Lesson Plan For Space Shooters

*Note: Only covers new topics.*

Day 1: Getting used to it

* Windows
* GameObject
* Rigidbody
* Scripting
  + void Start(), void FixedUpdate(), void Update()
  + GetComponent
  + Public variables

Day 2: Starting Space Shooters

* Camera
* Models
* Collider
* Scripting
  + Mathf (Clamp)

Day 3/4: Lasers and Bad Guys

* Prefabs
* Scripting
  + OnTriggerEnter
  + Destroy
  + Instantiate
  + Time

Day 4/5: Boundary and GameController

* Tagging
* CompareTag

Day 5/6: Sound, UI, and Special Effects

* Sound
* UI
* Special Effects

### Day 1: Getting used to it

* Goal: Get a Ball (GameObject) Rolling on a Plane (GameObject) using a script (PlayerMover).
* Assignment: Open up Unity, Create a Project (name can be anything)
* **Windows**
  + Scene: Where you edit your game.
  + Game: A preview of your game (you cannot edit your scene in the Game view).
  + Inspector: The small details and components.
* **GameObject**
  + The building blocks of the game.
  + Their traits can be edited by the components.
  + Assignment: Create a Plane at the origin. Create a Sphere right above it (no overlap or gaps between) (Hint: A regular Sphere would have a radius of 0.5 Unity units).
* **Rigidbody**
  + Assignment: Place a Rigidbody component on the Sphere (Add Component, Rigidbody).
  + For physics people: Makes the GameObject obey the laws of physics (1st, 2nd, and 3rd law)
  + For non-physics people: Makes it a movable object.
    - Gravity will cause the object to fall down the y-axis
  + Demo: Unmovable object
* **Scripting**
  + Assignment: Create a PlayerMover Script for your ball.
  + Click the ‘Add Component’ button (make sure that the Sphere is selected). Type in “Player Mover” and confirm twice. This will create a new script called “Player Mover for the Sphere.
  + Declare a private Rigidbody.
  + **void Start()**
    - The Start() method is called by the Unity Engine when the GameObject is activated (like when the GameObject is present when the game is started
    - In the Start() method, type “rb = GetComponent<Rigidbody>()”
      * This saves typing and sets our player’s Rigidbody as rb.
  + **void FixedUpdate()**
    - The FixedUpdate() method is always called by the Unity Engine when the script and GameObject are active (this is usually used when there’s physics involved).
    - Add a “void FixedUpdate()” method and type inside of it “float horizontal = Input.GetAxis(“Horizontal”);” and “float vertical = Input.GetAxis(“Vertical”);”
      * These record your input when you press the arrow keys.
    - Difference between Update() and FixedUpdate()
  + **GetComponent<Rigidbody>().velocity**
    - Add a “rb.velocity = new Vector3(horizontal, 0.0f, vertical);”
    - This makes the velocity of the GameObject change based on the input of the player.
    - **Physics Lesson**
      * Vectors: A magnitude and a direction.
        + Position vectors: The magnitude is the displacement from the origin, and the direction is where the object is relative to the origin.
        + Velocity vectors: The magnitude is the speed, and the direction is where the object is going.
        + Force vectors: The magnitude is the magnitude of the force, and the direction is where the object’s acceleration is pointing (not required for this lesson).
    - When you play it, attempt to move the GameObject (it is slowwwww).
    - We want to be able to change the speed of the GameObject (but also look professional while we’re at it).
    - **Public int speed**
      * Declare a PUBLIC int and name it (speed).
      * In the rb.velocity line, multiply the Vector3 with the “speed”.
        + Just like it says, it multiplies the velocity of the rigidbody by the speed.
      * Before testing it, you need to change the speed.
      * When you get back to the editor, there should be a new option in the inspector for the PlayerMover script. It should have the word “speed” next to it..
      * Change the number from 0 to 8 and test it. The player should go faster.
    - Save and run
* Congratulations, you have finished your first game in Unity!

