CDG Oral Exam

By: Caleb C.

My Contribution: Enemies!

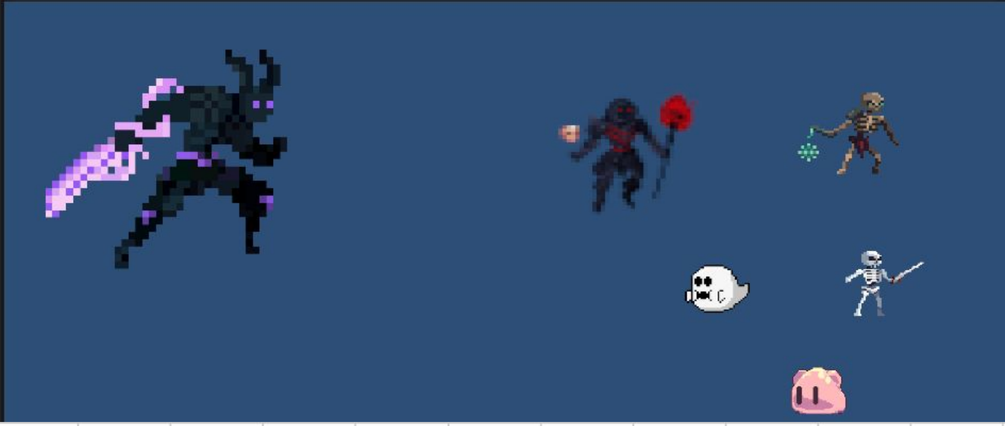
My job was to design the enemies that appear throughout the game.

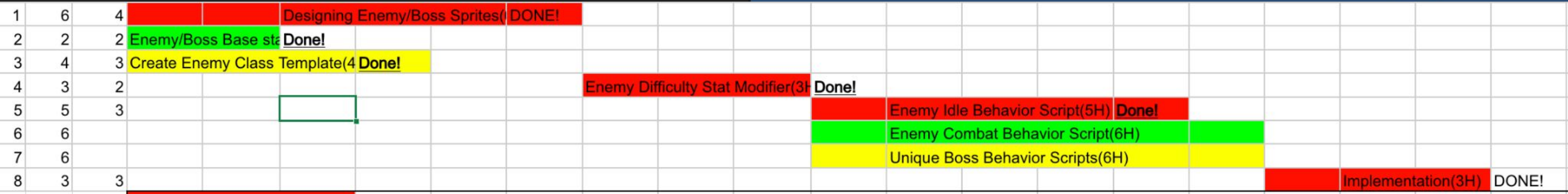
Including:

-Player interaction

-Spawning

-Simple AI





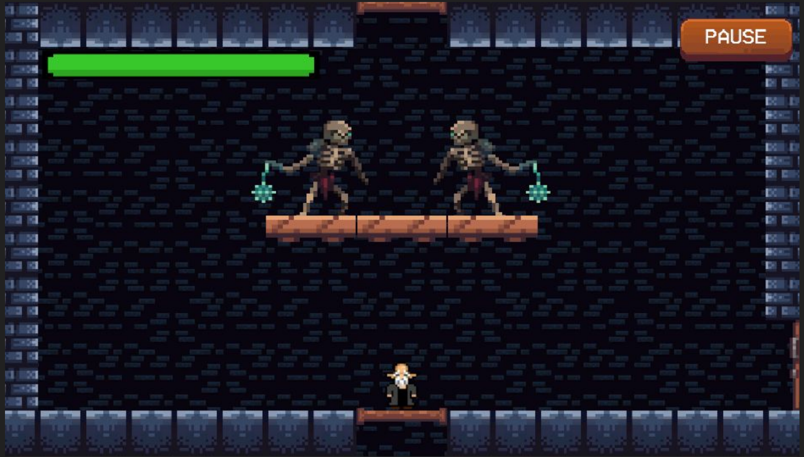
1	6	4	🟥	🟥	Designing Enemy/Boss Sprites(	🟥	🟥	DONE!												
2	2	2	Enemy/Boss Base sta	🟩	Done!															
3	4	3	Create Enemy Class Template(4	🟨	🟨	Done!														
4	3	2							Enemy Difficulty Stat Modifier(3H	🟥	🟥	Done!								
5	5	3										🟥	Enemy Idle Behavior Script(5H)	🟥	🟥	Done!					
6	6											🟩	Enemy Combat Behavior Script(6H)	🟩	🟩	🟩	🟩				
7	6											🟨	Unique Boss Behavior Scripts(6H)	🟨	🟨	🟨	🟨				
8	3	3																🟥	Implementation(3H)	🟥	DONE!

