...
100         if (m_MovementAudio.clip == m_EngineIdling)
101         {
102             // ... change the clip to driving and play.
103             m_MovementAudio.clip = m_EngineDriving;
104             m_MovementAudio.pitch = Random.Range(m_OriginalPitch - m_PitchRange, m_OriginalPitch + m_PitchRange);
105             m_MovementAudio.Play();
106         }
107     }
108 }
```

3. Uncomment the call of the EngineAudio method on line 79.

```
0 references | 0 changes | 0 authors, 0 changes
73  private void Update ()
74  {
75      // Store the value of both input axes.
76      m_MovementInputValue = Input.GetAxis (m_MovementAxisName);
77      m_TurnInputValue = Input.GetAxis (m_TurnAxisName);
78
79      EngineAudio ();
80  }
81
```

4. Uncomment the setting of the original pitch variable.

```
0 references | 0 changes | 0 authors, 0 changes
62  private void Start ()
63  {
64      // The axes names are based on player number.
65      m_MovementAxisName = "Vertical" + m_PlayerNumber;
66      m_TurnAxisName = "Horizontal" + m_PlayerNumber;
67
68      // Store the original pitch of the audio source.
69      m_OriginalPitch = m_MovementAudio.pitch;
70  }
71
```

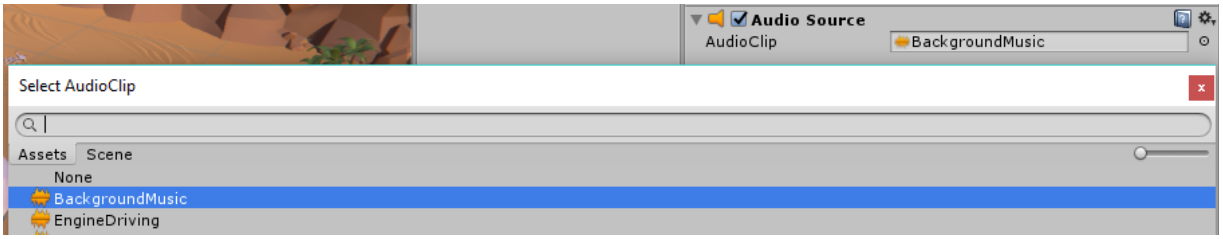
Enable Explosion Audio Code

1. Open ShellExplosion.cs file in Visual Studio
2. Uncomment line 62

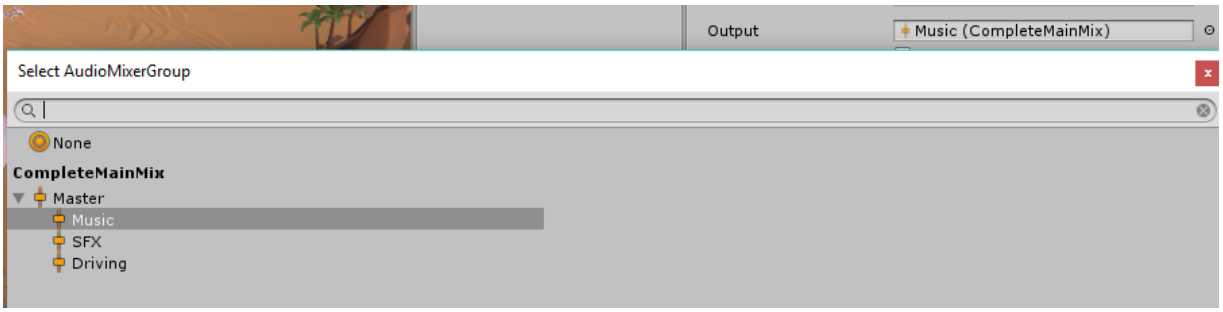
```
61      // Play the explosion sound effect.
