Name	Jason Culbertson	Team	Tappa Tappa Keyboard	TL	5	Date		Time	

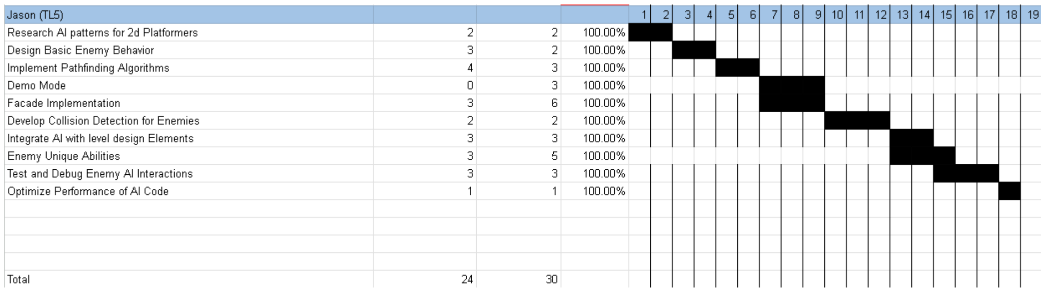
Fill in the underlined areas (and the boxes above), now but don't write on the remainder of this form.

Contribution: Briefly describe what your feature(s) is/are:

/10

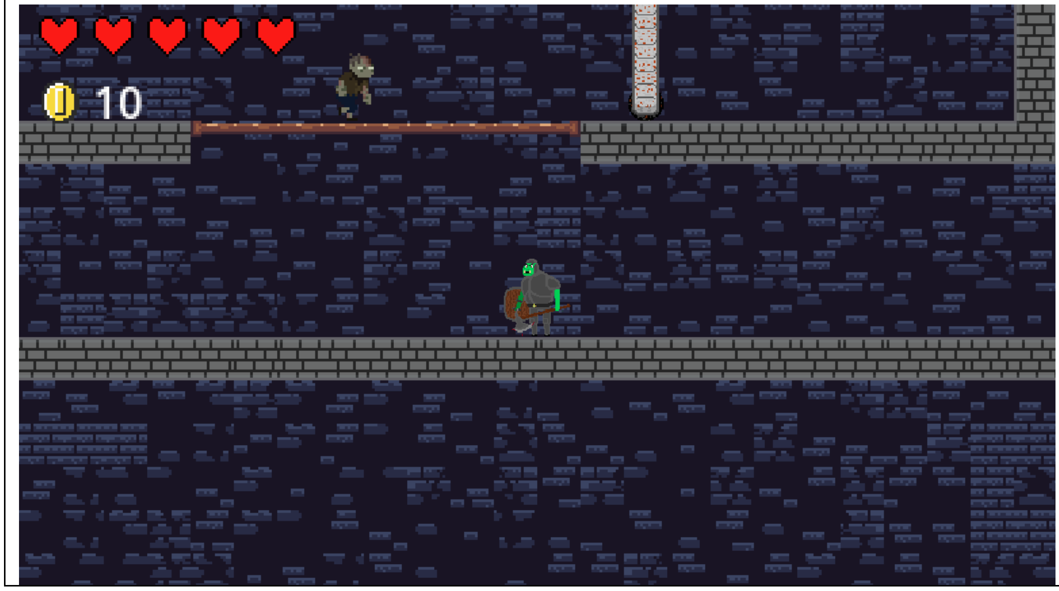
Enemies

Walk me through your Gantt chart. How long did this take? How long did you estimate it would take? What did you learn about your skill as an estimator?
The Project took me a total of 30 hours. I estimated that it would only take 24. I learned that it is hard to accurately gauge the amount of time a project like this would need. When I am doing projects by myself, I think it is easier. When working with a team, and especially on features that interact with each other, it is hard predict how long everything is going to take. I have gained more respect for good project estimators.



