Week 11 Journal of frustration

My code was not moving correctly, and I still have no idea why, but out of frustration I made a new unity 2D project and reimported the package from slate and re-did all of week 10's tasks to get the code to respond properly. Also turns out if you have miro open for more than 3 ish hours it starts to forget how to actually do things like move stuff, so new miro board duplicated from my old one.

Here is the new repository link: https://github.com/Xiomaraze/Assignment2_Platformer

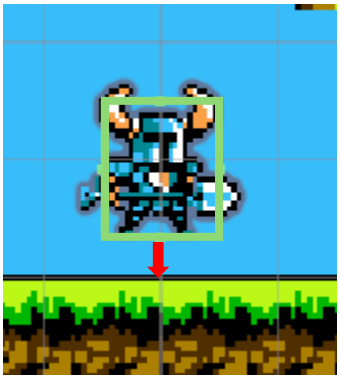
Here is the new miro board link: https://miro.com/app/board/uXjVLB-w8_I=/?share_link_id=500075425258

The pdf of the board is in the Project Folder along side this PDF. I will also add both as files to my submission. (Miro board also contains the images made for the proposals)

Task 1: maximum jump height and speed (this is where I got frustrated, I was determined to figure this out)

Time Taken: 1 day, 5 hours ish

This started off pretty easy, taking the given height and applying it to get the proper position for the maximum jump, and dividing by time to get the speed, but the hard part was getting the program to understand when a jump actually happened, to record the maximum jump position for that instance, vs when it was already in the air and just needed to wait for it to reach the already given maximum, or land. I ended up exploring into the raycast stuff more thoroughly, and I must admit I am a little disappointed in the unity api manual for not showing examples of how the raycasting works. I never realized that when I used the rigidbody2D.cast it used it directly from the attached collider's EDGE instead of center. And when I was trying to understand how the method used the distance variable to denote how far the ray could actually go, I never actually understood how to measure that. In the end I had to use the raycasthit2d.distance to check if the distance was under 0.01 to verify that yes, it was because the object was touching on the side I was looking at (in this case the bottom). Also unity has issues if you forget to give it a jump height and jump time. It does not enjoy the concept of infinity.



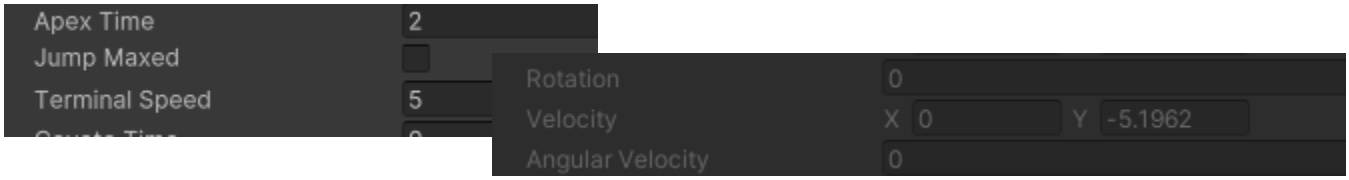
The image left shows the edge of the boxcollider on the player in green. The red arrow shows where the raycast in the code below moves between. From the green box the raycast is sent downwards.

```
public bool IsGrounded()
{//jumping complicates things huh
    int grounded = 0;
    rb2.Cast(Vector2.down, contacts);
    foreach (RaycastHit2D contact in contacts)
```

Task 2: Terminal Speed

Time Taken: 25 minutes

Alright in theory this one is pretty simple, just make sure your acceleration or addforce is a 0 in y if it's moving faster down than the terminal speed, and make sure the velocity on the rigidbody is limited by vector2.clampmagnitude. However, I have no idea how to test it. Figured it out, just let him fall from above the scene view and monitored the velocity in the inspector.



Task 3: Coyote Time

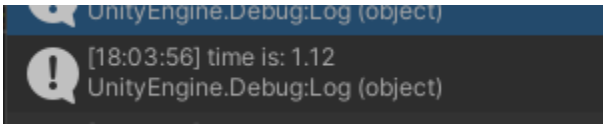
Time Taken: 17 minutes

This name is amusing, so I went with a silly name for the tracking variable. Basically I just have it mark what the time.time is everytime the isgrounded returns true, so once it's no longer returning true, the jump if statements can check if the player is grounded, OR, if they are still within coyote time of being grounded.

```
if (grounded > 0)
{
    lastTimeTouchGrass = Time.time;
    return true;
```

```
if (Input.GetKey(KeyCode.Space))
{
    //Debug.Log("time is: " + Time.time); coyotetime testing purposes
    if (apexHeight == 0)d
    {
```

Turns out it returns the time in seconds up to 2 decimals. (yes I fixed that weird out of place d)