### 

### 

### Day 2: Starting Space Shooters

* Pretty much doing the same thing as Day 1
* Goal: Should have a Player GameObject that is able to move with the Player’s control with a Black background and a boundary
* **Background**
  + Assignment: Have the Game view be a black screen (tougher than you think)
  + There are many ways to changing the background, but we’ll be focusing more on the Camera settings.
  + Rotate the MainCamera (should be default, if not, add a Camera using the GameObject tab) 90 degrees on the x-axis.
  + If you check the Game view, the screen is gray. To change this, we must change the Clear Flags from Skybox to Solid Color
    - Clear Flags replaces the things that it can’t render with something else (by default, it uses SkyBox)
    - Foggy simile.
    - In the Inspector, click on the dropbox next to Clear Flags and change it to the Solid Color. Your screen should turn blue.
    - This means that whatever the Camera can’t render, it turns it into the background color
    - Change the background Color to black.
* **Models**
  + Assignment: Drag and drop a Good Guy model from the models folder to the scene view.
    - Models are just custom shaped GameObjects. They are equivalent to a GameObject cube.
  + When you drag the model into the Scene, it’s missing two crucial components
    - Rigidbody (Review what is a Rigidbody)
    - Collider (This is not enabled by default for models)
* **Collider**
  + Assignment: Add a Mesh Collider to the Good Guy GameObject
  + Colliders allow GameObjects to interact with other GameObjects when they “collide.”
  + Without a collider, the GameObjects will just go right through.
    - Using Add Component, add a Mesh Collider
    - Mesh Colliders need meshes in order to be able to detect a collision. Luckily for us, the mesh is automatically the plane from the Good Guy Model it was made.
    - Select Convex (this must be enabled to be able to collide with other Mesh Colliders).
* **Adjusting the Game View**
  + Assignment: Edit the Camera component to get a good view of the Good Guy
  + When you check out the Game view, notice that you can’t see anything. This is because the Camera isn’t in the proper position.
  + Reset the Camera’s position and increase the y variable by 10 (set it to 0, 10 ,0).
  + You should now be able to see the Good Guy, but there’s two more steps to do on the Camera.
  + We need to change the projection from Perspective to Orthographic
    - Perspective vs Orthographic Demo
  + We also need to change the size from 5 to 15. This allows for more of the Game to be viewed
  + Finally, we need to change the aspect view from Free Aspect to 600 x 900 (found on the top left corner of the Game View).
* **Scripting**
  + Assignment: Create a PlayerMover script for your Good Guy GameObject and create a Boundary class.
  + Do the same thing as the scripting from Day 1.
  + **Boundary/Mathf.Clamp**
    - Try moving the Player out of the screen. Oops, problem. Don’t want that happening, so let us create a boundary.
    - Mathf.Clamp Lesson
    - Boundary
      * [System.Serializable]
      * xMin, xMax, yMin, yMax
  + **Optional: Rotator script**

### 

### 

### Day 3/4: Lasers and Bad Guys

* Goal: Create a Laser and a Bad Guy prefab and a DestroyByContact Script on the Bad Guy. Also, create a Boundary GameObject to destroy the excess Bad Guys and Lasers as the leave the area.
* **Lasers**
  + Assignment: Create a Laser GameObject with the Mover Script.
  + Create a GameObject (Sphere, Cube, Plane) and rename it “Laser”.
  + Make sure that the laser has a Rigidbody component and a (Mesh/Sphere/Capsule/etc.) Collider component.
  + Create a new Script and call it Mover
  + **Mover**
    - Assignment: Create the Mover Script. The GameObject must be able to move by z-axis at a constant speed, which can be edited directly by the editor.
    - Return the editor and set the speed to some positive value.
  + At this point, when you test the game, the Laser GameObject should be moving up at a constant velocity.
* **Bad Guy**
  + Assignment: Create a Bad Guy GameObject using the Bad Guy model given with the Mover Script and a DestoryByContact Script.
  + Click and drag a Bad Guy Model to the Hierarchy and set its position so that it is visible in the Game view.
  + Make sure that the GameObject has the Rigidbody and Mesh Collider components.
  + Use the same Mover Script as with the Laser GameObject (all scripts should be a script folder).
  + Create a new Script and name it “DestroyByContact”.
  + **DestroyByContact**
    - Assignment: Create the DestroyByContact Script. This means that when the GameObject’s (Bad Guy) collider collides with another GameObject’s (Good Guy, Laser) collider, this script will “destroy” both of them.
    - **void OnTriggerEnter(Collider other)**
      * This is a method called by the Unity Engine when a Collider (other) collides with this GameObject’s (Bad Guy) collider.
      * **Trigger**
        + A trigger collider is intangible by other GameObjects and calls the OnTrigger methods (OnTriggerEnter, OnTriggerExit).
        + Triggers have to be intangible because another GameObject’s collider has to go through the Trigger’s collider in order for the OnTrigger methods to be called.
      * Create this method and inside write Destroy(other.GameObject);
      * **Destroy(other.GameObject);**
        + The Destroy method removes a GameObject off the hierarchy.
        + Note: Changes while testing the game are only temporary. It’ll revert back to its original settings afterwards.
        + Destroy this GameObject (Bad Guy) as well (Destroy(GameObject);)
  + Go back to the editor and place a negative value for the Mover Script so that the Bad Guy moves the other way.
  + Test the game and try to Destroy the Good Guy GameObject.
* **Prefabs**
  + Assignment: Save both the Bad Guy GameObject and the Laser GameObject as prefabs.
  + Drag and Drop both the Bad Guy GameObject and the Laser GameObject into the assets (probably want to create a Prefabs folder (or a folder called Prefabs)).
  + Prefabs are like models.
  + Delete the Bad Guy and Laser GameObjects in the hierarchy (don’t need them anymore).
* **Laser Spawn**
  + Assignment: Create the GameObject LaserSpawn to spawn the Laser GameObjects.
  + Create an Empty GameObject (yes, an empty one). Rename it “Laser Spawn”.
  + This is the reference to where we will spawn our lasers.
  + Make it the child of the Good Guy GameObject (drag and drop it on the Good Guy).
  + Now, the Laser Spawn will move whether the Good Guy GameObject moves.
  + **Instantiate**
    - Assignment: Whenever a certain key is pressed, instantiate a laser at the Laser Spawn location.
    - Instantiate produces a clone GameObject while the Game is running.
    - You need to Instantiate a Laser GameObject at the ShotSpawn location and rotation(Instantiate(laser.GameObject, shotSpawn.position, shotSpawn.rotation);) when the a certain button (“Fire1”) returns true (Input.getButton(“Fire1”);). You also need to declare what a Laser and ShotSpawn is (public GameObject laser/shotSpawn).
  + **Time.time**
    - You also need to create separate shots (otherwise, tons of lasers will be spawning with just a simple press).
    - Figure a way to split the shots using Time.time (which returns how much time has passed after the start of the game (float)).