Jason (TL5)				1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Research AI patterns for 2d Platformers	2	2	100.00%	■	■																	
Design Basic Enemy Behavior	3	2	100.00%			■	■															
Implement Pathfinding Algorithms	4	3	100.00%					■	■													
Demo Mode	0	3	100.00%							■	■	■										
Facade Implementation	3	6	100.00%							■	■	■										
Develop Collision Detection for Enemies	2	2	100.00%										■	■	■							
Integrate AI with level design Elements	3	3	100.00%													■	■					
Enemy Unique Abilities	3	5	100.00%													■	■	■				
Test and Debug Enemy AI Interactions	3	3	100.00%															■	■	■		
Optimize Performance of AI Code	1	1	100.00%																		■	
Total	24	30																				

Run your game and point out places where your code is called and run. (I will cycle through asking you this question and the next one until you either run out of interesting things to talk about or it is clear that you have made an above average contribution.)

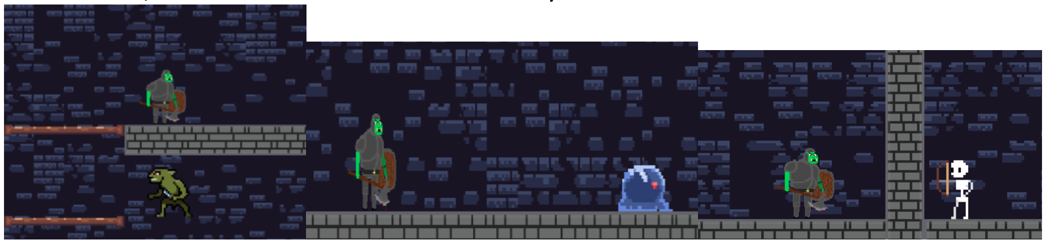


Show the C++/C# code that was run. Walk me through the methods called from the time it enters your section of code.

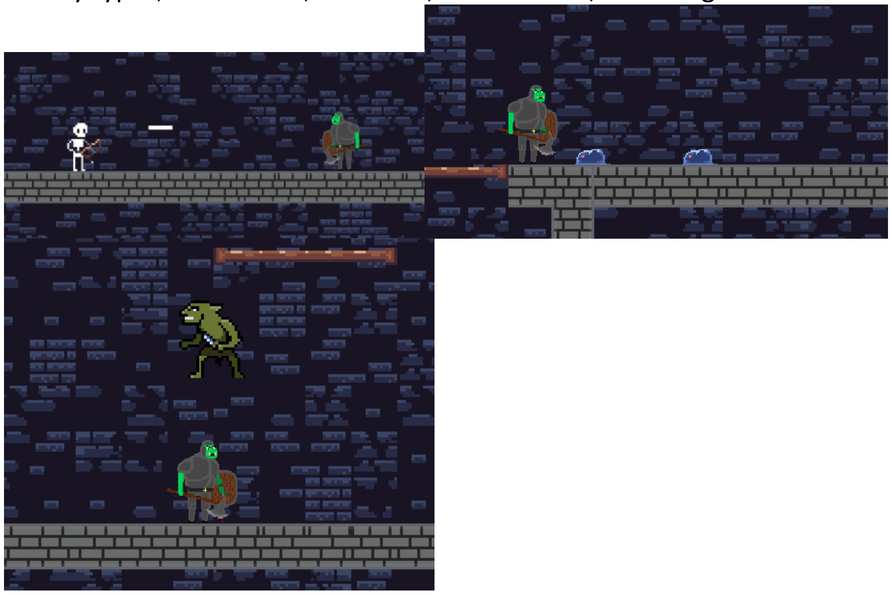
In this screenshot, we see our first enemy. This is my code.

```
private void SpawnEnemies(int health = 100, int damage = 25)
{

var rand = new Random();

// GD.Print("Spawning enemies...");

int enemyType;
foreach (var spawnPoint in spawns)
{
enemyType = rand.Next(4);
switch (enemyType)
{
case 0:
BaseEnemy enemy0 = (Zombie)ZombieScene.Instantiate();
enemy0.GlobalPosition = spawnPoint;
enemy0.Damage = damage;
enemy0.SetMaxHealth(health);
AddChild(enemy0);
enemy0.SetupEnemy();  // Custom setup method for enemies
break;
case 1:
BaseEnemy enemy1 = (Skeleton)SkeletonScene.Instantiate();
enemy1.GlobalPosition = spawnPoint;
enemy1.Damage = damage;
enemy1.SetMaxHealth(health);
AddChild(enemy1);
enemy1.SetupEnemy();  // Custom setup method for enemies
break;
case 2:
BaseEnemy enemy2 = (Slime)SlimeScene.Instantiate();
enemy2.GlobalPosition = spawnPoint;
enemy2.Damage = damage;
enemy2.SetMaxHealth(health);
AddChild(enemy2);
enemy2.SetupEnemy();  // Custom setup method for enemies
break;
case 3:
BaseEnemy enemy3 = (Goblin)GoblinScene.Instantiate();
enemy3.GlobalPosition = spawnPoint;
enemy3.Damage = damage;
enemy3.SetMaxHealth(health);
AddChild(enemy3);
enemy3.SetupEnemy();  // Custom setup method for enemies
break;
}
// GD.Print("Enemy instantiated.");
}
}
```

