Definition:

- BAFK is the working title of a space themed bullet hell shooter game developed in Unity Engine.
- The player controls a ship with 3 main abilities and fights worm based enemies.
- Many enemies can spawn at a time, and this scales the further the player gets into the game.
- A single boss exists which can spawn on the 5th wave of enemies, and when defeated offers the player the opportunity to augment their ships power in the form of relics for the remainder of the run.
- Currently the run is endless, but the idea of an encapsulated run that lasts perhaps 20 waves, with each multiple of 5 offering a relic and a noticeable scale in the difficulty of subsequent waves.
- Relics are a powerful augmentation to the ship that last the remainder of the run.
- Relics are offered randomly, 3 at a time.
- There are only 3 available relics right now, meaning

❏ A currency system exists in the game where enemies can drop resources using RNG. The currency system could be extended to have currencies that are only acquirable through crafting, to introduce new materials that cannot be acquired from enemies.

❏ The use of currency is to purchase permanent upgrades to the ship, and permanent debuffs for enemies - more damage, firing faster, and lowering enemy health are all within the confines of such a shop. **This feature is not currently implemented. The currency system exists, but is not currently tied to the shop, meaning the upgrades are cosmetic.**

❏ Currently only 1 ship is programmed into the game, but others can be added.

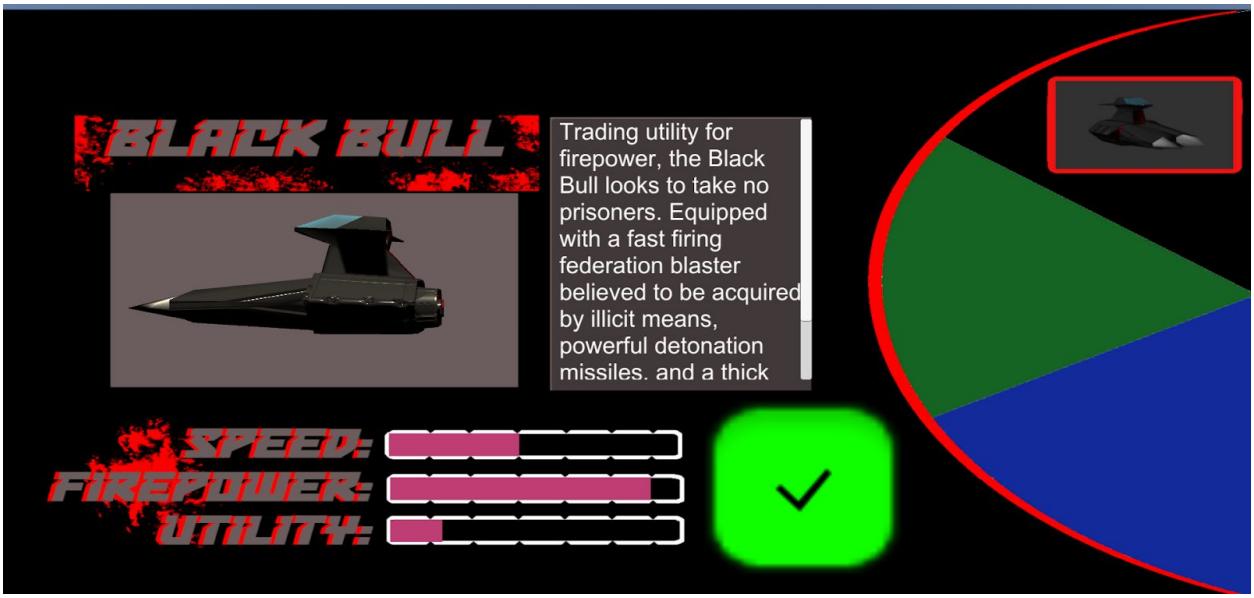
Installation:

- Included with the files will be a zipped folder which will contain the full contents of the game. To run the game, simply run “BAFK.exe”.

Users Manual:

To start a new game, click the ‘Play’ button on the main menu.

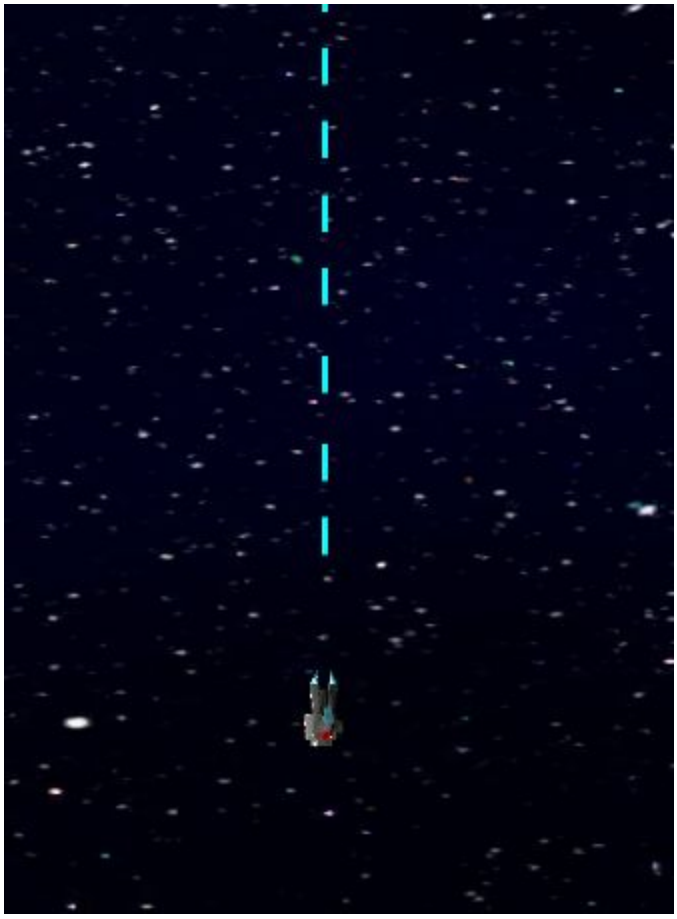
Select a ship from the selection screen and hit the green check mark to begin your run:



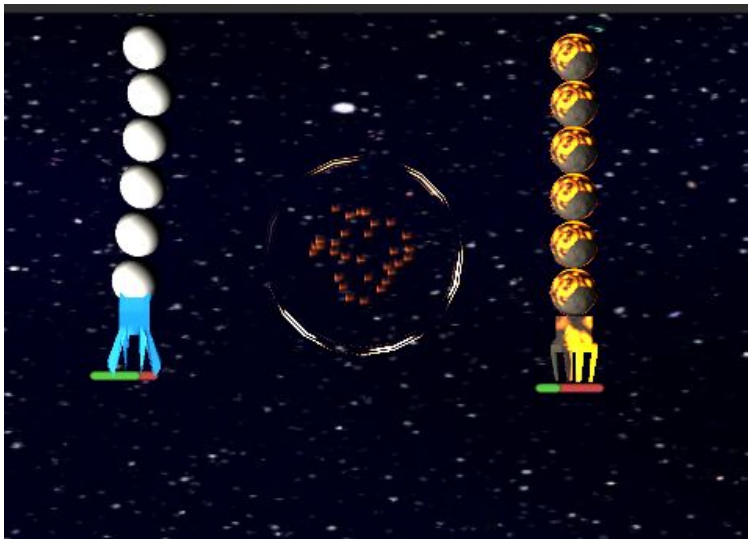
This is your ship, move it with a thumbstick on controller, or arrow keys on a keyboard.

Your ship has 3 attacks:

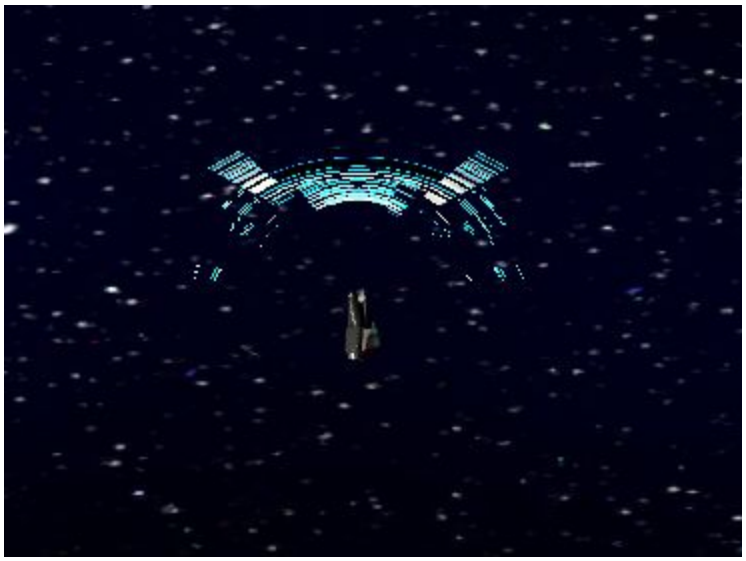
Right Trigger or clicking the left mouse is your primary fire:



Left Trigger or right mouse click is your Alt Ability. By default it is a lobbed rocket that explodes on contact with an enemy or by itself after a few seconds.
Note that both the rocket and the explosion will damage enemies, so try to hit them!



A third special attack exists, and can be executed by pressing the right bumper on the controller, or left shift on the keyboard. After a brief animation, it will launch forward and halve the health of any regular enemy it hits, and continue to do constant damage while in contact with an enemy.

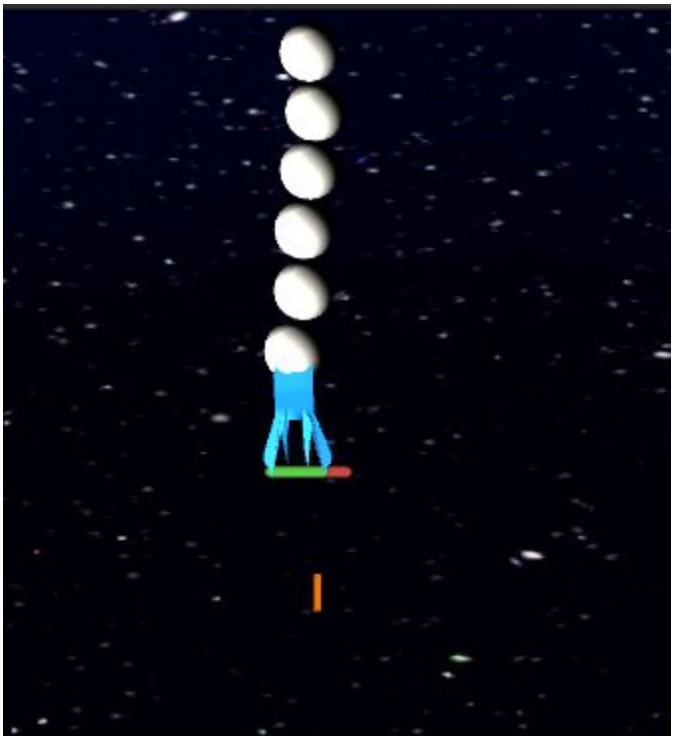


Your health, and the cooldowns of your abilities can be monitored in the bottom left of the screen.

4 basic enemies exist in the game.

