Development Notes

**Task: Recreate the game The Line in Unity**

**Core features:**

* Initial Screen (still frame with the dot and instruction on how to play.
* Road generation (jumps if we have time?) We need an algorithm that creates a **continuous path** of rectangular units.
* Player controls: what packages do we use to access the “finger” location from an iOS device?
* Scoring system (simple distance travelled)
* Game over logic – collision-based failure detection
  + Overlay screen (probably not a separate scene) for game over.
  + **Collision (not using Unity Physics or OnTriggerEnter)**
* System should support easily adding more booster types in the future (object oriented inheritance)
* Jump over river logic

**UI & Screens:**

* Game over screen – Displays score and a “Try Again” button
* Menu screen – Includes “keep going” and “try again” buttons

**Do not need to implement sound, X6 path logic, sharing, or leaderboards**

**Important game objects:**

* The player: this is a blue circle, which can be represented with a radius p\_r and coordinates p\_x and p\_y
* Collision blocks: these are red blocks, slightly rectangular, of fixed width and height we can call block\_height and block\_width. Upon any sort of contact with the player, trigger the game over screen.

**Collision Implementation Notes:**

Lets keep track of the coordinates and size of both the player and the blocks. We will subdivide the game area in a grid division, which we then use to evaluate which blocks we want to actually care about. If character is in quadrant 1 of 9, we only check blocks where they are intersecting this portion of the screen.

**Path Generation Notes:**

For the path generation, we are only allowed to generate a path that is continuous. That is, there cannot be any open space where an open space does not connect to another open space through an up connection.

Mathematically, we can randomly generate this by doing the following:

1. Represent as vertical levels
2. For the path generation, have a number generator randomly choose between Left and Right and up. Once left is chosen, right cannot be and vice versa. To avoid paths with width 2, if the previous stage chose left, the following cannot choose right. Thus, we have something like:  
   LLLLF  
   F  
   F  
   F  
   RRF  
   F  
   In an ideal case.

**Extra notes**

* To simplify collisional coordinates, we can keep the player still and move the blocks.
* For efficiency, remove passed blocks as they go past the stage.
* What is the **Object pooling design pattern**?
  + It is used in game development to reuse objects instead of constant creation and destruction and should help reduce load on memory.
  + **Pool size**? About 8 visible lines blocks on screen with a width of 7 blocks for each line. Rounding to 10 for safety, this requires us to render about 70 blocks at any single instance

**Implementation Steps:**

1. Collision rectangle pool structure **done**
2. Movement, placement, spawn timings of collision rectangles **done**
3. Path generation algorithm **done**
4. Player movement **done**
5. Collision algorithm **done**
6. Creation of river tiles, Jumps **done**
7. Creation of boosters: **done**
8. Gameplay loop **done**
9. Menu, start up sequence, game over sequence **done**

**Future Things to be Done:**

1. Jump Tile Words, + instructions during approach
2. Player Color Changes/Animations during Jump / Invulnerability Booster State.
3. Power up timers.
4. Have a game reset procedure instead of just restarting the scene; scene restart takes a while to finish so the experience isn’t great.