This is how an enemy gets spawned in, from the EnemyController. What kind of enemy spawns in is random, and is determined by this switch statement. A random enemy is determined, and then created as a BaseEnemy.



Here are some more enemies that we might see as we play the game. There are four enemy types, the zombie, the slime, the skeleton, and the goblin.



There are three enemies with unique abilities. The skeleton, who shoots an arrow. The slime, who splits in half, and the goblin, who jumps in the air.

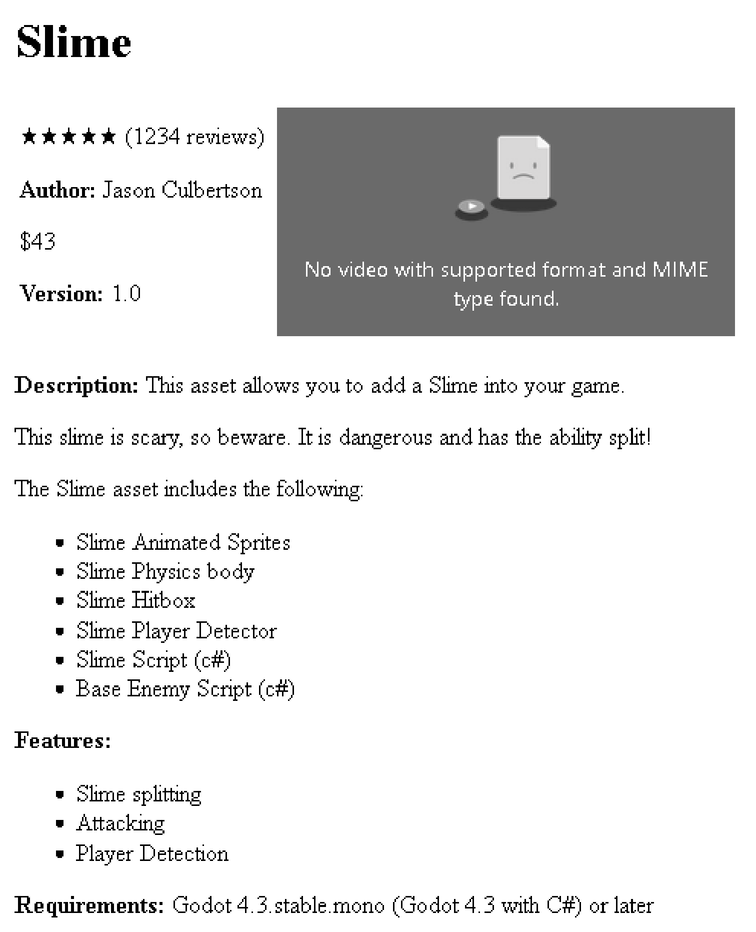
```
public void SpawnSlime()
{

if (SplitsRemaining <= 0)
{
return;
}

Slime enemy2 = (Slime)SlimeScene.Instantiate();
enemy2.GlobalPosition = this.GlobalPosition + new Vector2(-20,-5);
enemy2.Scale = new Vector2(enemy2.Scale.X/2,enemy2.Scale.Y/2);
enemy2.SplitsRemaining = SplitsRemaining - 1;
enemy2.SetMaxHealth(MaxHealth / 2);
AddSibling(enemy2);
enemy2.SetupEnemy();
GD.Print("Split 1");

Slime enemy3 = (Slime)SlimeScene.Instantiate();
enemy3.GlobalPosition = this.GlobalPosition + new Vector2(20, -5);
enemy3.SetMaxHealth(MaxHealth / 2);
enemy3.Scale = new Vector2(enemy3.Scale.X / 2, enemy3.Scale.Y / 2);
enemy3.SplitsRemaining = SplitsRemaining - 1;
AddSibling(enemy3);
enemy3.SetupEnemy();

HasSlimed = true;
GD.Print("Splitting!");
}
public override void EnemyAttack()
{
if (!CanShoot)
{
return;
}
Cooldown.Start();
CanShoot = false;

Arrow arrow = (Arrow)ArrowScene.Instantiate();
arrow.direction.X = base.direction.X;
arrow.GlobalPosition = this.GlobalPosition;

AddSibling(arrow);
GD.Print("Skeleton attacks!");
base.EnemyAttack();
}
```