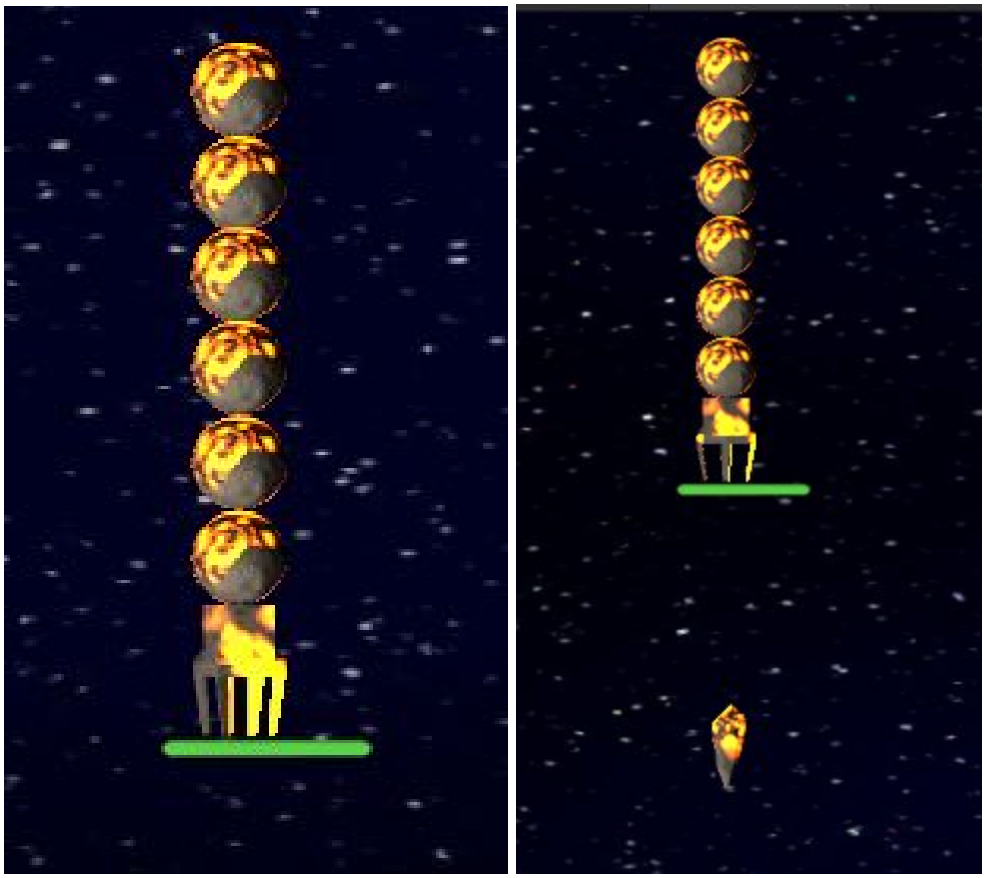
The Standard wyrm:

Shoots basic bullets at a regular interval ~(0.8-1.5s)
Has 100 health, a rather vanilla enemy.



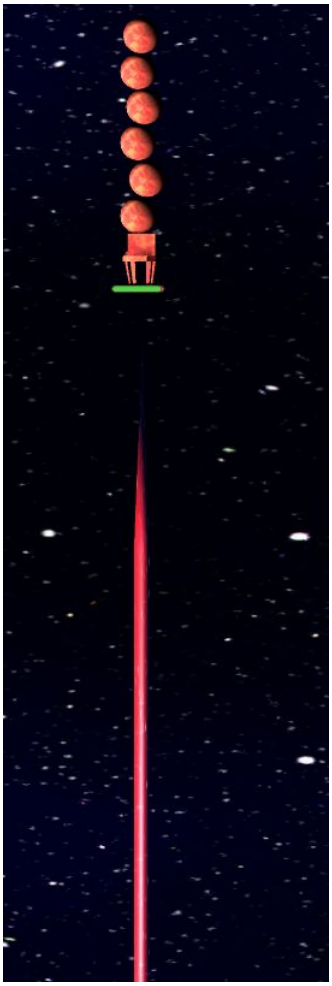
The magma wyrm:

A hardy enemy with 150 health, shoots a fast moving lava spike every 1-2 seconds that deals 15 damage.



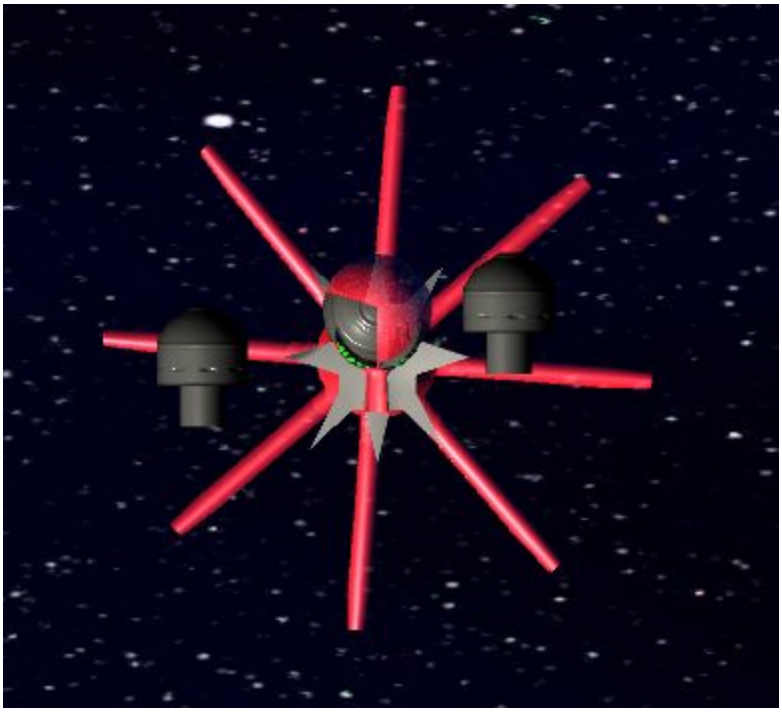
The lazer wyrm:

The lazer wyrm charges a powerful attack that deals damage every frame the player is inside it, don’t stay there too long!



The Boss:

Wave 5 features an octopus boss with a few attack mechanics.



The cannons surrounding the boss rotate and shoot every 0.8 seconds.

The boss can teleport to the center of the screen and extend its tentacles, and then spin in place, forcing the player to move to avoid them.

The boss also has an attack that teleports it randomly around the screen, where it will throw the sawblade in a circular arc each teleport.

Design:

The core gameplay loop of the game is as follows:
Play a run -> accrue resources, eventually die or win -> upgrade your ship/unlock new ships

The first thing to understand about the project is how Unity implements scenes, GameObjects, and some properties of its scripts.

In Unity a scene can be thought of as a complete screen. The scene can be populated with UI elements, GameObjects, and most other assets that Unity manages. Scenes can have default components that are created automatically when the scene is loaded, or components can be added at runtime. The components design can either come from “prefabs” - prefabricated game objects which can be placed inside an assets folder, or created on the fly and all necessary properties set by the runtime script. The first option is usually suitable for most purposes.

When scenes are changed, all previous game objects are destroyed, unless we specifically tell Unity not to destroy them. This can create some problems which I will address later.

Game Objects are the building blocks of a Unity scene. Empty game objects contain nothing but a *transform*, and can be thought of as an invisible container at some point in 3D space. Unity gameobjects can have components, things such as lights, images, canvases, scripts, colliders, or rigidbodies. Objects with many parts can be put into a container together, and these various parts will in turn give complexity to the game object to form something distinct and/or unique.

Imagine building a car into the game, the game object containing this car will need the following components depending on the functionality of the car:

- A renderer, which renders the model of the car, with an attached 3D model.
- A rigidbody, if we wish to perform any physics operations on the car.
- A collider, if we wish for the car to be able to collide with other objects.
- An animation controller, if the car ever needs to change animations at runtime.
- An attached script, coded to create a desired behavior.

Our car is quickly becoming very complicated even though Unity is handling a lot of the overhead for all of these systems. The first major challenge of the project was identifying the task we need to perform, consulting the documentation to find out which

Unity components were capable of carrying out this task, and then deciding and implementing a component for best fit.

Unity's scripting is handled with C#, and the scripts have some unique properties.

- There is a start() function that is called upon the creation of the game object that the script is attached to, it is only called once, but can be useful for setting properties or behaviors that you only need to occur once. Perhaps setting a boolean value for the animation controller for an animation you only need one time.
- The update() function is called once per frame, always. This can cause while loops to break if they are in different functions. We have to be aware of this while coding. It also introduces a new issue: if frame rate is irregular, this can break certain behaviors. If we are transforming an object in update(), depending on the computer's performance, we will receive different results after different periods of time. Not always a disaster, but poor design.
- fixedUpdate() provides a solution to the aforementioned problem. It is favored for handling any type of physics or movement behavior because it updates in a fixed interval based on the physics engine. This means that calculations done within this loop are called at a fixed real time interval, and frame rate won't have an impact on the regularity of some physics.

The first challenge of the project was with the ship itself, regarding movement.

Two main ways of moving an object with Unity have to do with a transform, or applying physics to a rigidbody.

Moving a transform will translate the object along some world axis (x,y, or z) a specified amount, usually a floating point number. It is not usually recommended to use this method if you are trying to build a physics based game with lots of collisions and/or reactions. This is because the transform behavior applies instant velocity instead of any kind of acceleration. If we are looking for a realistic physics simulation, we should be adding force at the impact point of a collision, and should ideally never be transforming.

Because the game we are developing is more "arcade" by nature and is not meant to simulate real physics, using the transform method is fine for us.

Another component of the ship script will be inputs. We need to receive inputs from the players controller or keyboard and translate them to actions ingame. Unity has an input manager that allows us to map various inputs to a named input. For example, we use "Fire1" as the name for our primary fire, and we can map any number of inputs to it,

whether it be from the mouse, the keyboard, or a controller. In the script, we check for inputs each frame, namely inputs that match “Fire1”. We can store the reading for the frame inside of a boolean, and decide to take an action if this is true. So Unity takes the complex action of mapping inputs, and lets us handle any of them with just a few frames.

The primary fire of the ship wasn’t a particularly difficult task to implement, but it did present one interesting problem. In the update() loop of the shooting script, when we detect the “Fire1” input being pressed, we fire. There is another condition, which is if enough time has passed since our last shot. Fortunately unity allows us to continually increment our currently waited time, making this portion of the task easy. Once inside the shoot() function, we simply create a temporary game object and instantiate it as a “bullet” prefab at the position of the bulletEmitter at the front of the ship, and apply a physics force to it.

The bullet prefab has all of the components we require: its own script to handle any kind of collision with another object, a rigidbody so physics forces can be applied to it, and a material to give it a glowing laser feel.

I did notice when initially adding the primary attack that it was possible to shoot yourself by running into your own bullets. The most egregious offense was when you were moving in the direction that the bullets would be fired, as you would run into the bullet as it spawned. The solution to this was once again thanks to an already existing Unity system. Unity has a physics layering system that prevents or allows collision between specified layers.

For the purposes of the project, I created 4 layers: enemies, player, enemy bullets, and player bullets. Interactions are as such:

	Player	Player Bullets	Enemies	Enemy Bullets
Player	-		x	x
Player Bullets		-	x	
Enemies	x	x	-	
Enemy Bullets	x			-

The implications are as follows:

- Enemies can’t shoot other enemies.

- The player cannot shoot themself.
- Normal collision between Enemies + player bullets and player + enemy bullets can still occur.
- Enemies that require direct collision can still hit the player to deal damage.

Populating the game with enemies was a multi step process involving:

- Modelling the enemy, making sure that any parts that need to be animated independently need to be modelled as such. For example, if we want to animate a human arm, we would want to make sure the top half of the arm, the bottom half of the arm, the hand, and the 5 fingers are all separate parts that belong to the same model, so we can independently control each part as we animate it.
- Texturing the enemy, this is a time consuming process and I gained a new appreciation for even the simplest of graphics in games from attempting this portion myself.
- Creating a prefab of the enemy. We create a premade enemy, ready to be cloned and used against the player. The common components to be added are a behavior script, a rigidbody, and a collider, and an animation controller.

Once the prefab is complete, we can create temporary gameobjects in our scripts that we can then instantiate at specific points in 3d space.

In the run screen, where the main game is played, we have a gameobject called WaveController with nothing attached but a script for sending the waves of enemies. Unity provides a nice feature where when we declare public objects, we can fill these spaces in the scene editor and use them in the script.

Example: public int[] arr = new int[5];

This will give us 5 spaces on the frontend to plug in numbers, imagine we populate with 6, 2, 10, 15, 7

Now in our script when we reference arr[4] we get 7.

This can also be done with things such as prefabs. Meaning we can have all of our different enemy types populate a public array of gameobjects, and when we instantiate we just need to reference this public array.

The behavior of WaveController works as such:

- In the update() loop, if there isn't a wave currently occurring (either points remaining or enemies present) send a wave.
- When we send a wave, set an integer number of points to be used by the wave.
- While there are remaining points, randomly spawn an enemy (using RNG) every 4.0s. Each enemy subtracts some amount of points from the remaining points.
- Once we are out of points, and no enemies remaining on the screen (we have a function that checks and sets a boolean), send the next wave and repeat the process.
- On the 5th wave, don't assign any points and spawn 1 boss instead of any enemies.
- After the 5th wave we give an option to select a "relic" using a UI overlay with 3 images. We also pause the future wave spawns until a selection has been made.

