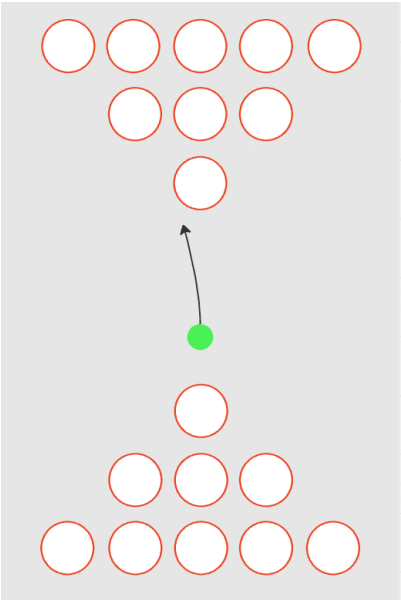
Ideation #1: Curving Beer Pong



For this game, Players would take turns trying to sink the ball in the other person's cup but there is a twist: The ball ALWAYS curves either left or right.

This will create a sense of difficulty for each player when sending the ball forward.

The players know which way it will curve but don't know exactly HOW much it curves.

The player whose turn it is selects a trajectory they want the ball to go and how much power to use. A line using Pmouse x and y would assist them in judging how much power will be used.

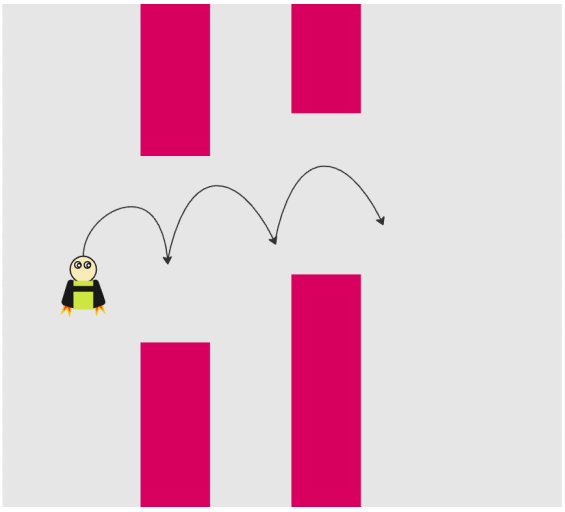
The game would have a start screen with a basic coin flip 50/50 to see who goes first. This will create early tension between players creating a more competitive vibe during the game. Hopefully, with that as a feature, it will increase replayability for players who want rematches or to challenge new people to the game.

Once the coin flip is completed the game will immediately begin with the player who won going first. The first player who removes all the balls from their opponent's board will get a win screen and be prompted to "PLAY AGAIN?". I was thinking of including it being color-changing to help encourage the player to try again for a better shot at winning next time.

Required Elements:

- 1 Object used for the cups, would have code that said "if the ball stops over its location, stop appearing
- The ball would need the use of a random sin value every time the turn changes over to the other player
- The ball would need a Pmouse x and y line that comes out when clicked and dragged towards the other player's balls to a certain limit
- A Pvector would determine the ball position, speed, and velocity after the button has been pressed to send it toward the cups

Ideation #2: Jetpack Flap



The player will control a little guy with a jetpack trying to avoid obstacles (not decided what kind yet).

The game will include tapping on the screen to boost the jets and the player upwards a small amount. The player will constantly be going forward but always falling.

The game would also include very simple power-ups that allow for temporary ease while the player, Things such as: “Remove all current things on the screen” or “Extra Life”

WIN/LOSE CONDITION:
To win the game, the player must reach a certain score that would increase by the frame count. I noticed in these kinds of games, the endless increase in difficulty does add a level of intense gameplay but causes too much boredom and frustration in players too quickly. Adding an actual win condition to a game like this will add more satisfaction to the player and maybe make the player come out of it happy rather than always being disappointed when dying.

Required Elements:

- The large obstacles would need to be objects that are selected from a range of patterns to how they would come out
- Power-ups would be randomized and have to be matched up with the large obstacles to make it possible to grab them
- The player would be a Pvector that uses its location and adds velocity which would change every time the player clicks the mouse
- The acceleration would have to increase as the player starts falling to the ground to create the urgency to save them before either hitting the ground or an obstacle

Ideation #3: Word Sort

In this game, players will attempt to create words out of the letters given, but the twist is the letters are constantly floating around the screen.