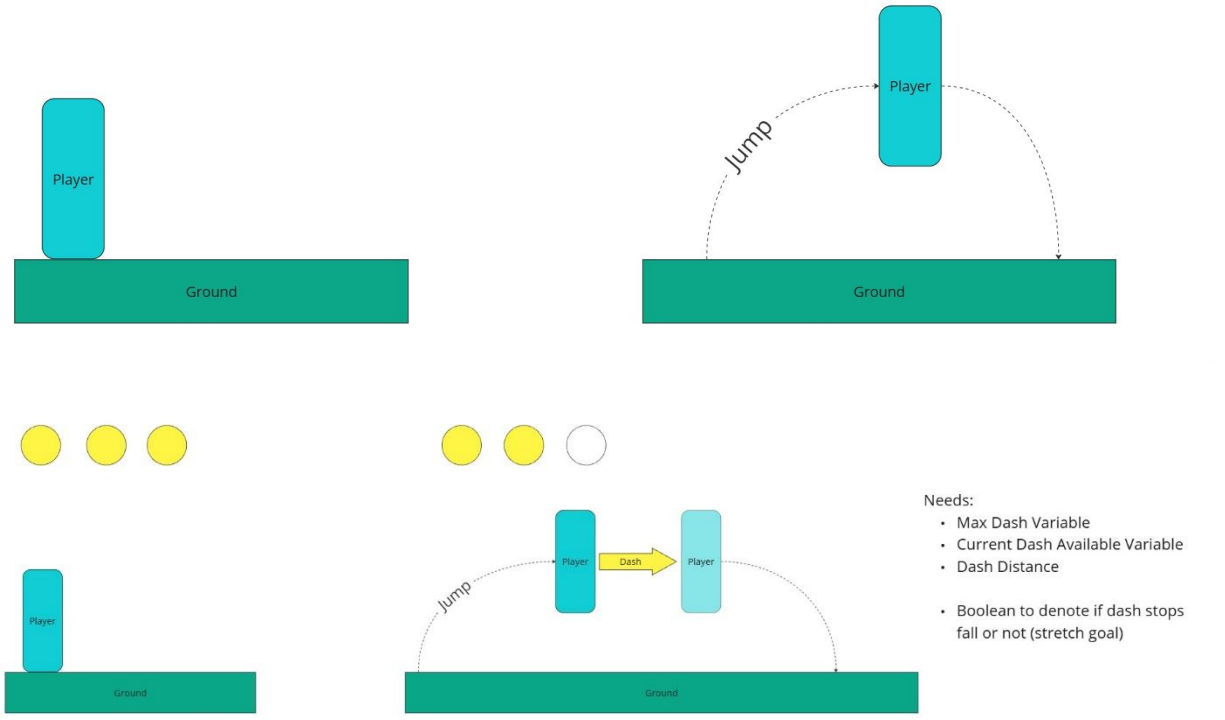
Part 3: Proposed mechanics

Proposal 1:

Simple dash mechanic via stored “charges”.

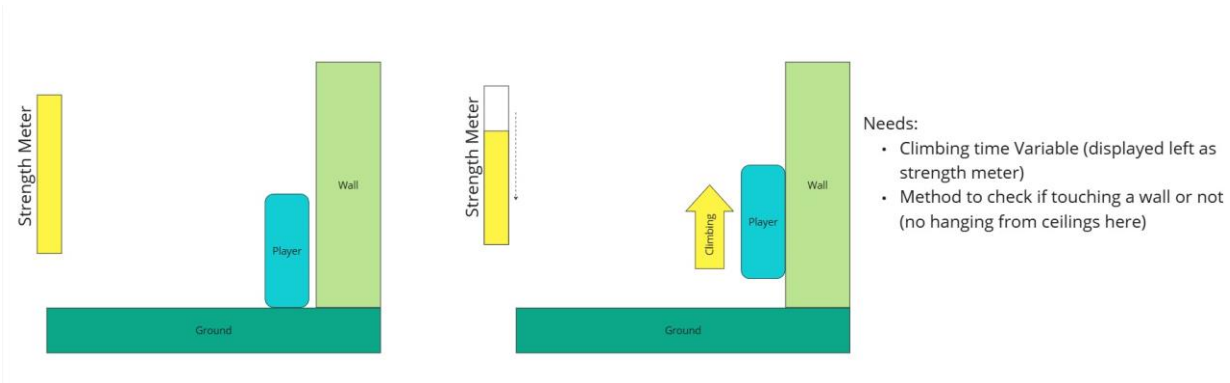
Nothing complex just tracking incoming speed, dash speed, and whether it stops a fall or not.

Current Jump state above, Proposed mechanic below.



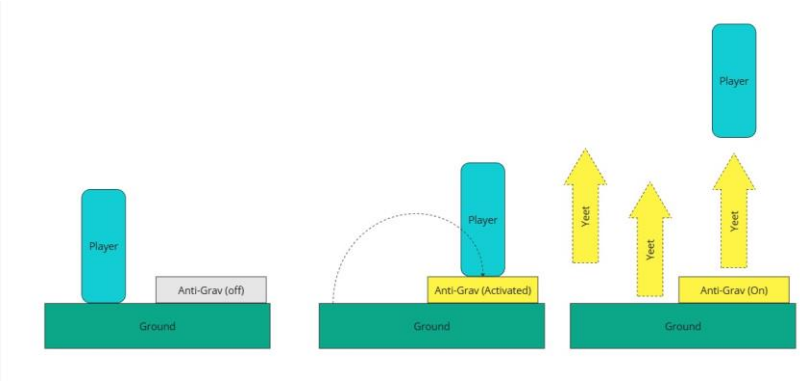
Proposal 2:

Climbing via a time-tracking variable and a Boolean method.



Proposal 3:

Gravity reverse button or trigger. Most likely to use the trigger collider states and the gravity scale in the 2d rigidbody system.



Needs:

- Tile variation to track ground grav state
- Boolean to track grav state
- A ceiling to not lose the player