62      m_ExplosionAudio.Play();
63
```

Add Background Music

1. Select GameManager
2. Add Audio --> Audio Source component
3. Change AudioClip to Background Music

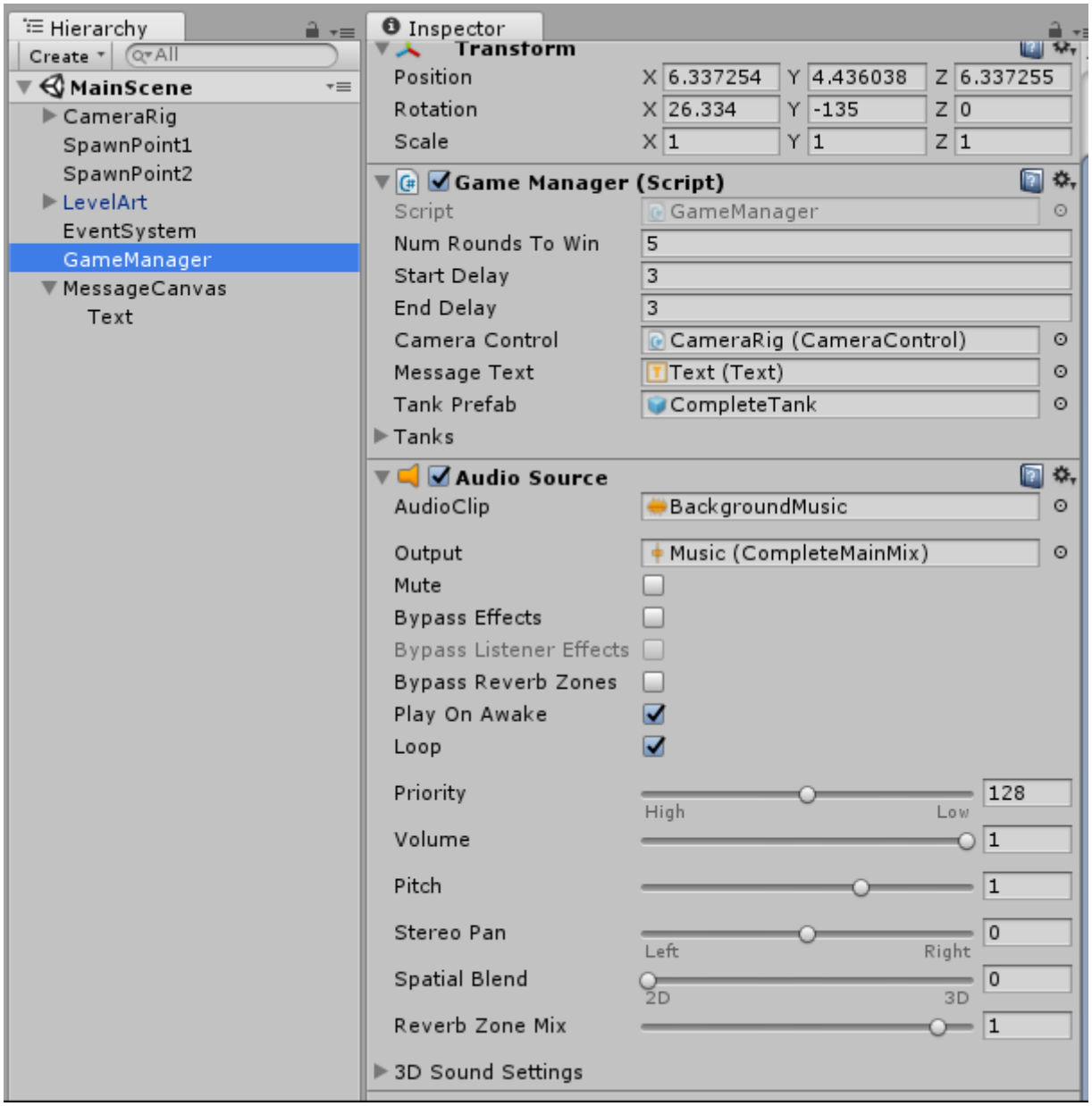


4. Change output to Music



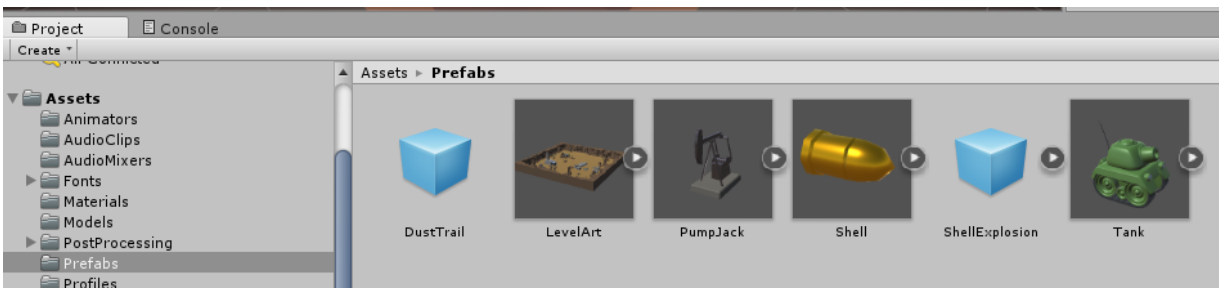