The letters would be randomized but simplified to always be able to create a word no matter what. The difficulty lies in the players trying to create these words while the letters are not only flying around but bouncing off one another.

Games like 'Wordle' give the players challenges by using point values depending on the letter, my game will be a race to make 5 words in 30 seconds.

Adding the time limit will create a proper challenge for the player without adding too much friction. It will be simple and hopefully enjoyable to watch as the letters fly all around the screen gently bouncing around.
Win/Lose Cond.
As previously stated, the way to win the game is to make 5 words out of the letters in the time limit. If the player fails to make these words in the 30 seconds given, the player will be brought back to the main menu and prompted to play again.

Required Elements:

- The letters need to be objects with randomized letters inside them sorted using an array.
- Each of the letter blocks will need to be Pvectors that include the position, velocity, and acceleration (acceleration might be randomized per block to make it a bit more difficult)
- The letter holder on the bottom will have to keep the blocks still (set the velocity and acceleration of them to 0 when entering the square)
- A timer is needed to track how long the game has been going and how much time is left before the game restarts.

Implementation Log #1: Win/Lose

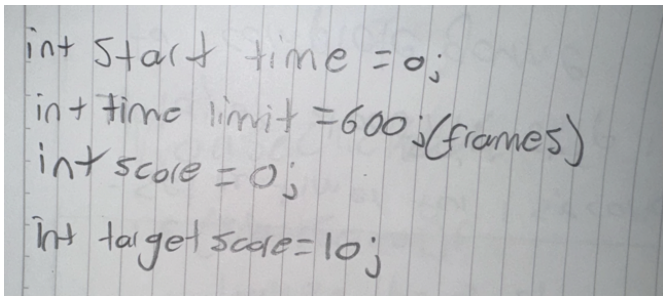
Problem:
Adding a win and loss condition was the first thing I wanted to make sure worked correctly. I have never had to think about different screens overlapping until now so using booleans and functions to display them without overlapping became very frustrating.

Solution:

1. Defined all the different booleans needed to later display the different screen later. By having each screen having a different boolean, you should be able to apply them to the functions that will display the screen later

```
boolean startScreen = true;
boolean inGame = false;
boolean gameOver = false;
```

2. Create variables for how the player will win and lose. timeLimit is how fast the player has to complete the game and the target score is how many times they will be pressing the ghost to win.



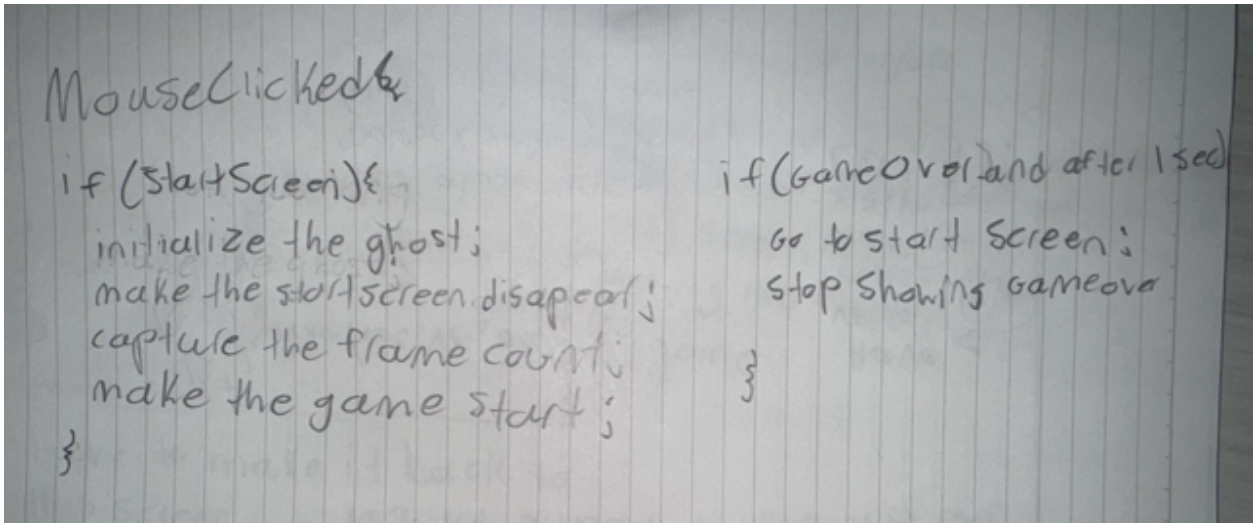
3. Use if statement to check both winning and losing. First one should be reaching the score, then if that isn't true then you get the ending screen. (Later added different timing to fix overlapping of spam clicking and entering these screens)

```
void runGame() {
  //set a way to check the amount of time that has passed while playing
  int elapsedTime = (millis() - startTime) / 1000;
  //Create the lose condition and win condition that will later display an indicator to the player
  //check if they won by reaching the targeted score
  if (score >= targetScore) {
    hasWon = true;
    gameOver = true;
    inGame = false;
    //if they instead took too long, make a game over happen
  } else if (elapsedTime >= timeLimit) {
    gameOver = true;
    hasWon = false;
    inGame = false;
  }
```