Another important topic is collisions versus triggers. A collision occurs when between two objects and uses the rigidbodies attached to the objects to simulate physics. This was initially frustrating because when bullets or other projectiles were colliding with enemies they were either pushing enemies away or the projectiles themselves were being pushed away.

Collisions are handled in scripting with the aptly named OnCollisionEnter and OnCollisionStay functions, and are good when we want actual collisions to occur. When we don't want to simulate any physics the collider components used to detect collisions have a checkbox with an option "Is Trigger". When this option is checked on one collider, we instead call the OnTriggerEnter and OnTriggerStay. This means we can respond in scripts when object colliders enter each others' space, and not simulate any physics. Perfect!

Interactive UI is something we are used to, and it took a little digging but Unity has an elegant solution for handling the creation of these types of components. Unity has an EventSystem, which acts as a controller for when a gameobject is clicked, moused over, or dragged. We tell the EventSystem to listen for a specific action on an object (a click, or a drag for instance). In the script we simply have to tell the event system what kind of inputs to listen for, and add an event system class function for handling the action.

For example, mousing over the buttons on the main menu causes the text on them to turn green, we are using the event system OnMouseEnter() function to detect when our cursor is over the game object attached. When it is called we replace the image, and when the mouse exits the region we change the image again.

The same technique was employed for the upgrades menu, and was combined with attaching public game objects to the script. Meaning, when we click the buttons on that menu, we can interact with other game objects in the scene.

The final challenge involves both an implemented system and an unimplemented system in the game, but I would like to discuss both as learning about them was beneficial to me.

As discussed scenes within Unity are encapsulated collections of game objects. When we load a scene, all the default game objects are created, and the scripts attached to these game objects begin to run. When scenes are changed, the gameobjects of the old scene are all destroyed, and the attached scripts cease to run. This can create a problem if we are storing some kind of data in a script and need to pass it to the next scene. Specifically for the project, I need to pass data from the upgrade menu to the runScreen, and I need to pass data from the runScreen to the upgrade menu to properly keep track of accrued resources.

There are two main ways of dealing with the issue. The first is to save the data to a file, and subsequently read that data on load. This is probably the best solution to build out, as you can reuse the code and simply change the parsing for portability in different sections of your project. The second is to store all the data needed in a gameobject and call the built in Unity function DontDestroyOnLoad(). This function prevents a gameobject from being destroyed when a new scene is loaded, and lets you preserve the data living in the script.

I did run into a bug with the shop menu where I was continually creating shop menu game objects. I fixed this by modifying the script to calculate on start if there was more than 1 shop menu, if there was destroy this gameobject. The original shop menu with the player data would persist, and as new shop menus were created, they would see they were not the original and destroy themselves. Well behaved clones.

The Code:

I will paste each script into the section below with headers to help navigation. I will also group them by purpose. (the enemy behaviors will go together, the collision will go together, etc.)

ENEMY BEHAVIORS

basicWyrm.cs - Basic Enemy Wyrm

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

public class basicWyrm : MonoBehaviour
{
    public float health = 100.0f;
    public RectTransform healthbar;
    public GameObject bulletRed;
    public float bulletforce;
    public float bulletlifeTime;
    public GameObject basicbulletEmitter;
    public GameObject[] drops = new GameObject[5];
    public enum mapped { tin, copper, coal, gold, wyrm}


    // Start is called before the first frame update
    void Start()
    {
        //Have the enemies shoot at a bounded random interval.
        Invoke("shoot", UnityEngine.Random.Range(0.8f, 1.5f));
    }
```

```
// Update is called once per frame
void Update()
{

    if (health < 1.0f)
    {
        die();
    }
    healthbar.sizeDelta = new Vector2(health / 2, healthbar.sizeDelta.y);


}


/**/
/*
void shoot()

NAME

    basicWyrm::shoot() - handles shooting for the basic wyrm enemy.

SYNOPSIS

    void shoot()

DESCRIPTION

    Instantiate a temporary game object - essentially a clone, attach a rigidbody (for
physics handling) and add a force vector to the attached rigidbody.
    this will effectively "fire" a bullet. At the end of the function we make sure the bullet
will be destroyed one way or another, and we call the function again
    using the same bounded random interval. This gets around using any kind of
boolean handling in our update loop.

RETURNS
```

AUTHOR

James P Giordano

DATE

8/3/2020

*/
/**/
void shoot()
{
GameObject tempBullet;
tempBullet = Instantiate(bulletRed, basicbulletEmitter.transform.position, bulletRed.transform.rotation) as GameObject;

Rigidbody tempRigidbody;
tempRigidbody = tempBullet.GetComponent<Rigidbody>();

tempRigidbody.AddForce(0, 0, -bulletforce);

Destroy(tempBullet, bulletlifeTime);
Invoke("shoot", UnityEngine.Random.Range(0.8f, 1.5f));
}
/* void shoot()*/

/**/
/*
void die()

NAME

die()

SYNOPSIS

DESCRIPTION

Responsible for destroying the gameobject attached to this script when it runs out of health. Also included before destroying the gameobject

is the random generation of loot to be dropped. An integer from 1-100 inclusive is rolled and sets a mapped value to be tested against a switch case.

This then selects from an array the correct resource and to display.

for example: (random roll) -> 67 -> maps to "coal" from enum -> case "coal" in the switch -> accesses the array of possible resource drops at dropArray[2]

this allows for modularity at any stage of the generation, The chance of each resource, OR what the resource is.

Instantiates an image of the drop as a pop-up on screen. Then finishes destroying the dying gameObject we are attached to.

RETURNS

nothing.

AUTHOR

James P. Giordano

DATE

8/3/2020

```
*/
/**/
void die()
{
    GameObject tempImage;
    int roll = UnityEngine.Random.Range(1, 101);
    mapped mappedStr;
    if (roll < 30)
    {
        mappedStr = (mapped)0;
    }
    else if (roll >= 30 && roll < 60)
    {
        mappedStr = (mapped)1;
```

```
        }
        else if (roll >= 60 && roll <= 90)
        {
            mappedStr = (mapped)2;
        }
        else if (roll <= 98)
        {
            mappedStr = (mapped)3;
        }
        else
        {
            mappedStr = (mapped)4;
        }


        switch (mappedStr)
        {
            case ((mapped)0):
                tempImage = Instantiate(drops[0], this.gameObject.transform.position,
drops[0].transform.rotation) as GameObject;
                break;
            case ((mapped)1):
                tempImage = Instantiate(drops[1], this.gameObject.transform.position,
drops[1].transform.rotation) as GameObject;
                break;
            case ((mapped)2):
                tempImage = Instantiate(drops[2], this.gameObject.transform.position,
drops[2].transform.rotation) as GameObject;
                break;
            case ((mapped)3):
                tempImage = Instantiate(drops[3], this.gameObject.transform.position,
drops[3].transform.rotation) as GameObject;
                break;
            case ((mapped)4):
                tempImage = Instantiate(drops[4], this.gameObject.transform.position,
drops[4].transform.rotation) as GameObject;
                break;

        }
```

```
        Destroy(this.gameObject);

    }

    //we dont want to have lots of empty clone parent objects floating around, so this just ensures that they are leaving too.
    private void OnDestroy()
    {
        if (transform.parent != null) // if object has a parent
        {
            if (transform.childCount <= 1) // if this object is the last child
            {
                Destroy(transform.parent.gameObject, 0.1f); // destroy parent a few frames later
            }
        }
    }
}
```

BAxeBehavior.cs - Battle Axe Wyrm Behavior

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

public class BAxeBehavior : MonoBehaviour
{

    public float health = 200.0f;
    bool right = true;
    bool movement = true;
    Vector2 screenSize;
    Animator con;
    public float damage = 12.0f;
    public RectTransform healthbar;
    public GameObject[] drops = new GameObject[5];
    public enum mapped { tin, copper, coal, gold, wyrm }

    // Start is called before the first frame update
    void Start()
    {
```

```
        screenSize = new Vector2(Camera.main.aspect * Camera.main.orthographicSize,
Camera.main.orthographicSize);
        con = gameObject.GetComponent<Animator>();
    }

    // Update is called once per frame
    void Update()
    {
        //refresh the size of the health bar in case it has changed.
        healthbar.sizeDelta = new Vector2(health / 2, healthbar.sizeDelta.y);

        if (health < 1.0f)
        {
            die();
        }

        //while it would be nice to not have to worry about this, we need to not be updating our
position while animating. So there needs to be times where we turn this
        //automatic scriped movement off. This is done withe the boolean named movement.
        if (movement)
        {
            //are we moving to the right or the left?
            if (right)
            {
                transform.position = new Vector3(transform.position.x + 0.15f, 0, transform.position.z);
            }
            else
            {
                transform.position = new Vector3(transform.position.x - 0.15f, 0, transform.position.z);
            }
        }

        //if we hit the edge of the screen (left side or right side) we need to play the appropriate
animation for turning around, and then resume our normal movement pattern after the duration
of the animation.
        //the 'right' boolean lets us know if we are going right or not.
        if (transform.position.x > screenSize.x)
        {
            right = false;
            con.SetBool("rightside", true);
            movement = false;
            Invoke("turnOffRight", 1.65f);
        }
        else if (transform.position.x < -screenSize.x)
```

```
        {
            right = true;
            movement = false;
            con.SetBool("leftside", true);
            Invoke("turnOffLeft", 1.65f);
        }

        //once we hit the bottom of the screen, reset the z coordinate to a random bounded
interval.
        if(transform.position.z < -screenSize.y)
        {
            transform.position = new Vector3(transform.position.x, 0, Random.Range(20.0f, 35.0f));
        }
    }

    /**/
    /*
    void turnOffRight()

    NAME

        turnOffRight

    SYNOPSIS


    DESCRIPTION

        sets the 'rightside' variable to false, which is a boolean inside the animator controller that
helps control the state of the animations.
        by setting a boolean here we can determine that it is time to change an animation state.
Also lets us resume our normal movement pattern in
        'Update()'

    RETURNS

        nothing.

    AUTHOR

        James P. Giordano

    DATE
```

```
    8/4/2020

*/
/**/

void turnOffRight()
{
    con.SetBool("rightside", false);
    movement = true;
    //transform.position = new Vector3(transform.position.x, 0, transform.position.z - .05f);
    //transform.Rotate(0.0f, 180.0f, 0.0f, Space.Self);
}

/**/
/*
void turnOffLeft()

NAME

    void turnOffLeft()

SYNOPSIS


DESCRIPTION

    sets the 'leftside' variable to false, which is a boolean inside the animator controller that
helps control the state of the animations.
    by setting a boolean here we can determine that it is time to change an animation state.
Also lets us resume our normal movement pattern in
    'Update()

RETURNS

    nothing.

AUTHOR

    James P Giordano

DATE

    8/4/2020
```

```
*/
/**/

void turnOffLeft()
{
    con.SetBool("leftside", false);
    movement = true;
}
/**/
/*
void OnCollisionEnter()

NAME

    void OnCollisionEnter()

SYNOPSIS


DESCRIPTION

    Whenever a collision occurs involving the gameObject this script is attached to, this
function is automatically called.
    We can get the other game object we have come into contact with, and perform actions
accordingly. In this case, when we collide with the player
    we want to ensure that they lose health, so we can access the script attached to the Player
gameObject and get the float holding the current HP.

RETURNS

      nothing.

AUTHOR

    James P Giordano

DATE

    8/4/2020

*/
/**/
void OnCollisionEnter(Collision other)
{
```

```
    if (other.gameObject.tag == "Player")
    {
        other.gameObject.GetComponent<Player>().currHP -= damage;
    }
}

/**/
/*
void die()

NAME

    die()

SYNOPSIS



DESCRIPTION

    Responsible for destroying the gameobject attached to this script when it runs out of
health. Also included before destroying the gameobject
    is the random generation of loot to be dropped. An integer from 1-100 inclusive is rolled
and sets a mapped value to be tested against a switch case.
    This then selects from an array the correct resource and to display.

    for example: (random roll) -> 67 -> maps to "coal" from enum -> case "coal" in the switch ->
accesses the array of possible resource drops at dropArray[2]
    this allows for modularity at any stage of the generation, The chance of each resource, OR
what the resource is.
    Instantiates an image of the drop as a pop-up on screen. Then finishes destroying the
dying gameObject we are attached to.

RETURNS

    nothing.

AUTHOR

    James P. Giordano

DATE
```

```
    8/3/2020

*/
/**/
void die()
{
    GameObject tempImage;
    int roll = UnityEngine.Random.Range(1, 101);
    mapped mappedStr;
    if (roll < 30)
    {
        mappedStr = (mapped)0;
    }
    else if (roll >= 30 && roll < 60)
    {
        mappedStr = (mapped)1;
    }
    else if (roll >= 60 && roll <= 90)
    {
        mappedStr = (mapped)2;
    }
    else if (roll <= 98)
    {
        mappedStr = (mapped)3;
    }
    else
    {
        mappedStr = (mapped)4;
    }


    switch (mappedStr)
    {
        case ((mapped)0):
            tempImage = Instantiate(drops[0], this.gameObject.transform.position,
drops[0].transform.rotation) as GameObject;
            break;
        case ((mapped)1):
            tempImage = Instantiate(drops[1], this.gameObject.transform.position,
drops[1].transform.rotation) as GameObject;
            break;
        case ((mapped)2):
            tempImage = Instantiate(drops[2], this.gameObject.transform.position,
drops[2].transform.rotation) as GameObject;
```

```
            break;
        case ((mapped)3):
            tempImage = Instantiate(drops[3], this.gameObject.transform.position,
drops[3].transform.rotation) as GameObject;
            break;
        case ((mapped)4):
            tempImage = Instantiate(drops[4], this.gameObject.transform.position,
drops[4].transform.rotation) as GameObject;
            break;

    }
    Destroy(this.gameObject);
  }
}
```

