Files worked on:

- ItemCollisionHandler (excluding line 41 and any other refactoring done for the objects)
- PlayerProjectileCollisionHandler(excluding most of line 67)
- EnemyProjectileCollisionHandler(excluding line 48)
- Lines 16-19 in PlayerCollisionHandler
- ArrowImpactProjectile
- Lines 77-81 in ArrowProjectile
- Lines 131-138 in PlayerProjectileSpriteFactory
- Minimal updates to the Update method of each projectile (just changing lifeSpans to not delete the object)
- Lines 96-103 in SwordBeamProjectile (I just copy pasted the block of the method from earlier code)
- Fairy (updated movement to be random. Need to delete lines 46-62)

KT's Review
Author: KT Goldstein
Date: 3/13/2021
Sprint #3
Files being reviewed: Listed above
Author: Jaci

ItemCollisionHandler (excluding line 41 and any other refactoring done for the objects)
Quality: Looks good to me, only thing I'm not certain of is the use of the more specific Directions, like UpLeft, UpRight, DownLeft, DownRight—I thought the ICollisionHandlers would only be passed one of the main directions, but maybe it's different in this case.
Also, one small thought, but not for this sprint— I'm wondering if IItem should be refactored at some point to have an IsCollectable property or something so that we don't need longer conditions like in Line 19, but there's no way to avoid this for now.
Readability: Good, I think it was a good choice to have the Line 45 assignment because it makes the following code easier to read, especially since it has more cases.

PlayerProjectileCollisionHandler(excluding most of line 67)
Quality: Good cohesion/separation of responsibilities for having the CurrentPlayerProjectile spawn its own explosion; I wonder if in the future (perhaps not for this Sprint) we could have a single SpawnDeathAnimation() method for projectiles, but this might not be worth it.
Readability: descriptive method and variable names, looks good

EnemyProjectileCollisionHandler(excluding line 48)
Quality: Simple, I have nothing else to add.
Readability: Again, good choice with adding the simple variable assignment to make it more readable (Line 25).

Lines 16-19 in PlayerCollisionHandler
Quality: Good that the player handles being damaged in its own class; higher cohesion.
Readability: Simple and therefore readable.

ArrowImpactProjectile & Lines 77-81 in ArrowProjectile
Quality: Simple, good separation of the arrow and its impact animation.
Readability: Simple, good variable names, easy to read.

Update method of each projectile (just changing lifeSpans to not delete the object)
Quality: Good design choice, works well with collisions.
Readability: Not really applicable here, but it does happen to make the rest of the code easier to read.

Lines 96-103 in SwordBeamProjectile
Quality: Good that SpawnSwordExplosion() is its own method.
Readability:

Fairy
Quality: Good, getting rid of the lines you mentioned helps make it even simpler as well. In my opinion the magic numbers don't make it harder to read, either, since the logic is so simple.
Readability: Neat, simple, and readable.

Summary:
Number of minutes taken to complete review: ~60
Overall: Nice quality code. A small suggestion I have in general is to delete the comments before we submit, and have the empty methods be on a single line e.g.:

```
public void HandleEnemyCollision(IEnemy enemy, Direction direction) { }
```

(I'll have to do this with my code as well). A similar suggestion would be to eliminate any extra whitespace, but again this is very picky and doesn't have to do with the code itself. Also not quite related to the code itself, but the comments were extremely helpful throughout this Sprint to help with understanding logic even faster as I was refactoring/fixing bugs related to object/block collisions around your code. Overall, simple and readable, and easy to understand.

Simon's Review
Author: Simon Kirksey
Sprint #3
Code Author: Jaci Taylor

Collisions\ItemCollisionHandler.cs

Refactoring: Are the items in HandlePlayerProjectileCollision supposed to die on any projectile impact or only with a boomerang? For HandleObjectCollision, you don't need all of the cases in your switch statement; for simplicity, the direction passed will only ever be Up, Down, Left, or Right. (The other four are used by other non-collision methods that use the Direction enum)

Readability: For empty and single line methods, I prefer them to be shortened to a literal single line, like:
public void MethodName() { }
or
public void MethodName() => statement;
Also, now that we've implemented the methods, I think the comments can be removed as your code is solid and self-documenting.

PlayerProjectileCollisionHandler.cs

Refactoring: Line 67 should be split up. I think it'd be best to either make it have intermediary boolean values or add some other if/else statements (it'd be fine to add more nesting here.

Readability: Mostly the same as for the item collision handler. Comments, shorten line counts for methods, and get rid of redundant white space (line 6 is fine as it separates a property and the singleton code, but either the newline on line 9 or line 10 should be removed). When you cast an IObject as a specific class and only use it for a single line, it can be compacted; here's an example of lines 53 + 54:
IPlayer player = (CurrentPlayerProjectile as BoomerangProjectile).link;

Collisions/EnemyProjectileCollisionHandler.cs

Refactoring: Since we only want a reaction when the enemy is a Goriya and the boomerangs goriya, lines 28 & 26 can have their conditionals be combined.

Readability: Same comments as the other collision handlers about white space, comments, and reducing line counts where convenient.

Players/PlayerProjectiles/ArrowImpactProjectile.cs

Refactoring: width and height should be assigned the values of the sprites dimensions in the constructor. They can also be made readonly since they're never changed. You can remove all references to ArrowSkinType since that doesn't affect its behavior at all.

Readability: No issues with white space, solid job!

Players/PlayerProjectiles/ArrowProjectile.cs
Players/PlayerProjectiles/SwordBeamProjectile.cs

Refactoring: To avoid the need for these to be cast from the IPlayerProjectile interface when their deaths occur, I think a refactoring of the IPlayerProjectile interface to include a public void Death() method would be a good idea. Furthermore, the CollisionHandlers would only need to set Alive to false and the LegendOfZeldaDungeon class can call .Death() on every IPlayerProjectile that has IsAlive == false. Since this is a decent sized refactor, it should probably be held off until after the Sprint 3 submission.

Readability: No issues here!

Players\PlayerProjectils\PlayerProjectilsSprites\PlayerProjectileSpriteFactory.cs

Refactoring: Since ArrowSkinType isn't used, the method CreateArrowDeathSprite doesn't need any parameters passed to it.

Readability: xOffset and yOffset should use minimal magic numbers. This could be improved by following the formulas used in the other methods to get to the specific "box" in the atlas.

Items\Fairy.cs

Refactoring: Agree with the deletion of lines 46-62. switchDirection doesn't need to be a field, having it declared in after the check on timer == 20 (line 64) should be fine. Might want to call .Normalize() on FairyDirection after setting its value, otherwise the Fairy moves faster diagonally. Best way of doing this would be changing the FairyDirection set method to be { fairyDirection = value; fairyDirection.Normalize(); } where fairyDirection is a private field never directly modified (this would also require the get method to change).

Readability: Solid!

Recap

Summary

Solid coding overall! A lot of my suggestions are heavily based on opinions (which I have a lot of). The only comment that'd majorly change functionality would be refactoring IPlayerProjectile to have a Death() method.

Questions

What's your opinion on my opinion based suggestions?

Time Spent

100 minutes