Here is the code that runs to make that happen. It starts with dynamically binding the functions in the Base enemy, so that the enemies can do different things when attacking or dying.

Technical: Walk me through your test plan. Give an example where a test case later found a bug in your code by things a teammate added later. (Or explain why you chose a test case specifically because you wanted to ensure that a teammate would know if they broke your code.) 1. Test Player and Zombie 2. Test Fireball and Zombie	/4

3. Test Attack and Zombie	/3
4. Test Enemy stays on Ledge	
5. Test Enemy turns around on wall	
6. Test Enemy Floor Collision	
7. Test Skeleton Shoots Arrow	/3
8. Test Slime splits	
9. Test Goblin jumps	
10. Test Zombie Death	
11. Test Slime Death	
12. Test Skeleton Death	
13. Test Goblin Death	/4
14. Test Zombie Attack	
15. Test Skeleton Attack	
16. Test Slime Attack	
17. Test Goblin Attack	
18. Test Zombie Idle	/4
19. Test Skeleton Idle	
20. Test Slime Idle	
21. Test Goblin Idle	
22. Test Skeleton Detection	
23. Test Goblin Detection	
24. Test Slime Detection	
25. Test Zombie Detection	
26. Test Enemy Gravity	

27. Test Enemy Gravity 28. Test Enemy Gravity 29. Test Enemy Gravity 30. Test Enemy Attribute Dynamic Binding An example of a test that I added specifically to check if something would break, is enemy detection. This was because the enemies needed to be able to detect the player before they could utilize much else in their functionality. If my teammate was to change anything in the way the player works, it could also change how the enemies detect the player. ____ Pick a Prefab you have created that is documented well in a separate readme file. (I will point to several places in your code documentation and ask) What question where you trying to answer here? Who do you anticipate would be asking that question? What other questions might this person need the answers to? Prefab Name: Slime	



Slime

★★★★★ (1234 reviews)

Author: Jason Culbertson

$43

Version: 1.0

No video with supported format and MIME type found.

Description: This asset allows you to add a Slime into your game.

This slime is scary, so beware. It is dangerous and has the ability split!

The Slime asset includes the following:

- Slime Animated Sprites
- Slime Physics body
- Slime Hitbox
- Slime Player Detector
- Slime Script (c#)
- Base Enemy Script (c#)

Features:

- Slime splitting
- Attacking
- Player Detection

Requirements: Godot 4.3.stable.mono (Godot 4.3 with C#) or later

Show me a class in your code where there could be either static or dynamic binding. Write some mock code on this paper showing how you would set the static type and dynamic type of a variable.
Super Class: BaseEnemy
Sub Class: Zombie
Virtual Function: SetupEnemy()

```
BaseEnemy enemy0 = (Zombie)ZombieScene.Instantiate();
```

Choose a dynamically bound method. What method gets called now?

Enemy0.SetupEnemy would call:

```
public override void SetupEnemy()
{
base.Damage = 25;
base.Health = 100;
base.Speed = 25;
GD.Print("Zombie setup complete.");

}
```

Change the dynamic type. What method gets called now?

```
public virtual void SetupEnemy()
{
GD.Print("BaseEnemy setup.");
}
```

Pick a statically bound method. Which one would be called in each of the two previous cases?

A statically bound method like

```
private void OnArea2DBodyEntered(Node2D body)
{
if (body is Controller && !IsDead){
EnemyAttack();
}
}
```

Would always be called in either case because it is statically bound.