Lazer.cs - Lazer Wyrm Behavior

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

public class lazer : MonoBehaviour
{
  public float health = 100.0f;
  public GameObject laserShot;
  public RectTransform healthbar;
  public GameObject lazerEmitter;
  Animator con;
  public ParticleSystem ps;
  public GameObject[] drops = new GameObject[5];
  public enum mapped { tin, copper, coal, gold, wyrm }

  // Start is called before the first frame update
  void Start()
  {
    //this enemy has an attached particle system, but we only want it to play as a part of its
attack animation.
    ps.Pause();
    Invoke("charge", Random.Range(2.0f, 4.0f));
    con = gameObject.GetComponent<Animator>();
  }

  // Update is called once per frame
  void Update()
```

```
{
    if (health < 1.0f)
    {
        die();
    }
    healthbar.sizeDelta = new Vector2(health / 2, healthbar.sizeDelta.y);
}
/**/
/*
void charge()

NAME

    charge

SYNOPSIS


DESCRIPTION

    acts as a first step for animation purposes. Turns the particle system of the attached gameObject on and waits to fire, simulating a "charge".

RETURNS



AUTHOR

    James P Giordano

DATE

    8/4/2020

*/
/**/
void charge()
{
    ps.Play();
    Invoke("fire", 3.0f);

}
```

```
/**/
/*
void fire()

NAME

    fire

SYNOPSIS


DESCRIPTION

    clears and pauses the particle system, and instantiates a cylindrical gameObject that
expands rapidly in one direction, and moves slowly in that same direction.
    when viewed from a birds eye creates a sort of lazer like effect. Within the scope of this
function we also set a destruction time for the new object, and call our charge function again.

RETURNS



AUTHOR



DATE



*/
/**/

void fire()
{
    ps.Clear();
    ps.Pause();
    GameObject tempLazer;
    tempLazer = Instantiate(laserShot, lazerEmitter.transform.position,
laserShot.transform.rotation) as GameObject;
    //Rigidbody tempRigidbody;
    //tempRigidbody = tempLazer.GetComponent<Rigidbody>();
    //tempRigidbody.AddForce(0, 0, -100.0f, ForceMode.VelocityChange);
```

```
    Destroy(tempLazer, 4.0f);

    Invoke("charge", Random.Range(2.0f, 4.0f));
}

/**/
/*
void die()

NAME

    die()

SYNOPSIS



DESCRIPTION

    Responsible for destroying the gameobject attached to this script when it runs out of
health. Also included before destroying the gameobject
    is the random generation of loot to be dropped. An integer from 1-100 inclusive is rolled
and sets a mapped value to be tested against a switch case.
    This then selects from an array the correct resource and to display.

    for example: (random roll) -> 67 -> maps to "coal" from enum -> case "coal" in the switch ->
accesses the array of possible resource drops at dropArray[2]
    this allows for modularity at any stage of the generation, The chance of each resource, OR
what the resource is.
    Instantiates an image of the drop as a pop-up on screen. Then finishes destroying the
dying gameObject we are attached to.

RETURNS

    nothing.

AUTHOR

    James P. Giordano

DATE

    8/4/2020
```

```
   */
   /**/

   void die()
   {
      GameObject tempImage;
      int roll = UnityEngine.Random.Range(1, 101);
      mapped mappedStr;
      if (roll < 30)
      {
         mappedStr = (mapped)0;
      }
      else if (roll >= 30 && roll < 60)
      {
         mappedStr = (mapped)1;
      }
      else if (roll >= 60 && roll <= 90)
      {
         mappedStr = (mapped)2;
      }
      else if (roll <= 98)
      {
         mappedStr = (mapped)3;
      }
      else
      {
         mappedStr = (mapped)4;
      }


      switch (mappedStr)
      {
         case ((mapped)0):
            tempImage = Instantiate(drops[0], this.gameObject.transform.position,
drops[0].transform.rotation) as GameObject;
            break;
         case ((mapped)1):
            tempImage = Instantiate(drops[1], this.gameObject.transform.position,
drops[1].transform.rotation) as GameObject;
            break;
         case ((mapped)2):
            tempImage = Instantiate(drops[2], this.gameObject.transform.position,
drops[2].transform.rotation) as GameObject;
            break;
```

```
            case ((mapped)3):
                tempImage = Instantiate(drops[3], this.gameObject.transform.position,
drops[3].transform.rotation) as GameObject;
                break;
            case ((mapped)4):
                tempImage = Instantiate(drops[4], this.gameObject.transform.position,
drops[4].transform.rotation) as GameObject;
                break;

        }
        Destroy(this.gameObject);
    }

}
```

Magma.cs - Magma Wyrm Behavior

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

public class magma : MonoBehaviour
{
    public float health = 150.0f;
    public GameObject magmaShot;
    public RectTransform healthbar;
    public GameObject magmaEmitter;
    Animator con;
    public GameObject[] drops = new GameObject[5];
    public enum mapped { tin, copper, coal, gold, wyrm }

    // Start is called before the first frame update
    void Start()
    {
        //we call our charge function in 2-3 seconds and get the animator to be used.
        Invoke("charge", Random.Range(2.0f, 3.0f));
        con = gameObject.GetComponent<Animator>();
    }

    // Update is called once per frame
    void Update()
    {
        if (health < 1.0f)
        {
```

```
            die();
        }
        healthbar.sizeDelta = new Vector2(health / 2, healthbar.sizeDelta.y);
    }

    /**/
    /*
    void charge()

    NAME

        charge

    SYNOPSIS


    DESCRIPTION

        calls fire() after a set amount of time and sets the firing function boolean to true, so as to
line up with animation timing

    RETURNS



    AUTHOR

        James P Giordano

    DATE

        8/7/2020

    */
    /**/

    void charge()
    {
        Invoke("fire", 1.083f);
        con.SetBool("firing", true);

    }

    /**/
```

```
/*
void fire()

NAME



SYNOPSIS


DESCRIPTION

    resets the "firing" bool on the animator and instantiates a projectile to be launched.
    calls charge function again after 2 seconds. forming a complete behavior loop.

RETURNS



AUTHOR

    James P Giordano

DATE

    8/8/2020

*/
/**/

void fire()
{
    con.SetBool("firing", false);
    GameObject tempMagma;
    tempMagma = Instantiate(magmaShot, magmaEmitter.transform.position,
magmaShot.transform.rotation) as GameObject;
    Invoke("charge", 2.0f);

}

/**/
/*
void die()
```

NAME

die()

SYNOPSIS

DESCRIPTION

Responsible for destroying the gameobject attached to this script when it runs out of health. Also included before destroying the gameobject
is the random generation of loot to be dropped. An integer from 1-100 inclusive is rolled and sets a mapped value to be tested against a switch case.
This then selects from an array the correct resource and to display.

for example: (random roll) -> 67 -> maps to "coal" from enum -> case "coal" in the switch -> accesses the array of possible resource drops at dropArray[2]
this allows for modularity at any stage of the generation, The chance of each resource, OR what the resource is.
Instantiates an image of the drop as a pop-up on screen. Then finishes destroying the dying gameObject we are attached to.

RETURNS

nothing.

AUTHOR

James P. Giordano

DATE

8/5/2020

*/
/**/

```
void die()
{
    GameObject tempImage;
    int roll = UnityEngine.Random.Range(1, 101);
    mapped mappedStr;
    if (roll < 30)
```

```
    {
        mappedStr = (mapped)0;
    }
    else if (roll >= 30 && roll < 60)
    {
        mappedStr = (mapped)1;
    }
    else if (roll >= 60 && roll <= 90)
    {
        mappedStr = (mapped)2;
    }
    else if (roll <= 98)
    {
        mappedStr = (mapped)3;
    }
    else
    {
        mappedStr = (mapped)4;
    }


    switch (mappedStr)
    {
        case ((mapped)0):
            tempImage = Instantiate(drops[0], this.gameObject.transform.position,
drops[0].transform.rotation) as GameObject;
            break;
        case ((mapped)1):
            tempImage = Instantiate(drops[1], this.gameObject.transform.position,
drops[1].transform.rotation) as GameObject;
            break;
        case ((mapped)2):
            tempImage = Instantiate(drops[2], this.gameObject.transform.position,
drops[2].transform.rotation) as GameObject;
            break;
        case ((mapped)3):
            tempImage = Instantiate(drops[3], this.gameObject.transform.position,
drops[3].transform.rotation) as GameObject;
            break;
        case ((mapped)4):
            tempImage = Instantiate(drops[4], this.gameObject.transform.position,
drops[4].transform.rotation) as GameObject;
            break;
```

```
        }
        Destroy(this.gameObject);
    }
}
```

OctoBoss.cs - Octopus Boss Behavior

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

public class octoBoss : MonoBehaviour
{

    public float health = 1000.0f;
    public float maxHealth = 1000.0f;
    public RectTransform healthbar;
    public Text hptext;
    Animator con;

    private bool attacking = false;
    private int teleports = 0;
    private float attackTime = 6.0f;
    private float myTime = 0.0f;
    // Start is called before the first frame update
    void Start()
    {
        //attach the animator to a variable. This will be used to influence the animation state from
this script, at the times that we want.
        con = gameObject.GetComponent<Animator>();
    }

    // Update is called once per frame
    void Update()
    {
        //if we are already attacking, don't roll for an attack.
        //there is a base time between attacks, make sure we are above this.

        if (!attacking && myTime > attackTime)
        {
            //if both of these conditions are ok, roll for a random attack. Nice to write like this more
attacks can be added without changing the general idea behind the code.
            int roll = Random.Range(1, 3);
```

```
        switch (roll)
        {
            case 1:
                //The Boss throws the sawblade like object at the player.
                //invoke spinnerAttack() after 2.5 seconds, change the animation state, and check
off that we are attacking so we don't enter this loop again too soon.

