Players -> LinkPlayer-> LinkStates

- **LinkChargeState.cs**
- **LinkSpin.cs**
- **LinkUsingItemRight.cs**
  - Charge()
- **LinkUsingItemLeft.cs**
  - Charge()
- **LinkUsingItemUp.cs**
  - Charge()
- **LinkUsingItemDown.cs**
  - Charge()

Players -> LinkPlayer

- **LinkPlayer.cs**
  - Charge()
  - GrowCollider()
  - ShrinkCollider()

Enemies -> EnemyTypes

- **Dodongo.cs**

Enemies -> EnemyTypes -> DodongoStatePattern

- **CheckerboardAttack.cs**
- **CircleAttack.cs**
- **CircleGapAttack.cs**
- **HelicopterAttack.cs**
- **HomingCirceAttack.cs**
- **IDodongoState.cs**
- **PhaseOneIdle.cs**
- **PhaseTwoIdle.cs**
- **PhaseThreeIdle.cs**
- **WallAttack.cs**

—

KT's Review
Author: KT Goldstein
Date: 4/20/2021
Sprint #5
Files being reviewed: Listed above
Author: Alan

LinkStates & LinkPlayer
Quality: Looks great. Only comment I have is that in Charge() you might want to create a LinkPlayer GetStateCharge() method like Simon and I did when we worked on Link back in Sprint 2, just for purpose of single point of control and consistency. Same idea for Update() in LinkChargeState. But this isn't really a big thing at all so it's up to you. (If you're not sure what I mean, look at public void Die() in any of the LinkState classes.)

One thing that I think would be a good idea to change though is changing "direction" (in LinkSpin, and LinkChargeState constructor) from an int to an enum Direction that we have defined in LoZHelpers, since in LinkSpin you could use a Direction for a switch case just as well as an int, and that would help with readability probably too.
Quick example of what I'm trying to say:

```
Direction test = Direction.Left;
 switch (test)
 {
     case Direction.Down:
         linkPlayer.SetState(linkPlayer.GetStateStandingDown());
         break;
     case Direction.Right:
         linkPlayer.SetState(linkPlayer.GetStateStandingRight());
         break;
     case Direction.Up:
          linkPlayer.SetState(linkPlayer.GetStateStandingUp());
          break;
     case Direction.Left:
          linkPlayer.SetState(linkPlayer.GetStateStandingLeft());
          break;
     default:
          break;
 }
```

I'm saying a lot of general suggestions but overall this was really well done, and the result looks amazing. The GrowCollider() and ShrinkCollider() are really clever too.

Readability: Looks good overall, just that small thing with the Direction enum replacing the int to help with readability. Could probably also replace the 1f multiplication in line 117 with just a (float) cast.

Dodongo.cs

Quality: Awesome job with the state changes, and with getting him to move around. I'm wondering if we should put the directional floats either as constants or as enums. It might be good to get rid of the magic numbers somehow (or perhaps instead to use the LoZHelpers.Scale() method if necessary).

Readability: Looks great, great variable names and method names. I guess the public Invincible property is a little hard to read with the ternary statements but I do think it's worth it to keep it compact.

DodongoStatePattern

Quality: Awesome job with all the math for the attacks. There's a lot of raw math with some magic numbers but I think it's necessary here (or at least, I can't think of a better way to handle them/remove them, unless some of the numbers are the same as the directional floats in Dodongo.cs in which case they could maybe be moved to LoZHelpers).

Readability: Great overall. I'd maybe change the private GameStateMachine "g" to be called "game" instead, I remember it taking me a minute to understand the game is what was being passed into the constructors earlier. Not a huge deal though. You could also maybe change the "int r" variable used as a random number generator seed to be something like "randomSeed" or something but since it's only used in three lines and they're all right next to each other I think that one's probably readable enough already, but just something to consider.

Summary

Number of minutes taken to complete review: ~90 minutes

Overall comments: Great, high-quality code. The whole implementation of the spin attack was really well done and it looks awesome in-game. The Dodongo attacks are very clever too, and with the way you set it up it's easy to add more attacks (in terms of hypothetical code maintainability).

Yonace's Code Review
Sprint 5
4/21/2021

Approx time: 20 mins

Dodongo.cs && Enemies -> EnemyTypes -> DodongoStatePattern

Quality - No complaints quality is good. Reminder, there are lots of commented out code that can be deleted.
Readability - Readability is good, I can easily understand what each state does and its purpose.

All cs files from Players -> LinkPlayer-> LinkStates && LinkPlayer.cs

Quality - Again no complaints, my only question is if it is necessary to have a charge method in linkusingitem.cs files? (it probably is, I just don't know why?)

Readability - Great! Same Dodongo.cs files, code is clear and understandable.