5. Check the Play On Awake checkbox
6. Check the Loop checkbox
7. Play game.
8. Have students experiment with changing priority, volume, pitch, and other settings.

Finished Background Music Audio Source Component

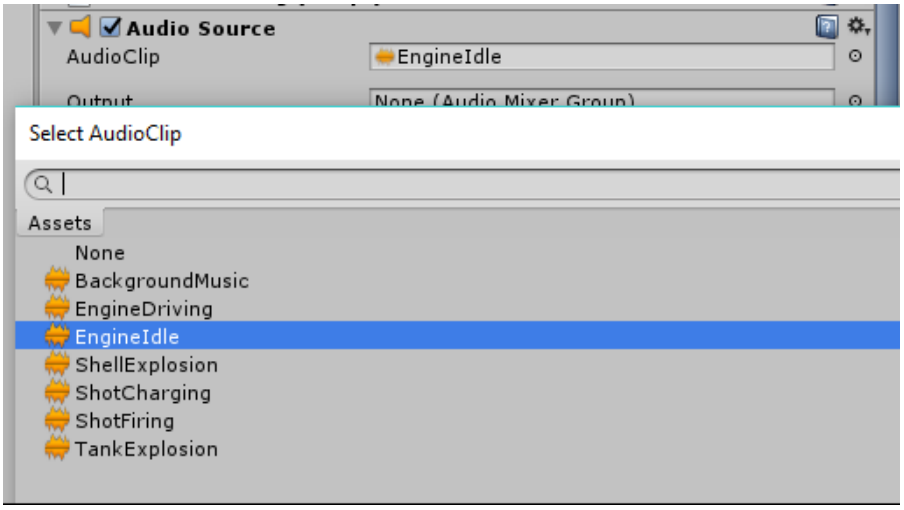


Add Tank Engine Audio