                Invoke("spinnerAttack", 2.5f);
                con.SetBool("saw1", true);
                con.SetBool("idle", false);
                attacking = true;
                //reset the attack time.
                myTime = 0;
                break;
            case 2:
                this.gameObject.transform.position = new Vector3(0.0f,
this.gameObject.transform.position.y, 0.0f);
                Invoke("stabAttack", 2.5f);
                con.SetBool("stab", true);
                attacking = true;
                myTime = 0.0f;
                break;
        }

    }
    //continue incrementing time while we wait for our next attack.
    myTime += Time.deltaTime;
    //continue updating the healthbar
    healthbar.sizeDelta = new Vector2((health / maxHealth) * 200, healthbar.sizeDelta.y);
    hptext.text = "BOSS:" + health.ToString() + "/1000";
    if (health <= 0)
    {
        Invoke("die", 0.0f);
    }
}
/**/
/*
void spinnerAttack()

NAME



SYNOPSIS
```

DESCRIPTION

change the animation state and invoke the next portion of the attack "teleport". We Invoke after 1.2 seconds to give enough time for the animations to play.

RETURNS

AUTHOR

James P Giordano

DATE

9/14/2020

```
*/
/**/
void spinnerAttack()
{

    con.SetBool("saw1", false);
    con.SetBool("saw2", true);
    Invoke("teleport", 1.2f);

}

/**/
/*
void teleport()
```

NAME

SYNOPSIS

DESCRIPTION

the spin attack is comprised of two teleports and two swipes with the saw. The if and else blocks handle how many teleports we've taken. If we still have teleports left,

modify the animation state appropriately, teleport to a random (bounded) location and initiate another spinner attack.

if we are out of teleports, change the animation state back to the idle animation, reset our teleport counter, and set the attacking boolean to allow us to roll randomly for attacks again.

RETURNS

AUTHOR

James P Giordano

DATE

9/12/2020

*/

/**/

```
void teleport()
{
    teleports += 1;
    if (teleports < 3)
    {

        con.SetBool("saw1", true);
        con.SetBool("saw2", false);
        this.gameObject.transform.position = new Vector3(Random.Range(-90.0f, 90.0f),
this.gameObject.transform.position.y, Random.Range(10.0f, 30.0f));
        Invoke("spinnerAttack", 1.1f);
    }
    else
    {
        con.SetBool("saw1", false);
        con.SetBool("saw2", false);
        con.SetBool("idle", true);
        teleports = 0;
        attacking = false;
    }
}
```

```
    void stabAttack()
    {
        attacking = false;
        con.SetBool("stab", false);
    }

    void die()
    {

        Destroy(this.gameObject);
    }
}
```

OctoCannon.cs - Boss Cannon Behavior

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

public class octoCannon : MonoBehaviour
{
    public GameObject bullet;
    private float fireRate = 0.8f;
    private float myTime = 0;
    public GameObject bulletSpawner;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        //the cannons fire every 0.8 seconds, as mytime is added to, everytime is is over 0.8s then
a shot is fired and the shoot function is Invoked.
        myTime += Time.deltaTime;
        if(myTime > fireRate)
        {
            Invoke("shoot", 0.0f);
            myTime = 0.0f;
        }
```

```
}

/**/
/*
void shoot()

NAME



SYNOPSIS


DESCRIPTION

    Instantiate a bullet object to be spawned at a designated spot, append a rigidbody to handle physics, and add some force to it.

RETURNS



AUTHOR

    James P Giordano

DATE

    9/15/2020

*/
/**/

void shoot()
{
    GameObject tempBullet;
    tempBullet = Instantiate(bullet, bulletSpawner.transform.position, bullet.transform.rotation);

    Rigidbody tempRigidbody;
    tempRigidbody = tempBullet.GetComponent<Rigidbody>();

    tempRigidbody.AddForce(0, 0, -2150.0f);

    Destroy(tempBullet, 10.0f);
```

```
    }
}
```

COLLISION HANDLING

BACol.cs - Battle Axe Collision

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

public class BACol : MonoBehaviour
{
    public float damage = 12.0f;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }

    void OnCollisionEnter(Collision other)
    {
        //Debug.Log("Collided");
        //Thank to Unity's layering system, we can be almost certain that we have collided with the player, but it's nice to double check that we have.
        if(other.gameObject.tag == "Player")
        {
            other.gameObject.GetComponent<Player>().currHP -= damage;
        }
    }
}
```

bulletCol.cs - Player Bullet Collision

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

public class bulletCol : MonoBehaviour
{


    public float damage = 10.0f;

    public Material hitMat;
    public Material normalMat;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }

    /**/
    /*
    void OnCollisionEnter()

    NAME

        void OnCollisionEnter()

    SYNOPSIS


    DESCRIPTION

        Whenever a collision occurs involving the gameObject this script is attached to, this function is automatically called.
        We can get the other game object we have come into contact with, and perform actions accordingly. In this case, when we collide we an enemy
```

we look at the tag that is attached to the gameObject so we can access the correct script and remove the HP accordingly.

This script is only for player bullets, as the enemy bullets have a different prefab.

RETURNS

nothing.

AUTHOR

James P Giordano

DATE

8/4/2020

*/
/**/

```
void OnCollisionEnter(Collision other)
{
    //check the tag of the enemy, subtract HP from the acccording script. Then, destroy this object!
    if (other.gameObject.tag == "basic")
    {



        other.gameObject.GetComponent<basicWyrm>().health -= damage;

        Destroy(this.gameObject);
    }
    if (other.gameObject.tag == "BA")
    {


        other.gameObject.GetComponent<BAxeBehavior>().health -= damage;

        Destroy(this.gameObject);
    }
    if (other.gameObject.tag == "lazer")
    {
```

```
            other.gameObject.GetComponent<lazer>().health -= damage;

            Destroy(this.gameObject);
        }
        if (other.gameObject.tag == "magma")
        {


            other.gameObject.GetComponent<magma>().health -= damage;

            Destroy(this.gameObject);
        }
        if (other.gameObject.tag == "octoBoss")
        {


            other.gameObject.GetComponent<octoBoss>().health -= damage/2;

            Destroy(this.gameObject);
        }
    }

}
```

enemyBulletCol.cs - Enemy Bullet Collision

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

public class enemyBulletCol : MonoBehaviour
{
    public float damage = 10.0f;
    // Start is called before the first frame update
    void Start()
    {
        Invoke("die", 4.0f);
    }

    // Update is called once per frame
    void Update()
    {
```

```
}
/**/
/*
void OnCollisionEnter()

NAME

    void OnCollisionEnter()

SYNOPSIS


DESCRIPTION

    Whenever a collision occurs involving the gameObject this script is attached to, this
function is automatically called.
    We can get the other game object we have come into contact with, and perform actions
accordingly. In this case, when we collide we the player
    we look at the tag that is attached to the gameObject so we can access the correct script
and remove the HP accordingly.

    This script is only for enemy bullets, which thanks to Unity's layering system have been set
up to only be able to hit the player, but it is nice to check anyways.

RETURNS

      nothing.

AUTHOR

    James P Giordano

DATE

    8/4/2020

*/
/**/
void OnCollisionEnter(Collision other)
{

    if (other.gameObject.tag == "Player")
    {
```

```
            other.gameObject.GetComponent<Player>().currHP -= damage;
            Destroy(this.gameObject);
        }
    }

    void die()
    {
        Destroy(this.gameObject);
    }
}
```

lazerCol.cs - Lazer Wyrm Shot Collision

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

public class lazerCol : MonoBehaviour
{
    public float damage = 7.0f;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        transform.localScale += new Vector3(0, 0.15f, 0);
        transform.position += new Vector3(0, 0, -0.3f);
    }

    void OnCollisionEnter(Collision other)
    {
        if (other.gameObject.tag == "Player")
        {




            other.gameObject.GetComponent<Player>().currHP -= damage;
```

```
        }
    }

    void OnCollisionStay(Collision other)
    {
        if (other.gameObject.tag == "Player")
        {



            other.gameObject.GetComponent<Player>().currHP -= 0.5f;


        }
    }
}
```

magmaCol.cs - Magma Wyrm Shot Collision

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

public class magmaCol : MonoBehaviour
{


    public float damage = 15.0f;
    // Start is called before the first frame update
    void Start()
    {
        //if we don't hit the player we still need to worry about cleaning up the object.
        Invoke("die", 4.0f);
    }

    // Update is called once per frame
    void Update()
    {
        //a transform of position called every update creates the bullet like projectile behavior.
        transform.position += new Vector3(0, 0, -0.2f);
```

```
    }

    void OnCollisionEnter(Collision other)
    {

        if (other.gameObject.tag == "Player")
        {
            other.gameObject.GetComponent<Player>().currHP -= damage;
            Destroy(this.gameObject);
        }
    }

    void die()
    {
        Destroy(this.gameObject);
    }
}
```

RingExp.cs - Handle the collision of the explosion ring created by the rocket on death.

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

public class ringExp : MonoBehaviour
{
    public float damage = 25.0f;


    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }
```

```
    //when the ring from the explosion collides with an enemy, grab the attached script and
modify the health appropriately.
    void OnCollisionEnter(Collision other)
    {
        if (other.gameObject.tag == "basic")
        {
            other.gameObject.GetComponent<basicWyrm>().health -= damage;
            //Destroy(this.gameObject);
        }

        if (other.gameObject.tag == "BA")
        {



            other.gameObject.GetComponent<BAxeBehavior>().health -= damage;

            //Destroy(this.gameObject);
        }
        if (other.gameObject.tag == "lazer")
        {



            other.gameObject.GetComponent<lazer>().health -= damage;

            //Destroy(this.gameObject);
        }
        if (other.gameObject.tag == "magma")
        {



            other.gameObject.GetComponent<magma>().health -= damage;

            //Destroy(this.gameObject);
        }

    }
}
```

rocketCol.cs - Player Rocket Collision

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

public class rocketCol : MonoBehaviour
{
    public float damage = 50.0f;
    public GameObject explosion;
    public GameObject ringEXP;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }

    void OnCollisionEnter(Collision other)
    {
        if (other.gameObject.tag == "basic")
        {



            other.gameObject.GetComponent<basicWyrm>().health -= damage;

            rocketEX();
        }
        if (other.gameObject.tag == "BA")
        {
```

```
        other.gameObject.GetComponent<BAxeBehavior>().health -= damage;

        rocketEX();
    }
    if (other.gameObject.tag == "lazer")
    {



        other.gameObject.GetComponent<lazer>().health -= damage;

        rocketEX();
    }
    if (other.gameObject.tag == "magma")
    {



        other.gameObject.GetComponent<magma>().health -= damage;

        rocketEX();
    }
    if (other.gameObject.tag == "octoBoss")
    {



        other.gameObject.GetComponent<octoBoss>().health -= damage;

        rocketEX();
    }
}

/**/
/*
void rocketEX()

NAME
```

SYNOPSIS

DESCRIPTION

handles the explosion of the rocket by instantiating a particle explosion and an explosion ring at the moment the rocket is destroyed.

RETURNS

AUTHOR

James P Giordano

DATE

9/20/2020

*/
/**/

```
void rocketEX()
{

    GameObject exp;
    GameObject ringEX;
    exp = Instantiate(explosion, this.gameObject.transform.position,
this.gameObject.transform.rotation) as GameObject;
    ringEX = Instantiate(ringEXP, exp.transform.position, ringEXP.transform.rotation)
as GameObject;
    Destroy(this.gameObject);
    Destroy(exp, 1.0f);
    Destroy(ringEX, 1.0f);
}

void OnTriggerEnter(Collider other)
{
```

```
        if (other.gameObject.tag == "Player")
        {
            other.gameObject.GetComponent<Player>().currHP -= 5.0f;
        }
    }
}
```

specCol.cs - Player Special Attack Collision

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

public class specCol : MonoBehaviour
{

    public float damage = 20.0f;
    public Rigidbody rb;
    public Vector3 LastV = new Vector3();
    private float myTime;
    // Start is called before the first frame update
    void Start()
    {
        rb.detectCollisions = true;
    }

    // Update is called once per frame
    void Update()
    {
        myTime += Time.deltaTime;
        if (myTime > 2.0f)
        {
            transform.position += new Vector3(0, 0, .1f);
        }
    }



    void OnTriggerEnter(Collider other)
    {
        //rb.velocity = new Vector3(0,0,70.0f);
        if (other.gameObject.tag == "basic")
        {
```

```
        other.gameObject.GetComponent<basicWyrm>().health /= 2;


    }
    if (other.gameObject.tag == "BA")
    {



        other.gameObject.GetComponent<BAxeBehavior>().health /= 2;


    }
    if (other.gameObject.tag == "lazer")
    {



        other.gameObject.GetComponent<lazer>().health /= 2;


    }
    if (other.gameObject.tag == "magma")
    {



        other.gameObject.GetComponent<magma>().health /= 2;


    }
    if (other.gameObject.tag == "octoBoss")
    {


        other.gameObject.GetComponent<octoBoss>().health -= 30.0f;

        Destroy(this.gameObject);
    }
}
```

```
    void OnTriggerStay(Collider other)
    {
        if (other.gameObject.tag == "basic")
        {



            other.gameObject.GetComponent<basicWyrm>().health -= 0.6f;



        }
        if (other.gameObject.tag == "BA")
        {



            other.gameObject.GetComponent<BAxeBehavior>().health -= 0.6f;



        }
        if (other.gameObject.tag == "lazer")
        {



            other.gameObject.GetComponent<lazer>().health -= 0.6f;



        }
        if (other.gameObject.tag == "magma")
        {



            other.gameObject.GetComponent<magma>().health -= 0.6f;



        }
    }
}
```