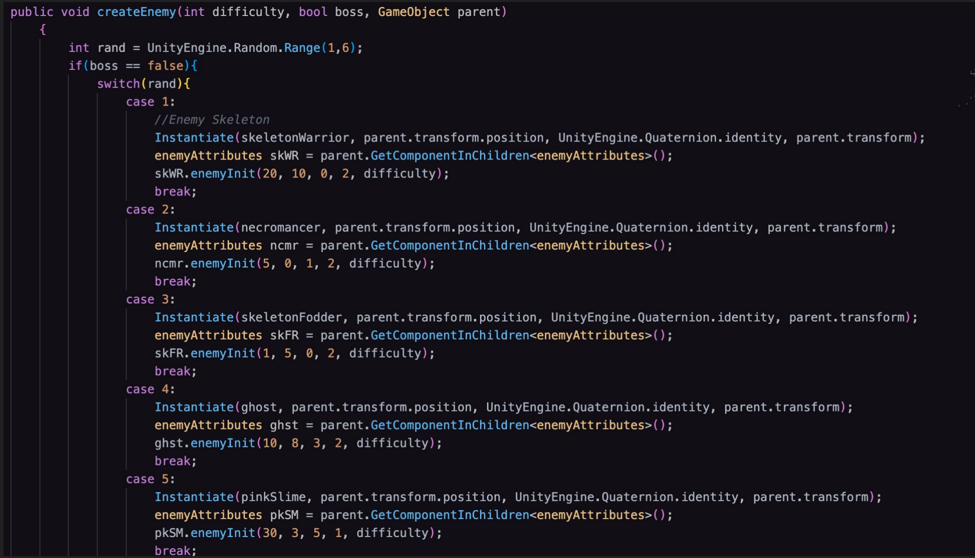
Contribution 2

How it works:

The level generator calls createEnemy()