### Day 4/5: Boundary and GameController

* **Boundary**
  + Assignment: Create a Boundary that will Destroy the GameObject clones when they pass through.
  + We need to destroy the clones as they spawn, otherwise, they can take up storage space and slow stuff down.
  + Use a GameObject Cube’s collider to act as the Boundary.
    - Be sure to disable the Mesh Renderer (shows the GameObject).
    - Use an OnTriggerExit method to destroy the extra GameObjects.
  + We don’t want the Bad Guy’s DestroyByContact Script to destroy the Boundary however when it collides.
  + We can use tags to identify a boundary.
* **Tagging**
  + Assignment: Create a ‘Boundary’ tag and use it to identify the Boundary in the DestroyByContact script.
  + Create a tag underneath the GameObject’s name in the Inspector window.
  + Tag the boundary
  + In the DestroyByContact script, use ‘other.tag’ and check if it’s equal to the Boundary tag name.
  + Make sure that if it is the Boundary, do not destroy it when colliding.
* **GameController**
  + Assignment: Create the GameController Script to
    - Spawn the Bad Guy GameObjects
    - Keep track the score
    - Enable Restarting the Game.
  + Hints:
    - You want to spawn the BadGuys at the top of the Game view, but not just in one position. Use Random.Range to produce a random number in a range.
    - We don’t want to spawn the Bad Guy GameObjects every single time, so we also need a Time.time to measure the spawn times between Bad Guy GameObjects.
    - Instantiate
    - To restart a level, use Application.LoadLevel(Application.loadedLevel);
    - To restart, the ship must be already destroyed. This has to be called by another script, we need to create a Public method to make some bool variable true (We’ll deal with this later).
* **What to do after a collision (besides destroying)?**
  + When the Bad Guy GameObject detects a collision (with the Laser/GoodGuy GameObject) with the DestroyByCollision script, we need to tell the GameController to add the score.
  + First, we need to allow the DestroyByCollision to “declare” the GameController script. (Declare a private GameController).
  + Next, we need to find the GameController GameObject using tags.
    - We can’t simply just drag and drop the GameController GameObject because it is not a prefab (complicated stuff to think about).
    - To find the GameController, use GameObject.FindGameObjectWithTag.
  + Save the GameController Script component in a variable.
  + Now, we will be able to call the public methods from the GameController script. Let’s create them.
* **isGameOver()**
  + Assignment: Create a public isGameOver() method in the GameController script that, when called, will make a bool variable true.
  + Hints:
    - The player must be destroyed when this is called. How then can we identify which collider is the GameObject?
* **addScore()**
  + Assignment: Create a public addScore() method that adds the score in the GameController when a Laser (and possibly and Good Guy GameObject) collides with a Bad Guy GameObject.

### 

### Day 5/6: Sound, UI, and Special Effects

* Assignment: We need to create the effects so that our game would seem pretty legit.
* **Sound**
  + Assignment: Add sound components on the Bad Guy, Good Guy, and Laser GameObjects. Add a background sound on the GameController.
  + Use the sound clips that I given you to add them to the prefabs GameObjects.
  + On the GameController, add the background Sound component and loop it.
* **UI**
  + Assignment: Add the ScoreText and the GameOverText.
  + Pivots and Anchors.
    - Pivot: Where the object rotates itself.
      * Think of it as a hinge placed on a point, and the object is attached to the hinge by a invisible rod.
    - Anchor: The object maintains a set distance from this point.
  + **Scripting**
    - Assignment: Use the GameController Script to change the values of the Score Text.
    - Hints:
      * Make sure the script is “using” UnityEngine.UI.
* **Particle Effects**
  + Assignment: Instantiate a particle effect at the Bad Guy (and Good Guy) GameObject’s position a Bad Guy and a Laser (or Good Guy) GameObject collides.
  + Hints:
    - The particle effects are given to you.
* **Materials**
  + Assignment: Assign the GameObjects some materials to provide color to your game.
  + Simply create a new Material in the Assets menu and drag it onto the GameObject in the Scene view.