SpinnerCol.cs - Boss SawBlade (spinner) Collision

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

public class SpinnerCol : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }

    void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.tag == "Player")
        {
            other.gameObject.GetComponent<Player>().currHP -= 20.0f;
        }
    }
}
```

tentCol.cs - Boss Tentacle Collision

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

public class tentCol : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
```

```
    }

    // Update is called once per frame
    void Update()
    {

    }

    void OnTriggerEnter(Collider other)
    {
        if(other.gameObject.tag == "Player")
        {
            other.gameObject.GetComponent<Player>().currHP -= 5.0f;
        }
    }
}
```

MENUING

backButton.cs - Move Back to Last Menu

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

public class backButton : MonoBehaviour, IPointerClickHandler
{
    public GameObject shopMenu;
    // Start is called before the first frame update
    void Start()
    {
```

```
    }

    // Update is called once per frame
    void Update()
    {

    }

    public void OnPointerClick(PointerEventData eventData)
    {
        //we need to hide the shop menu to preserve data, and change the scene.
        shopMenu.SetActive(false);
        SceneManager.LoadScene("mainMenu");
    }
}
```

backButtonDeath.cs - Move back to main menu after death

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

public class backButtonDeath : MonoBehaviour, IPointerClickHandler
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }

    public void OnPointerClick(PointerEventData eventData)
    {
```

```
        SceneManager.LoadScene("mainMenu");
    }
}
```

dontDestroy.cs - attaches to shop menu to not be destroyed

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

public class dontDestroy : MonoBehaviour
{
    // Start is called before the first frame update
    /*
     * This script exists to keep the shop menu active after leaving the scene, but an issue existed where many shop menus were being created.
     so we have to make sure never to destroy the original shop Menu but destroy the extras that are created.
     *
     */
    void Start()
    {
        GameObject[] gObjs;
        gObjs = GameObject.FindGameObjectsWithTag("shopMenu");
        if (gObjs.Length > 1)
        {
            Destroy(this.gameObject);
        }
        DontDestroyOnLoad(this.gameObject);
    }

    // Update is called once per frame
    void Update()
    {

    }
```

```
}
```

onLoad.cs - Show the existing shop menu once again

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


//this script exists to bring the shop menu back to activity when the shop scene is loaded.


public class onLoad : MonoBehaviour
{

    public GameObject toShow;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }

    void OnEnable()
    {
        SceneManager.sceneLoaded += OnSceneLoaded;
    }

    void OnDisable()
    {
```

```
        SceneManager.sceneLoaded -= OnSceneLoaded;
    }

    private void OnSceneLoaded(Scene scene, LoadSceneMode mode)
    {
        if (scene.name == "shop")
            {
            toShow.SetActive(true);
            }
    }
}
```

PlayButton.cs - Handle hovering and clicking of player button (replace image, and load scene)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.EventSystems;
using UnityEngine.SceneManagement;

public class playButton : MonoBehaviour, IPointerEnterHandler, IPointerExitHandler,
IPointerClickHandler
{
    public Texture highlight;
    public Texture original;

    private bool mouseover = false;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
      if (mouseover)
            {
            replaceImage();
```

```
        }

    if (!mouseover)
    {
        putImageBack();
    }
}

public void OnPointerEnter(PointerEventData eventData)
{
    mouseover = true;
    Debug.Log("Mouse enter");
}

public void OnPointerExit(PointerEventData eventData)
{
    mouseover = false;
    Debug.Log("Mouse exit");
}

public void OnPointerClick(PointerEventData eventData)
{
    SceneManager.LoadScene("shipSelect");
}

/**/
/*
void replaceImage()
```

NAME

SYNOPSIS

DESCRIPTION

switch the image of the a button when we hover over it.

RETURNS

AUTHOR

DATE

*/
/**/

```
void replaceImage()
{

	RawImage current = this.gameObject.GetComponent<RawImage>();
	current.texture = highlight;

}

/**/
/*
void putImageBack()
```

NAME

SYNOPSIS

DESCRIPTION

switch the image of a button when we stop hovering over it.

RETURNS

AUTHOR

DATE

```
    */
    /**/

    void putImageBack()
    {

        RawImage current = this.gameObject.GetComponent<RawImage>();
        current.texture = original;

    }

}
```

readyButton.cs - Handle moving from selectShip screen to runScreen

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

public class readyButton : MonoBehaviour, IPointerClickHandler
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }

    public void OnPointerClick(PointerEventData eventData)
    {
        //when we click the green check mark on the ship select screen, load the screen where the
game is played.
        SceneManager.LoadScene("runScreen");

    }
```

```
}
```

shipRotate.cs - rotates the ship object that can be seen in the ship select screen.

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

public class shipRotate : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        //rotates the ship in the select screen
        transform.Rotate(0, 0.1f, 0);
    }
}
```

shipSelectButton.cs - Handles UI on shipSelect screen when different ships are selected. There is only one ship so doesn't do much.

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

public class shipSelectButton : MonoBehaviour, IPointerClickHandler
{
    public string selected;
    public RectTransform speedBar;
    public RectTransform fireBar;
    public RectTransform utilityBar;
```

```
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }

    public void OnPointerClick(PointerEventData eventData)
    {
        //used for the ship selection button of the Black Bull. Sets the UI accordingly.
        selected = "BlackBull";
        speedBar.sizeDelta = new Vector2(200, speedBar.sizeDelta.y);
        fireBar.sizeDelta = new Vector2(400,  fireBar.sizeDelta.y);
        utilityBar.sizeDelta = new Vector2(80, utilityBar.sizeDelta.y);
    }
}
```

upgradeButton.cs - Handles the main menu upgrade button which takes you to the upgrade shop.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.EventSystems;
using UnityEngine.SceneManagement;

public class upgradeButton : MonoBehaviour, IPointerEnterHandler,
IPointerExitHandler, IPointerClickHandler
{
    public Texture highlight;
    public Texture original;

    private bool mouseover = false;
```

```
// Start is called before the first frame update
void Start()
{

}

// Update is called once per frame
void Update()
{
    if (mouseover)
    {
        replaceImage();
    }

    if (!mouseover)
    {
        putImageBack();
    }
}

public void OnPointerEnter(PointerEventData eventData)
{
    mouseover = true;
    Debug.Log("Mouse enter");
}

public void OnPointerExit(PointerEventData eventData)
{
    mouseover = false;
    Debug.Log("Mouse exit");
}

public void OnPointerClick(PointerEventData eventData)
{

    SceneManager.LoadScene("shop");
}
```

```
/**/
/*
void replaceImage()

NAME



SYNOPSIS


DESCRIPTION

    switch the image of the a button when we hover over it.

RETURNS



AUTHOR

    James P Giordano

DATE

    9/15/2020

*/
/**/
void replaceImage()
{

    RawImage current = this.gameObject.GetComponent<RawImage>();
    current.texture = highlight;

}
/**/
/*
void putImageBack()
```

```
    NAME



    SYNOPSIS


    DESCRIPTION

        switch the image of a button when we stop hovering over it.

    RETURNS



    AUTHOR

        James P Giordano

    DATE

        9/15/2020

    */
    /**/
    void putImageBack()
    {

        RawImage current = this.gameObject.GetComponent<RawImage>();
        current.texture = original;

    }

}
```

upgradeShopButton.cs - When we click a shop upgrade button, set related the meter accordingly.

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

```
using UnityEngine;
using UnityEngine.EventSystems;
using TMPro;

public class upgradeShopButton : MonoBehaviour, IPointerClickHandler
{
    public RectTransform barProgress;
    public TextMeshProUGUI textPercent;
    private float barSegments;
    private bool doneUpgrading = false;
    public int percentChange;
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }

    public void OnPointerClick(PointerEventData eventData)
    {
        //confirm the player has enough resources to make this purchase.
        if (doneUpgrading)
        {

        }
        barProgress.sizeDelta = new Vector2(barProgress.sizeDelta.x + 52,
barProgress.sizeDelta.y);

        //stop the bar from overflowing on the UI if this is the last upgrade.
        if (barProgress.sizeDelta.x > 590)
        {
            barProgress.sizeDelta = new Vector2(590, barProgress.sizeDelta.y);
            doneUpgrading = true;
        }

        //fill the bar based on the amount of purchases.
        barSegments = Mathf.Ceil((barProgress.sizeDelta.x - 35) / 52);
        if (percentChange > 0)
        {
```

```
            textPercent.text = "+" + (barSegments * percentChange).ToString() + "%";
        }
        else
        {
            textPercent.text = (barSegments * percentChange).ToString() + "%";
        }
    }


}
```

PLAYER SCRIPTS

Player.cs - Handles ship movement and all player related UI elements (health, alt fire, special fire)

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

public class Player : MonoBehaviour
{

    public float speed;
    private Rigidbody rb;
    public Vector3 movement;
    private Vector2 inputs;
    private float lastX;
    private float lastY;
    private bool first;
    public float shipWidth;
    public float shipLength;
    public RectTransform HPBar;
    public Text HPText;
    public GameObject deathCanvas;


    Vector2 screenSizeX;
    Vector2 screenSizeZ;
    public float MaxHP = 100.0f;
    public float currHP = 100.0f;
```

```
    // Start is called before the first frame update
    void Start()
    {
        rb = GetComponent<Rigidbody>();
        first = true;
        lastX = 0.0f;
        lastY = 0.0f;
       // shipWidth = transform.localScale.x / 2;
       // shipWidth = transform.localScale.z / 2;
        screenSizeX = new Vector2(Camera.main.aspect * Camera.main.orthographicSize +
shipWidth, Camera.main.orthographicSize);
        screenSizeZ = new Vector2(Camera.main.aspect * Camera.main.orthographicSize,
Camera.main.orthographicSize + shipLength);

    }

    // Update is called once per frame
    void Update()
    {
        //handle the player inputs on the displayed frames of the game.
        inputs = new Vector2(Input.GetAxis("Horizontal"), Input.GetAxis("Vertical"));
        //float hAxis = Input.GetAxis("Horizontal");
        //float vAxis = Input.GetAxis("Vertical");

        if (currHP > MaxHP)
        {
            currHP = MaxHP;
        }
        else if(currHP <= 0)
        {
            die();
        }
        //make sure the HP bar has the correct length.
        HPBar.sizeDelta = new Vector2(currHP * 2, HPBar.sizeDelta.y);
        HPText.text = currHP.ToString() + " / " + MaxHP.ToString();
    }


    void FixedUpdate()
    {

        float hAxis = inputs[0];
```

```
        float vAxis = inputs[1];
        //Handle the physics portion of the movement inside FixedUpdate, apply speed and rotation in the direction of our movement.
        movement = new Vector3(hAxis, 0, vAxis) * speed;
        Quaternion deltaRotation = Quaternion.Euler(new Vector3(movement[0],0,0) * 1.2f);
        rb.MoveRotation(rb.rotation * deltaRotation);
        rb.MovePosition(transform.position + movement);

        //the following if and else if blocks prevent us from leaving the screen region.
        if(transform.position.x < -screenSizeX.x)
        {
            transform.position = new Vector3(-screenSizeX.x, 0, transform.position.z);
        }

        else if(transform.position.x > screenSizeX.x)
        {
            transform.position = new Vector3(screenSizeX.x, 0 , transform.position.z);
        }

        if (transform.position.z < -screenSizeZ.y)
        {
            transform.position = new Vector3(transform.position.x, 0, -screenSizeZ.y);
        }

        else if (transform.position.z > screenSizeZ.y)
        {
            transform.position = new Vector3(transform.position.x, 0, screenSizeZ.y);
        }


        //reset our rotation slightly, this was intentionally added to give a more "hovership" effect but ended up creating more headaches than was worth.
        //need to work on this portion specifically
        if (!first && movement[0] != 0)
        {

            //the ship isn't stopped, needs to correct
            if (movement[0] != 0 && movement[0] > lastX)
            {
                rb.MoveRotation(rb.rotation * Quaternion.Euler(10,0,0));
            }
            if (movement[0] != 0 && movement[0] < lastX)
            {
                rb.MoveRotation(rb.rotation * Quaternion.Euler(-10, 0, 0));
```

```
        }
    }
    else
    {
        first = false;
    }
    lastX = movement[0];
    lastY = movement[2];
}

void die()
{
    deathCanvas.SetActive(true);
    Destroy(this.gameObject);
}

void OnCollisionEnter(Collision other)
{
    if (other.gameObject.tag == "BA")
    {
        currHP -= other.gameObject.GetComponent<BAxeBehavior>().damage;
    }
}
}
```