To receive a random, unique enemy for placement in each level

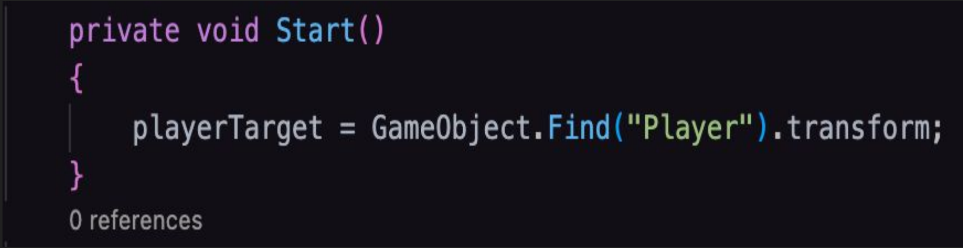




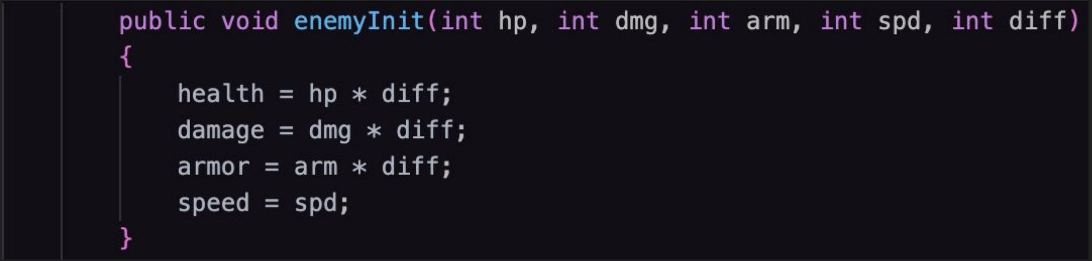
```
public void createEnemy(int difficulty, bool boss, GameObject parent)
    {
        int rand = UnityEngine.Random.Range(1,6);
        if(boss == false){
            switch(rand){
                case 1:
                    //Enemy Skeleton
                    Instantiate(skeletonWarrior, parent.transform.position, UnityEngine.Quaternion.identity, parent.transform);
                    enemyAttributes skWR = parent.GetComponentInChildren<enemyAttributes>();
                    skWR.enemyInit(20, 10, 0, 2, difficulty);
                    break;
                case 2:
                    Instantiate(necromancer, parent.transform.position, UnityEngine.Quaternion.identity, parent.transform);
                    enemyAttributes ncmr = parent.GetComponentInChildren<enemyAttributes>();
                    ncmr.enemyInit(5, 0, 1, 2, difficulty);
                    break;
                case 3:
                    Instantiate(skeletonFodder, parent.transform.position, UnityEngine.Quaternion.identity, parent.transform);
                    enemyAttributes skFR = parent.GetComponentInChildren<enemyAttributes>();
                    skFR.enemyInit(1, 5, 0, 2, difficulty);
                    break;
                case 4:
                    Instantiate(ghost, parent.transform.position, UnityEngine.Quaternion.identity, parent.transform);
                    enemyAttributes ghst = parent.GetComponentInChildren<enemyAttributes>();
                    ghst.enemyInit(10, 8, 3, 2, difficulty);
                    break;
                case 5:
                    Instantiate(pinkSlime, parent.transform.position, UnityEngine.Quaternion.identity, parent.transform);
                    enemyAttributes pkSM = parent.GetComponentInChildren<enemyAttributes>();
                    pkSM.enemyInit(30, 3, 5, 1, difficulty);
                    break;
```

Contribution 3

Every instance of a new enemy derives from my enemyBehavior() class, couples with enemyAttributes() which stores all of their hp, damage, and other important values. Each enemy starts by acquiring the target: “Player”



```
private void Start()
{
    playerTarget = GameObject.Find("Player").transform;
}
0 references
```



```
public void enemyInit(int hp, int dmg, int arm, int spd, int diff)
{
    health = hp * diff;
    damage = dmg * diff;
    armor = arm * diff;
    speed = spd;
}
```

Technical: Test Plan

Each test that will be design will be designed around these principals:

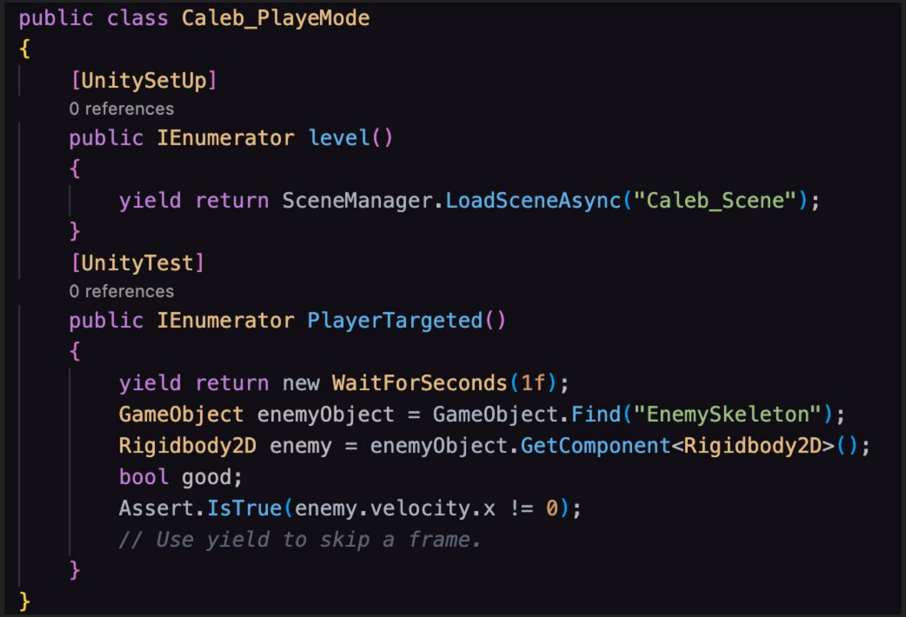
-Interaction

-Use by others

-Positioning

-Functionality

-Initialization



```
public class Caleb_PlayeMode
{
    [UnitySetUp]
    0 references
    public IEnumerator level()
    {
        yield return SceneManager.LoadSceneAsync("Caleb_Scene");
    }
    [UnityTest]
    0 references
    public IEnumerator PlayerTargeted()
    {
        yield return new WaitForSeconds(1f);
        GameObject enemyObject = GameObject.Find("EnemySkeleton");
        Rigidbody2D enemy = enemyObject.GetComponent<Rigidbody2D>();
        bool good;
        Assert.IsTrue(enemy.velocity.x != 0);
        // Use yield to skip a frame.
    }
}
```

Technical: Test Plan 2

For example:

Test - enemyAttributeCheck()

-Checks that any values that are standard to an enemy are assigned correctly and can be accessed correctly by asserting pre-determined stats

This is checking for Use by others, functionality, and initialization

Technical: Prefab

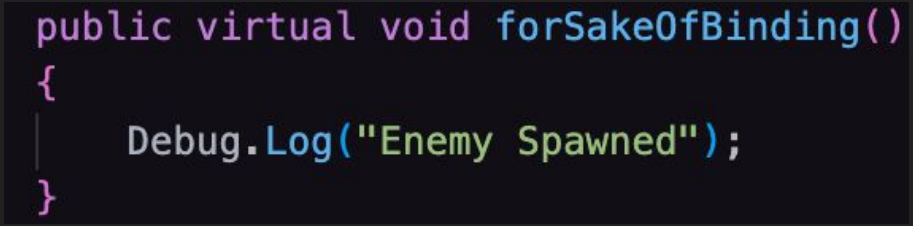
See .txt for additional information

Technical: Subclass/Superclass

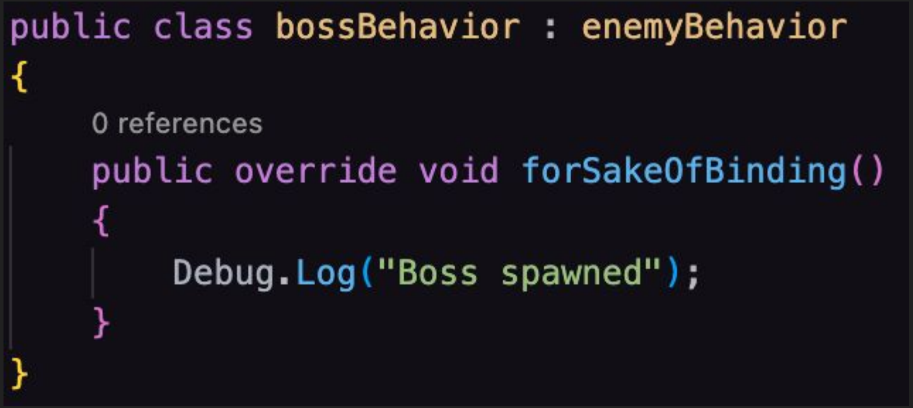
I chose to dynamically bind my enemyBehavior() class and my bossBehavior() class.

Functions in common:
forSakeOfBinding()