4. Design what the screens are actually going to look like as well start and end screens. Connect these screens to the booleans previously made to flick them on and off. For the game over, no need to make the winning and losing separate, making it an if statement that uses an else if for specifically losing.

```
void StartScreen() {
  background(0);
  image(Start, width/2, height/2);
  score = 0;
}


void GameOver() {
  background(0);
  if (hasWon) {
    image(Wining, width/2, height/2);
  } else {
    image(Lose, width/2, height/2);
  }
}
```

5. Now you just have to have a way of transferring between screens once the game is either over or in the starting screen. I want to do a mouse click (not pressed cause it will mess up in the game over screen and skip over the start screen cause the mouse will detect it is still up.



Testing and Iteration:
Originally, I didn't think far enough ahead when it came to how each screen would interact with the other. I tested an older version that included a lot of else-if statements thinking that having it check if it's not in one of the screens goes to another. This caused a lot of issues detecting if the score was correct so instead I separated them all into both booleans and functions, laying out everything made it so much easier to digest.

Outcome:
The win/loss conditions and feedback mechanisms significantly improved player experience. Players can now feel a sense of accomplishment or failure to play again when the game ends.

Implementation Log #2: Ghost Movements

Problem:
One key goal for the game was to make the ghost's movement scale in difficulty. Using just velocity made the motion too uniform and predictable. To address this, I used an acceleration vector to influence the ghost's velocity. This would allow the ghost's speed to vary increasing its difficulty substantially but integrating acceleration into the motion while ensuring it remained bounded by the screen edges presented unique challenges.

Solution:

1. Introduced a Pvector position, velocity, and acceleration variable to the Ghost class. These allow for freedom of movement later. Set the parameters to starting values of where I want it to start, how fast it will be at the start, and how much it will increase.

```
class Ghost {
  PVector position = new PVector(400, 200);
  PVector velocity = new PVector(0.5, 0);
  PVector acceleration = new PVector(0.09, 0);
  int size = 50;
```

2. Adjusted the Velocity vector with the acceleration of each frame using the add() method. This updates the position dynamically. .

```
  void update() {
    velocity.add(acceleration);
    position.add(velocity);
```

3. To prevent the ghost from moving out of bounds, I implemented an if statement to reverse both the velocity and acceleration directions when the ghost reached the screen edges. This ensured that it stayed visible but still felt responsive and dynamic.

```
if (position.x > width - size / 2 || position.x < 0 + size) {
  acceleration.x *= -1;
  velocity.x *= -1;
}
```

Testing and Iteration:
Initially, the ghost's speed increased uncontrollably, making it too difficult to click. I set the values to start pretty low as the limit is 10 seconds before failing so within that time, the max acceleration will be high but not too high to make it impossible by the end.

Outcome:
The ghost's motion now feels more natural and challenging. The acceleration mechanic added a subtle but noticeable improvement to gameplay, making it more engaging while maintaining balance.

Implementation Log #3: Background Obj.

Problem:
To give the factory background more complexity, I included multiple machines in different positions. To make the game scalable and organized, I decided to store the machines in an array. A secondary challenge was spreading the machines out by their x-coordinates to render them in the correct order for a visually consistent layout.