Show me an example of reuse in your code where you violate copyright law.
How does it violate copyright?____
What did you have to do to integrate it with the code you wrote? What are the legal implications if you market your code with the re-used portion? Use fair use argue that you can use this anyway.

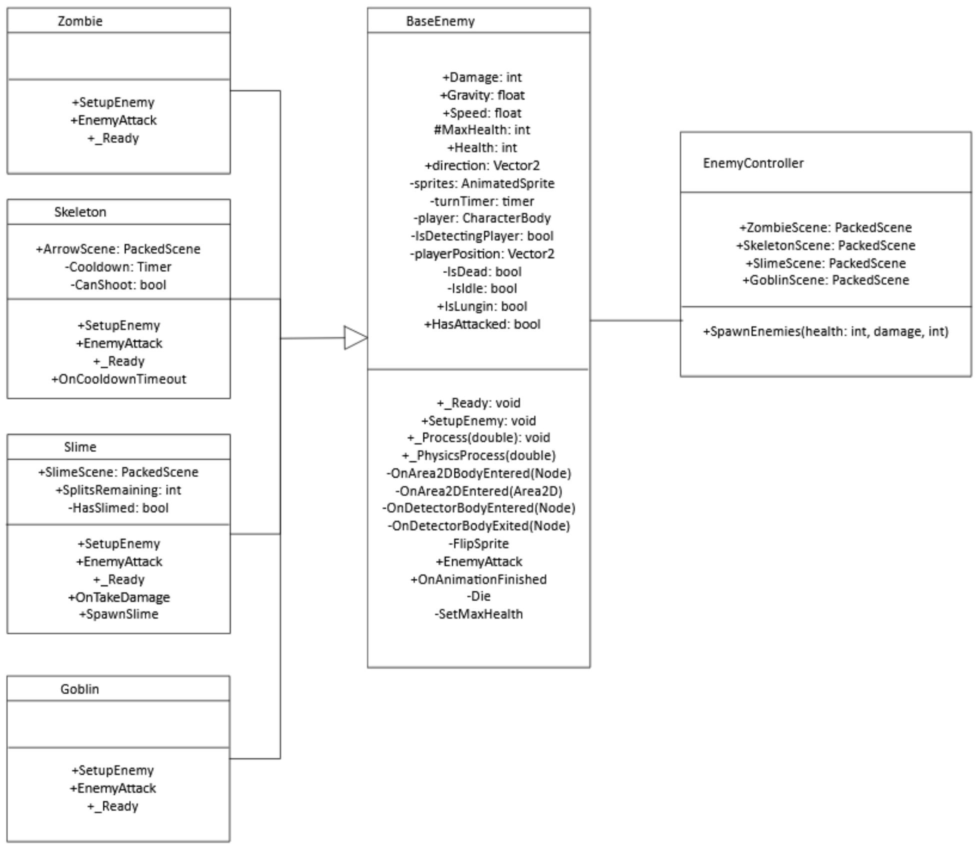
This is the sprite for the arrow that my skeleton shoots. It violates copyright because it was directly stolen from a painting by David Teniers the Younger, titled “Peasants at Archery”. The legal implication of marketing my code with this reused portion is being sued to remove the asset, and I may even have to pay money if I have profited from the use of the asset. My argument for fair use is this: David Teniers the Younger died in 1690, meaning that this painting has since entered the public domain.

4. One big or two small, well-chosen patterns.
Small Patterns = {Singleton, Private Class Data}
Which patterns did you choose?

1. Facade Design Pattern

Why did you choose each pattern? (Justify your use of it).

I used this pattern because the enemies needed to be modular, and that meant they needed an interface. One that would allow my teammate to feed it the spawns and spawn the enemies. Facades are just a higher-level interface for a complex system that facilitates actions without all of the depth. All that my teammates needed to do was spawn enemies with a specific health and damage, and that is what the enemy controller provides.



Would something else have worked as well or better than this pattern? When would be a bad time to use this pattern?

I think that there are other patterns that may have worked better for enemies that have more depth, since this pattern would require a lot of overriding and little overlap. A bad time to use this pattern would be if there is not much complexity in the system, since making a facade interface for it would just bloat the code.