Shooting.cs - Handles player shooting of all types

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

public class shooting : MonoBehaviour
{
    public GameObject bulletEmitter;
    public GameObject bullet;
    public GameObject rocket;
    public GameObject specialShot;
    public GameObject altSpawner;
    public float bulletforce;
    public GameObject explosion;

    public float bulletlifeTime;
    public float rocketlifeTime;
    public float specialLifeTime;
    public float fireRate;
    private float nextFire = 0.5f;
    private float myTime = 0.0f;
    private float myAltTime = 0.0f;
    private float mySpecialTime = 15.0f;
    public float altCD = 6.5f;
    public float specialCD = 15.0f;
    public bool press;
    public float trigger;
    public float altTrigger;
    public bool SpecPress;
    public RectTransform ALTBar;
    public RectTransform SpecBar;
    private float altbarx;
    private float specbarx;
    public GameObject ringEXP;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
```

```
        //be ready for and receive player inputs.
        myTime = myTime + Time.deltaTime;
        myAltTime = myAltTime + Time.deltaTime;
        mySpecialTime += Time.deltaTime;
        press = Input.GetMouseButton(0);
        trigger = Input.GetAxis("Fire1");
        altTrigger = Input.GetAxis("Fire2");
        SpecPress = Input.GetButtonDown("Fire3");

        //based on player inputs and cooldowns, ensure that the proper functions are called.
        if( (press || (trigger != 0)) && myTime > nextFire)
        {
            nextFire = myTime + fireRate;
            shoot();
            nextFire = nextFire - myTime;
            myTime = 0.0f;
        }
        if( altTrigger != 0 && (myAltTime > altCD))
        {
            myAltTime = 0.0f;
            shootAlt();
        }
        if (SpecPress && (mySpecialTime > specialCD))
        {
            mySpecialTime = 0.0f;
            shootSpec();
        }

        //modify the bars representing the cooldowns of the alternate fire ability and the special
ability, if necessary
        altbarx = (myAltTime / altCD) * 200;
        if (altbarx > 200)
        {
            altbarx = 200;
        }
        specbarx = (mySpecialTime / specialCD) * 200;
        if(specbarx > 200)
        {
            specbarx = 200;
        }

        ALTBar.sizeDelta = new Vector2(altbarx, ALTBar.sizeDelta.y);
        SpecBar.sizeDelta = new Vector2(specbarx, SpecBar.sizeDelta.y);
```

```
}

void FixedUpdate()
{

}
/**/
/*
void shoot()

NAME



SYNOPSIS


DESCRIPTION


    Instantiate a temporary game object - essentially a clone, attach a rigidbody (for physics handling) and add a force vector to the attached rigidbody.
    this will effectively "fire" a bullet. At the end of the function we make sure the bullet will be destroyed.

RETURNS



AUTHOR

    James P Giordano

DATE

    8/27/2020

*/

/**/
```

```
    void shoot()
    {

        GameObject tempBullet;
        tempBullet = Instantiate(bullet, bulletEmitter.transform.position, bullet.transform.rotation)
as GameObject;



        Rigidbody tempRigidbody;
        tempRigidbody = tempBullet.GetComponent<Rigidbody>();

        tempRigidbody.AddForce(0, 0, bulletforce);

        Destroy(tempBullet, bulletlifeTime);


    }
    /**/
    /*
    void shootAlt()
```

NAME

SYNOPSIS

DESCRIPTION

Instantiate a temporary game object - essentially a clone, attach a rigidbody (for physics handling) and add a force vector to the attached rigidbody.

this will effectively "fire" a rocket. At the end of the function we make sure the rocket is destroyed, and start a coroutine the handle the rocket explosion.

RETURNS

AUTHOR

James P Giordano

DATE

```
        8/28/2020

    */
    /**/

    void shootAlt()
    {
        GameObject tempRocket;
        tempRocket = Instantiate(rocket, altSpawner.transform.position, rocket.transform.rotation)
as GameObject;

        Rigidbody tempRigidbody;
        tempRigidbody = tempRocket.GetComponent<Rigidbody>();

        tempRigidbody.AddForce(0, 0, 5000);
        Destroy(tempRigidbody, 3.0f);
        StartCoroutine(rocketEX(tempRocket, rocketlifeTime));
        Destroy(tempRocket, rocketlifeTime);
    }

    /**/
    /*
    IEnumerator rocketEX()

    NAME

        IEnumerator rocketEX(GameObject tempRocket, float toWait)

    SYNOPSIS

        temprocket -> the gameObject used to determine where the explosion will spawn
        toWait -> the number of seconds to wait before creating this explosion.

    DESCRIPTION

        sleep until just before we need to create the explosion, and then instantiate an explosion
and the explosion ring.

    RETURNS



    AUTHOR
```

James P Giordano

DATE

8/29/2020

*/
/**/

```
IEnumerator rocketEX(GameObject tempRocket, float toWait)
{
    yield return new WaitForSeconds(toWait - 0.05f);
    GameObject exp;
    GameObject ringEX;
    exp = Instantiate(explosion, tempRocket.transform.position, rocket.transform.rotation) as
GameObject;
    ringEX = Instantiate(ringEXP, exp.transform.position, ringEXP.transform.rotation) as
GameObject;
    Destroy(exp, 1.0f);
    Destroy(ringEX, 1.0f);
}
```

/**/
/*
void shootSpec()

NAME

SYNOPSIS

DESCRIPTION

Instantiate a temporary gameObject to form our special attack, start a coroutine to call the function to handle the physics, because we need to wait for the animation to finish.

RETURNS

AUTHOR

James P Giordano

DATE

9/2/2020

*/
/**/

```
void shootSpec()
{
    GameObject tempSpec;
    tempSpec = Instantiate(specialShot, altSpawner.transform.position,
specialShot.transform.rotation) as GameObject;
    StartCoroutine(pushSpec(tempSpec, 2.0f));

}

/**/
/*
IEnumerator pushSpec(GameObject tempSpec, float toWait)
```

NAME

SYNOPSIS

GameObject tempSpec - > the gameobject representing the attack that needs physics applied.
toWait - > the amount of time to wait before we start applying physics.

DESCRIPTION

while the animation is playing, sleep. Wake up, apply a rigidbody and addforce to the rigidbody. Destroy the temporary gameobject eventually.

RETURNS

AUTHOR

```
    James P Giordano

  DATE

    9/2/2020

  */
  /**/

  IEnumerator pushSpec(GameObject tempSpec, float toWait)
  {
    yield return new WaitForSeconds(toWait);
    Rigidbody tempRB;
    tempRB = tempSpec.GetComponent<Rigidbody>();

    tempRB.AddForce(0, 0, 3500);
    Destroy(tempSpec, rocketlifeTime);
  }
}
```

Gameplay Scripts

DropBehavior.cs - Handle the behavior of the randomly generated drop from an enemy. Is attached to the instantiated drop image.

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

public class dropBehavior : MonoBehaviour
{
  public TextMeshProUGUI plusText;
  public RawImage itemImg;
  private float myTime;
  bool fade = false;
  // Start is called before the first frame update
  void Start()
  {
    itemImg = this.gameObject.GetComponent<RawImage>();
    Invoke("fading", 0.5f);
  }
```

```
    // Update is called once per frame
    /*
     * This script is to be attached to the images that drop from basic enemies. It causes them to fade out at a regulated rate and also move up slowly. It creates a nice effect, and we destroy the image after it has fully
     faded out.
     */
    void Update()
    {
        if (fade && myTime > 0.1f)
        {
            Color current = plusText.color;
            current.a -= 0.08f;
            plusText.color = current;
            current = itemImg.color;
            current.a -= 0.08f;
            itemImg.color = current;
            myTime = 0.0f;
        }
        if (plusText.color.a == 0 && itemImg.color.a == 0)
        {
            Destroy(this.gameObject);
        }
        myTime += Time.deltaTime;
        this.gameObject.transform.position += new Vector3(0, 0, 0.01f);
    }

    void fading()
    {
        fade = true;
    }
}
```

relicSelection.cs - Handle the selection of a relic, apply the changes the relic should, Communicate with the wavecontroller to advance the game after a selection

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

```
public class relicSelection : MonoBehaviour, IPointerClickHandler
{
    public GameObject player;

    // Start is called before the first frame update
    void Start()
    {
        player = GameObject.Find("BlackBull");

    }

    // Update is called once per frame
    void Update()
    {

    }

    public void OnPointerClick(PointerEventData eventData)
    {
        //when we click the relic we choose, apply its effects based on the tag. Break and
deactivate the canvas that all the buttons belong to.
        //reset the booleans in the wave controller to allow more enemies to spawn.
        switch (this.gameObject.tag)
        {
            case ("speedRelic"):
                player.GetComponent<Player>().speed = player.GetComponent<Player>().speed *
1.3f;
                break;
            case ("cdrRelic"):
                player.GetComponent<shooting>().specialCD =
player.GetComponent<shooting>().specialCD * 0.6f;
                player.GetComponent<shooting>().altCD = player.GetComponent<shooting>().altCD
* 0.6f;
                break;
            case ("bulletRelic"):
                player.GetComponent<shooting>().fireRate =
player.GetComponent<shooting>().fireRate * 0.5f;
                break;
        }
        GameObject.Find("relicCanvas").SetActive(false);
        GameObject.Find("WaveController").GetComponent<Waves>().unlocked = false;
        GameObject.Find("WaveController").GetComponent<Waves>().deciding = false;
        Destroy(this.gameObject);
```

```
    }
}
```

selfDestruct.cs - Destroys this gameobject when it has no children left. We do this in a separate script because when the children are destroyed all scripts attached to them are as well.

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

public class selfDestruct : MonoBehaviour
{
    public int cc;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        //responsible for destroying the empty parent gameObject, when all of it's children (and thus their scripted behaviors) are destroyed.
        cc = transform.childCount;
        if (transform.childCount == 0)
        {
            Destroy(transform.gameObject);
        }
    }

}
```

Waves.cs - Handles all of the enemy instantiation, setting all the properties related to waves, sending new waves of enemies, and showing the relics.

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

```
public class Waves : MonoBehaviour
{
    public int wave = 1;
    public GameObject basicSpawner;
    public GameObject BaxeSpawner;

    private float spawnfreq = 4.0f;
    public GameObject basicEnemy;
    public GameObject BaxeEnemy;
    public GameObject lazerEnemy;
    public GameObject magmaEnemy;
    public GameObject octoBoss;
    private float spawnTime = 0.0f;
    public int points;
    public Image waveImg;
    public Text tempText;
    public bool unlocked = false;
    public bool deciding = false;
    public GameObject[] relics = new GameObject[3];
    public GameObject relicCanvas;


    // Start is called before the first frame update
    void Start()
    {
        setNum(1);
        sendWave(1);
    }

    // Update is called once per frame
    void Update()
    {
        //make sure we aren't spawning enemies too fast.
        spawnTime = spawnTime + Time.deltaTime;

        //once we have reached the spawn frequency timer, and we still have enemy points to
allocate, we can spawn an enemy.
        if(spawnTime > spawnfreq && points > 0)
        {
            //reset the spawn time.
            spawnTime = 0.0f;
            //decide what kind of enemy will spawn with a random number.
            int enemyTag = Random.Range(1, 5);
```

```
            StartCoroutine(spawnBasic(enemyTag, 1.0f));
        }

        //if we are out of enemy points to allocate, the next wave of enemies may spawn.
        if (points <= 0 && !enemiesAlive() && !unlocked)
        {
            wave += 1;
            setNum(wave);
            sendWave(wave);
        }

        //unlocked is a boolean that tells us the Invoke relicChoice after defeating a boss. deciding is a boolean that doesn't let relicChoice get called hundreds of times per second
        //as it is in the update loop.
        if (unlocked == true && !deciding && !enemiesAlive())
        {
            deciding = true;
            Invoke("relicChoice", 0.0f);
        }
    }

    /**/
    /*
    void sendWave(int waveNum)

    NAME



    SYNOPSIS

        waveNum - > the number of the wave to send. Certain numbers will correspond with in game actions.