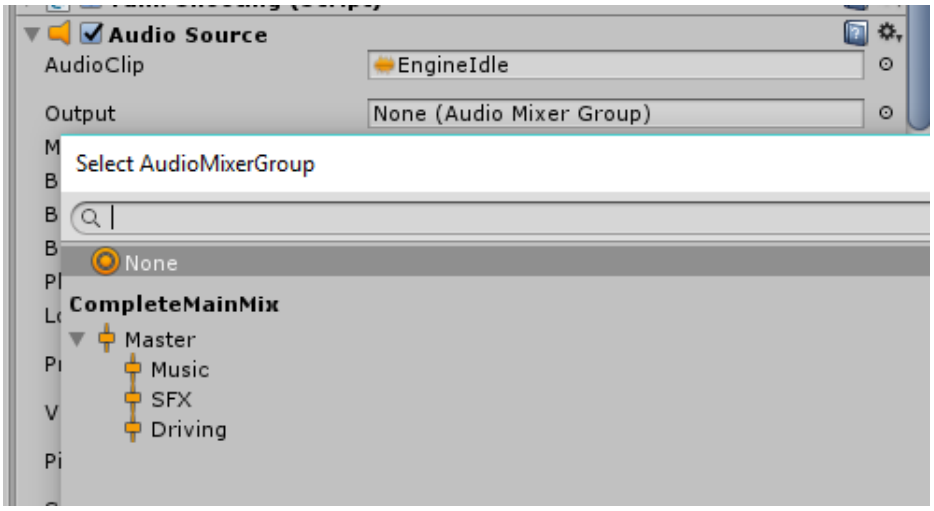
1. Select Tank Prefab



2. Add Audio Source Component
3. Set Audio Clip to EngineIdle

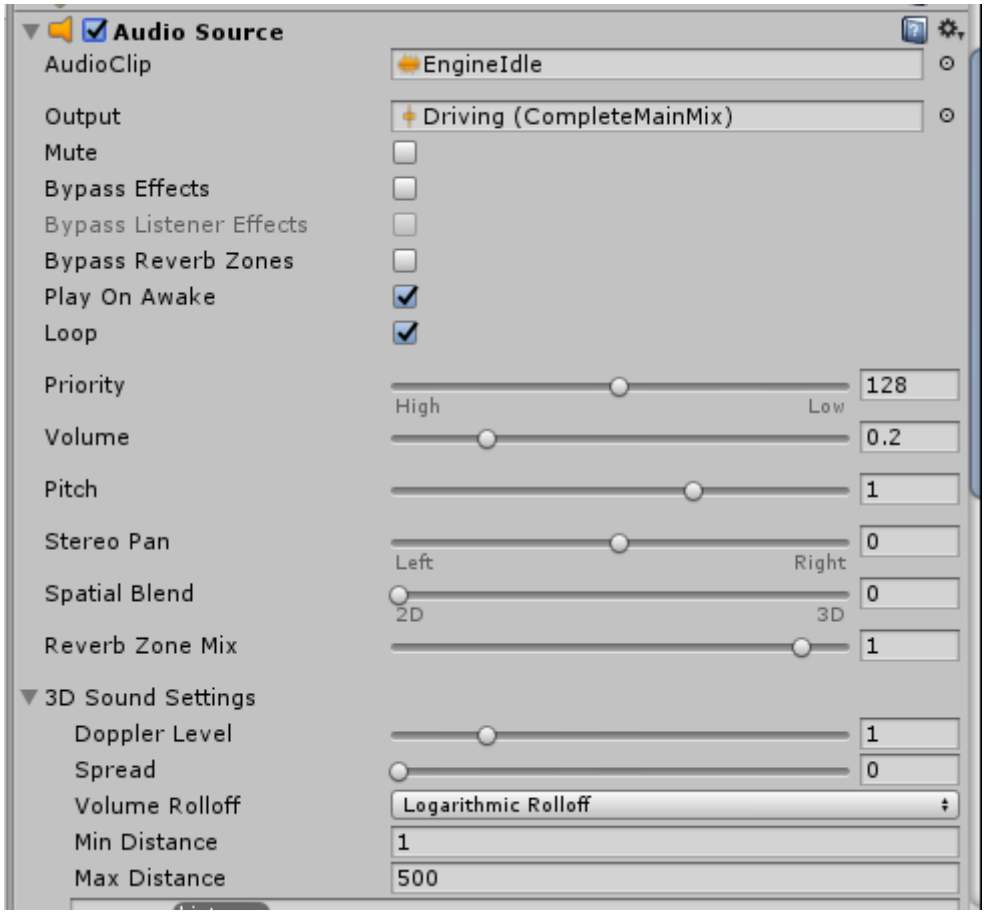


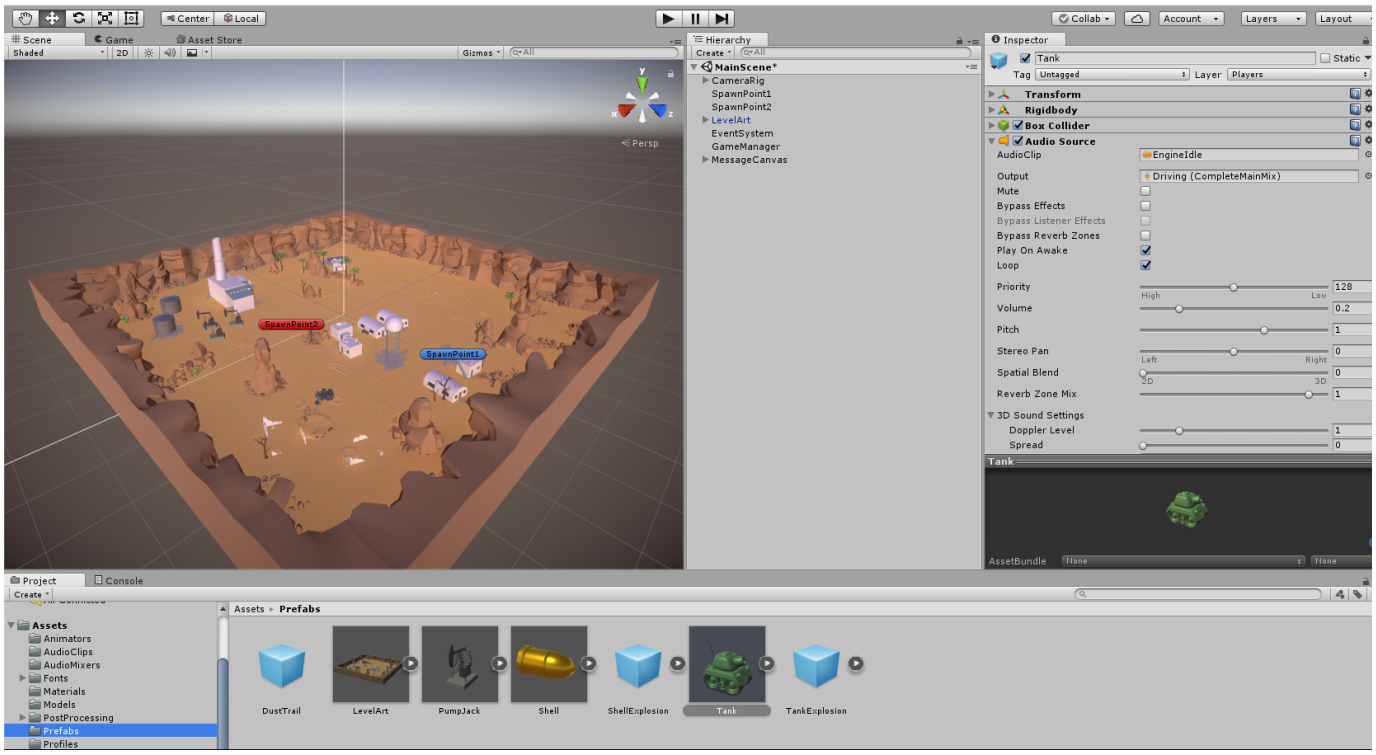
4. Set ouput to Driving



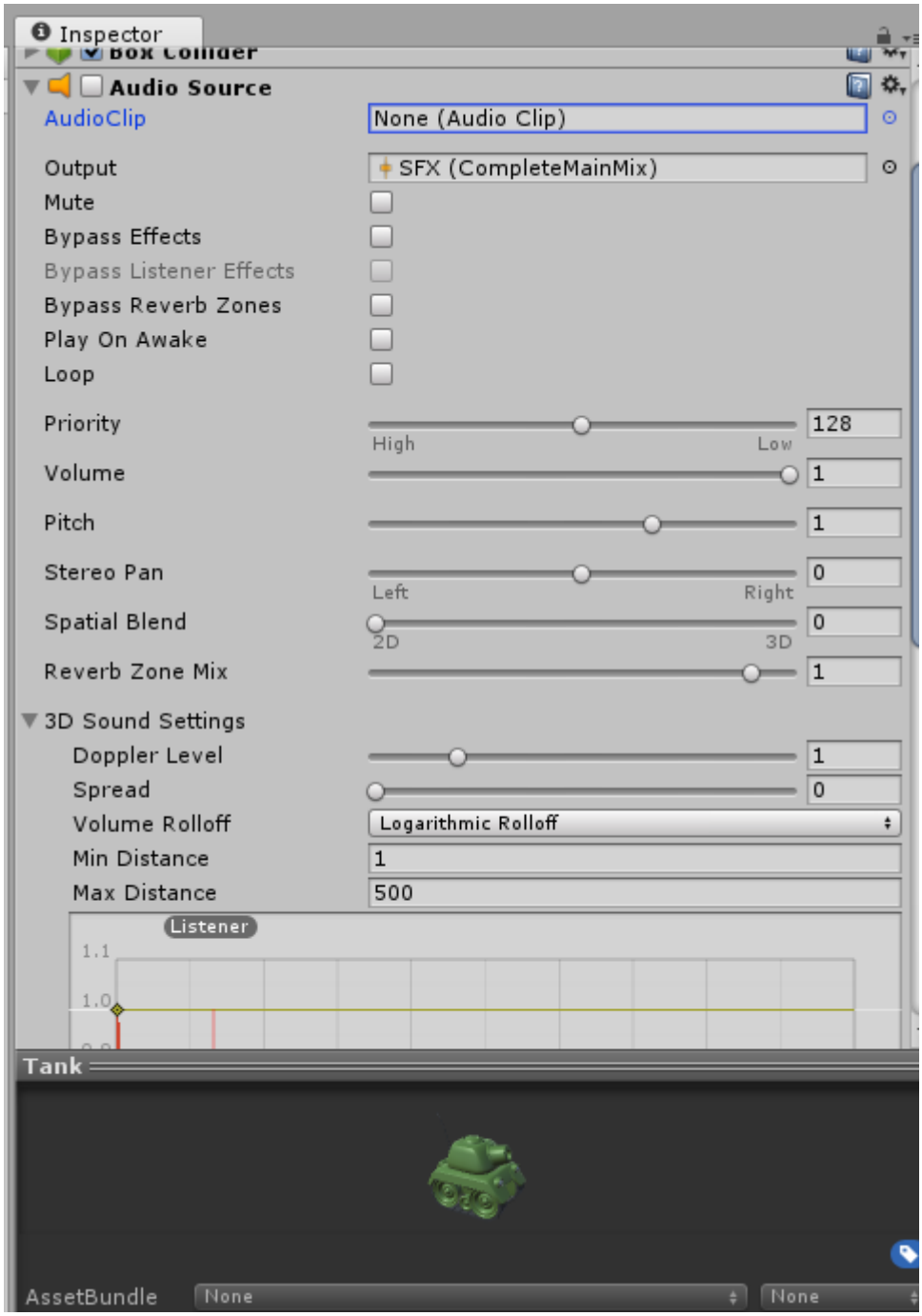
5. Check the Play On Awake and Loop check boxes

Finished Tank Engine Audio Source Component





Add Tank Impact Audio Component


Inspector
Box Collider
Audio Source
AudioClip
None (Audio Clip)
Output
SFX (CompleteMainMix)
Mute
Bypass Effects
Bypass Listener Effects
Bypass Reverb Zones
Play On Awake
Loop
Priority
128
High
Low
Volume
1
Pitch
1
Stereo Pan
0
Left
Right
Spatial Blend
0
2D
3D
Reverb Zone Mix
1
3D Sound Settings
Doppler Level
1
Spread
0
Volume Rolloff
Logarithmic Rolloff
Min Distance
1
Max Distance
500
Listener
1.1
1.0
Tank
AssetBundle
None
None