Solution:

1. Start by creating and initializing the array Machine[] machines = new Machine[6] to include a maximum of 6 machines
2. Create the class and include a display function so it can be called later. Also set position variables within the constructors to later change the distance between them

```
class Machine {
  float x, y;

  Machine(float posX, float posY) {
    x = posX;
    y = posY;
  }

  void Mdisplay() {
    fill(100, 100, 120);
    rect(x, y-100, 100, 150);

    fill(200, 50, 50);
    ellipse(x + 50, y + 20, 20, 20);
  }
}
```

3. Set up the loops to go through the array to display them in the background. For the loop in setup, make sure to add the parameters to the object that will add a gap between the previous one depending on how far in the array you are.

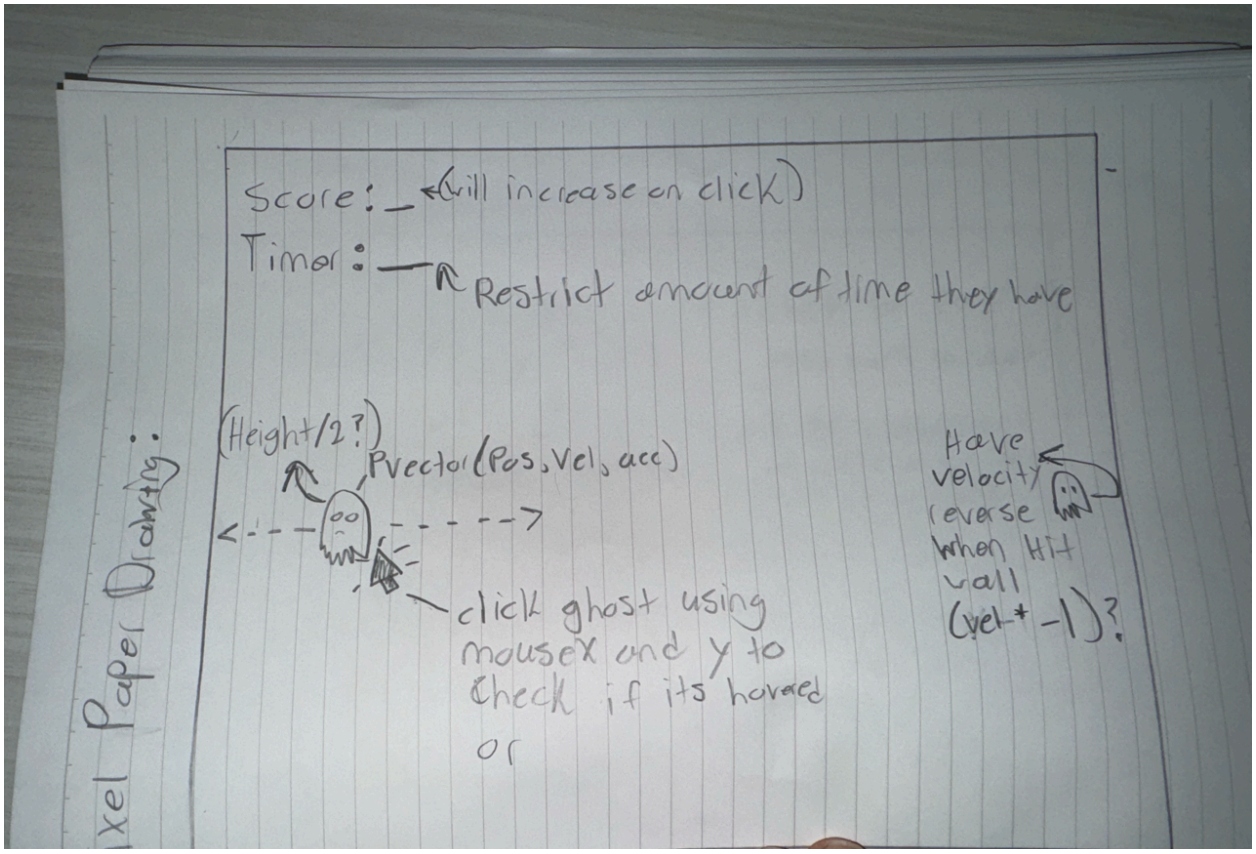
```
for (int i = 0; i < machines.length; i++) {
  machines[i] = new Machine(50 + i * 150, random(50,300));
}

for (int i = 0; i< machines.length; i++) {
  machines[i].Mdisplay();
}
```