    DESCRIPTION

        Allocate points to the wave will will be used to spawn enemies. if it is wave 5 we need to spawn a boss and nothing else. Unlock the relic choices when they will become applicable.

    RETURNS



    AUTHOR
```

```
    James P Giordano

DATE

    9/21/2020

*/
/**/

void sendWave(int waveNum)
{
    //points = 50;
    points += waveNum * 10;




    if (waveNum == 5)
    {
        points = 0;
        GameObject tempEnemy;
        tempEnemy = Instantiate(octoBoss, basicSpawner.transform.position,
octoBoss.transform.rotation) as GameObject;
        tempEnemy.transform.position = new Vector3(0, 0, 30.0f);
        unlocked = true;









    }
    }


/**/
/*
IEnumerator spawnBasic()

NAME
```

SYNOPSIS

int enemyValue -> a random number sent that determines the enemy to be spawned.
float toWait -> we don't want to spawn all the enemies at once, so we wait this long every time before sending a new enemy.

DESCRIPTION

wait for a set amount of time, and instantiate an enemy based on the enemy ID number provided to the function.

RETURNS

AUTHOR

James P Giordano

DATE

9/16/2020

```
*/
/**/
IEnumerator spawnBasic(int enemyValue, float toWait)
{
    yield return new WaitForSeconds(toWait);
    GameObject tempEnemy;

    //determine the type of enemy to spawn based on the random number rolled
    switch (enemyValue)
    {
        //rolling a 1 spawns a "basicWyrm"
        case 1:
            tempEnemy = Instantiate(basicEnemy, basicSpawner.transform.position, basicEnemy.transform.rotation) as GameObject;

            tempEnemy.transform.position = new Vector3(Random.Range(-90.0f, 90f), 0, 45.0f);
            /*
            Rigidbody tempRigidbody;

            tempRigidbody = tempEnemy.AddComponent<Rigidbody>();
```

```
            tempEnemy.GetComponent<Rigidbody>().useGravity = false;
            tempRigidbody.AddForce(0, 0, -875);



            Destroy(tempRigidbody, 2.0f);
            */
            points -= 2;
            break;

        //rolling a 2 spawns a "BA wyrm"
        case 2:
            tempEnemy = Instantiate(BaxeEnemy, BaxeSpawner.transform.position,
BaxeEnemy.transform.rotation) as GameObject;

            tempEnemy.transform.position = new Vector3(-90.0f, 0, Random.Range(10.0f, 20.0f));
            points -= 2;
            break;

        //rolling a 3 spawns a "lazer wyrm"
        case 3:
            tempEnemy = Instantiate(lazerEnemy, BaxeSpawner.transform.position,
lazerEnemy.transform.rotation) as GameObject;

            tempEnemy.transform.position = new Vector3(Random.Range(-90.0f, 90.0f), 0, 30.0f);
            points -= 2;
            break;
        //rolling a 4 spawns a "magma wyrm"
        case 4:
            tempEnemy = Instantiate(magmaEnemy, BaxeSpawner.transform.position,
magmaEnemy.transform.rotation) as GameObject;

            tempEnemy.transform.position = new Vector3(Random.Range(-90.0f, 90.0f), 0, 30.0f);
            points -= 3;
            break;
    }


}

/**/
/*
void setNum(in waveNum)
```

NAME

SYNOPSIS

int waveNum - > the number wave that it is, we will be displaying this.

DESCRIPTION

set the text that will alert the player what wave they are on. Invoke the fadeIn function to show the image to the player.

RETURNS

AUTHOR

James P Giordano

DATE

9/24/2020

```
*/
/**/
void setNum(int waveNum)
{
    tempText.text = "- Wave " + waveNum.ToString() + " -";
    Invoke("fadeIn", 0.01f);
}

/**/
/*
void fadeIn()
```

NAME

SYNOPSIS

DESCRIPTION

fade in the color of the text and the border image that shows the player the wave number. This function continually invokes itself so that there is a staggering in the speed of the fade in.

RETURNS

AUTHOR

James P Giordano

DATE

9/24/2020

*/
/**/

```
void fadeIn()
{

    if ((tempText.color.a < 1.0f) && (waveImg.color.a < 1.0f))
    {
        Color current = tempText.color;
        current.a += 0.1f;
        tempText.color = current;
        current = waveImg.color;
        current.a += 0.1f;
        waveImg.color = current;

        Invoke("fadeIn", 0.1f);
    }
    else
    {
        Invoke("fadeOut", 1.0f);
    }



}

/**/
/*
```

void fadeOut()

NAME

SYNOPSIS

DESCRIPTION

fade out the color of the text and the border image that shows the player the wave number. This function continually invokes itself so that there is a staggering in the speed of the fade out.

RETURNS

AUTHOR

James P Giordano

DATE

9/24/2020

*/

```
/**/
void fadeOut()
{
    if ((tempText.color.a > 0.0f) && (waveImg.color.a > 0.0f))
    {
        Color current = tempText.color;
        current.a -= 0.1f;
        tempText.color = current;
        current = waveImg.color;
        current.a -= 0.1f;
        waveImg.color = current;

        Invoke("fadeOut", 0.1f);
    }
}
```

/**/

```
/*
bool enemiesAlive()

NAME

    bool enemiesAlive()

SYNOPSIS


DESCRIPTION

    search for all of the possible enemy tags. If at any point an object exists with any tag that an enemy could have, this function returns true. returns false otherwise.

RETURNS



AUTHOR

    James P Giordano

DATE

    9/24/2020

*/
/**/

bool enemiesAlive()
{
    GameObject[] gos;
    gos = GameObject.FindGameObjectsWithTag("magma");
    if (gos.Length > 0)
        {
            return true;
        }
    gos = GameObject.FindGameObjectsWithTag("BA");
    if (gos.Length > 0)
    {
        return true;
    }
    gos = GameObject.FindGameObjectsWithTag("basic");
```

```
        if (gos.Length > 0)
        {
            return true;
        }
        gos = GameObject.FindGameObjectsWithTag("lazer");
        if (gos.Length > 0)
        {
            return true;
        }
        gos = GameObject.FindGameObjectsWithTag("octoBoss");
        if (gos.Length > 0)
        {
            return true;
        }
        return false;

    }

    /**/
    /*
    void relicChoice()

    NAME



    SYNOPSIS


    DESCRIPTION

        Instantiate the relics that will make up slots 1, 2 and 3 in the selection. Not currently
modular but could be made as such with the following steps:

        attach a GameObject[] array to this script, populate it with the GameObjects that will fill the
choices.
        randomly generate i using Random.Range(1, n+1); n is the number of possible rewards.
        Instatiate three gameobjects, with the parameter being relics[i], relics([i+1]% n), and
relics([i+2] % n).
        This will three consecutive choices from the array and will allow wrapping. Not the
objectively best choice, but a quick and easy pseudo-random fix
        without contiguous memory management.

    RETURNS
```

```
    AUTHOR

        James P Giordano

    DATE

        9/24/2020

    */
    /**/

    void relicChoice()
    {
        relicCanvas.SetActive(true);
        int i = 0;
        var tempImage = Instantiate(relics[i]) as GameObject;
        tempImage.transform.SetParent(relicCanvas.transform, false);
        tempImage.transform.position = new Vector3(-50.0f, 0, 0);

        tempImage = Instantiate(relics[i+1]) as GameObject;
        tempImage.transform.SetParent(relicCanvas.transform, false);
        tempImage.transform.position = new Vector3(0, 0, 0);

        tempImage = Instantiate(relics[i + 2]) as GameObject;
        tempImage.transform.SetParent(relicCanvas.transform, false);
        tempImage.transform.position = new Vector3(50.0f, 0, 0);

    }
}
```

Testing:

The nature of both the project and of the Unity Engine as a whole make testing and debugging the project relatively easy.

- If we need to determine if a specific function is being accessed, we can log something in the debugger (Debug.Log(“entered function quack.”))
- If we need to determine if a function is accomplishing a specific task, such as if bullets are actually subtracting health from an enemy, we can make the property to be changed public, and pause the game in the Unity editor when we suspect that the change we are

looking for has been made. We can then examine any objects in the scene and determine the values of any of their public properties.
- The game itself is highly visual, lots of actions (shooting, collisions, healthbars going down, explosions, enemy behavior, ship behavior) can be seen instantly. If they aren't to our liking we can see relatively quickly.

Using a mixture of these techniques I could test the game rigorously as changes were implemented. The logic of the game is not overly complex, but learning Unity and fitting the pieces together in a way that did not hinder further progression was the real task.

Summary:

This project had a lot of important takeaways for me.

- Game Engines are certainly powerful and have a lot of features, but they are not necessarily as scary as they seem. Like with most big projects, the easiest way to get going is to just start something, no matter how small it is. As you add components and the project begins to take more shape, it can be incredibly motivating and can also illuminate what the next steps will be.
- Designing UI, 3D models and textures, and any art takes a lot of time. I'm sure any artist worth their salt would have an easier time and a quicker time than I did designing most of the game, but I certainly feel that there is just as much if not more work to be done of the front end of games. This is especially true as the quality of the visuals goes up.
- Game logic does not necessarily need to be overly complex. The interminglings of the components and plugging them into each other is often much more complex than the actual code we are writing. Most often we are looking for simple changes to the game state caused by some trigger.
- Custom physics and animations are amongst the most complex systems to get right. It is very obvious when they are not behaving correctly. Animations are not as cut and dry as changing over from one to the other, as there are transition times. I did not use much simulated physics in the game, as I was looking for something more arcadey in nature, but while I was playing around with physics I quickly saw there was a lot of possibility for complexity with physics, particularly with getting certain behaviors just right.
- Debugging a game can be a tricky process. Try not to overlook even the obvious mistakes. There are many moving parts and it can take a while to pin something down if you are making assumptions.
- Try to build so that your code can be expanded upon, or changed easily. You may implement something and not like it. Make it easy to change so you can quickly see if you are going in the right direction. Don't be afraid to scrap a system if it just isn't providing what you thought it would.
- Remember that good things take a long time, and we are mostly used to seeing a finished product that has undergone many rounds of changes. Having worked for so long on this project, and having it still be so far away from exactly what I had in my mind's eye is okay with me. Iteration creates perfection.

- Game Engines put a lot of powerful tools right into your hands. You don't need to be a professional to create something of decent quality. You just have to be willing to learn, and willing to gain experience utilizing this powerful set of tools.
- Specifically for Unity, don't reinvent the wheel. People have been making games in Unity for a long time. There will always be many ways to accomplish a task, but that doesn't mean you shouldn't look some of them up before you try to implement it yourself.

I learned a great deal regarding game engines and game design during this project. I'm glad I got to dip my toes into Unity, and I won't shy away from using it in the future. I hope to dedicate some time to using it for some side projects in the future.

Resources:

These YouTube sources were all instrumental in helping me learn more about Unity.

https://www.youtube.com/user/Brackeys

https://www.youtube.com/channel/UC7j_49FiagPj30hnCdq8eOg

https://www.youtube.com/watch?v=_cCGBMmMOFw&list=PLFt_AvWsXl0fnA91TcmkRyhhixX9CO3Lw

Unity also provides a manual and scripting API which were again incredibly useful

https://docs.unity3d.com/ScriptReference/

https://docs.unity3d.com/Manual/index.html

Also part of the Unity website was the Unity Answers forum, a place where Q&A can be found on many types of questions.

https://answers.unity.com/index.html