A debuglog() that identifies what type of enemy has been spawned in

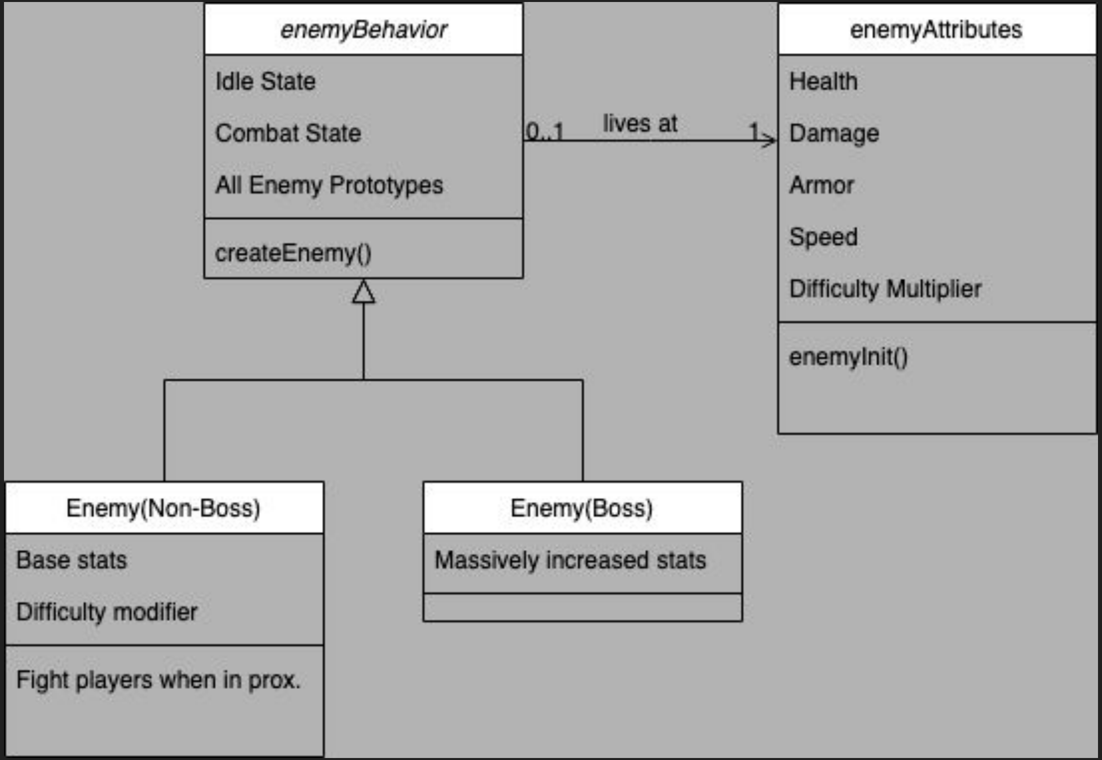


```
public virtual void forSakeOfBinding()
{
    Debug.Log("Enemy Spawned");
}
```



```
public class bossBehavior : enemyBehavior
{
    0 references
    public override void forSakeOfBinding()
    {
        Debug.Log("Boss spawned");
    }
}
```

Technical: Subclass/Superclass 2



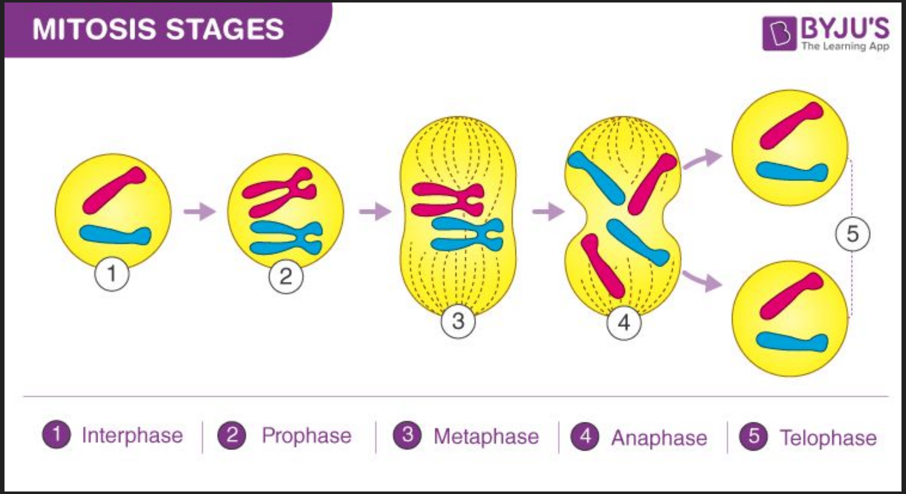
Technical: Prototype Pattern

I felt that the prototype pattern suited my code the best.

-Each enemy is a clone of a template

-Each clone is parented to an object that shows that it is, in fact, a clone

-Enemies can be continually made a virtually “unlimited” amount of times



Technical: Copyright Contribution

Meet: random christian elf boy!

As you can see he has the holy cross on his shield! You can’t copyright an entire religion! And besides, he’s just an elf boy with no distinguishable features! Hes 8 bit at best!

Please don’t murder me, Nintendo…