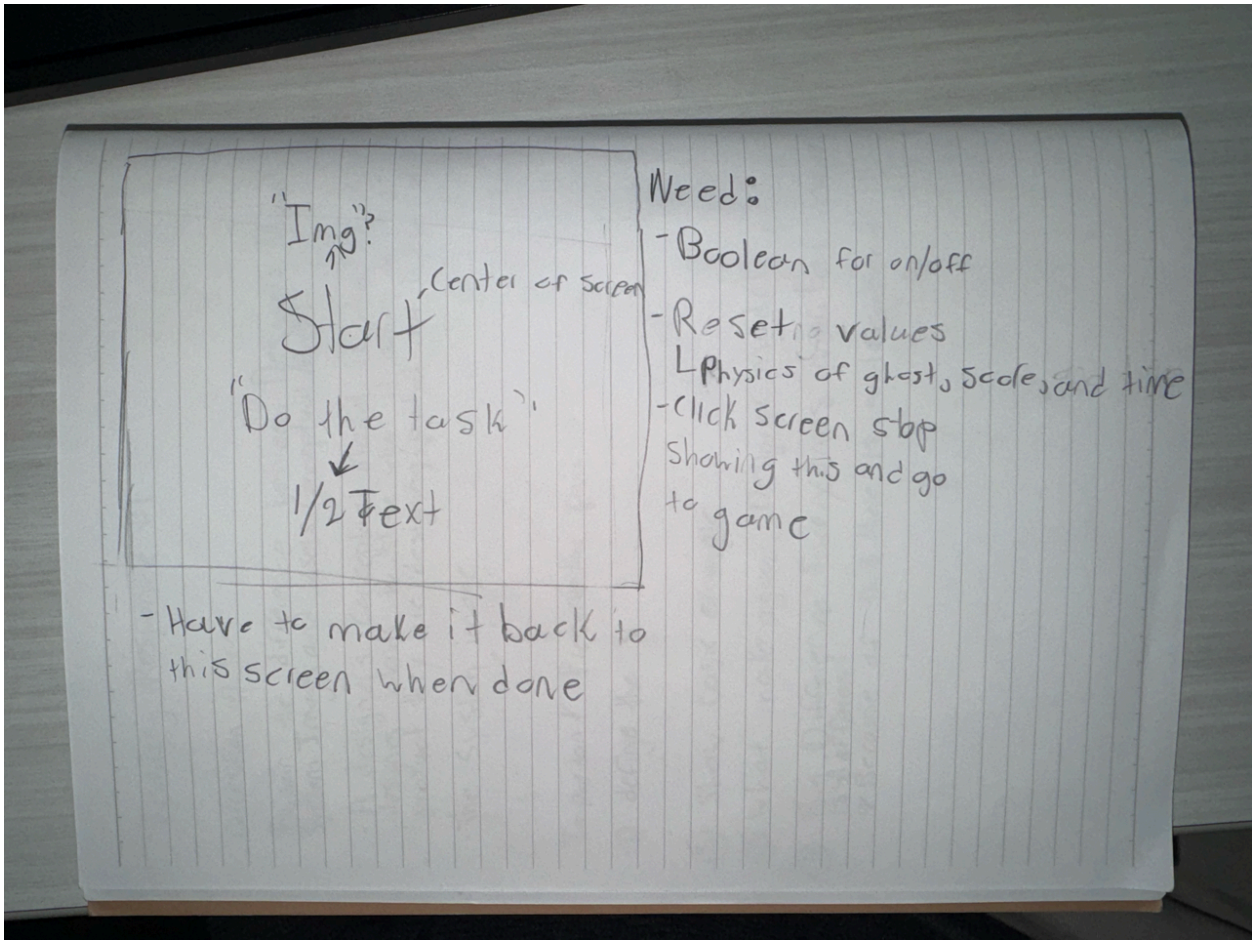
j

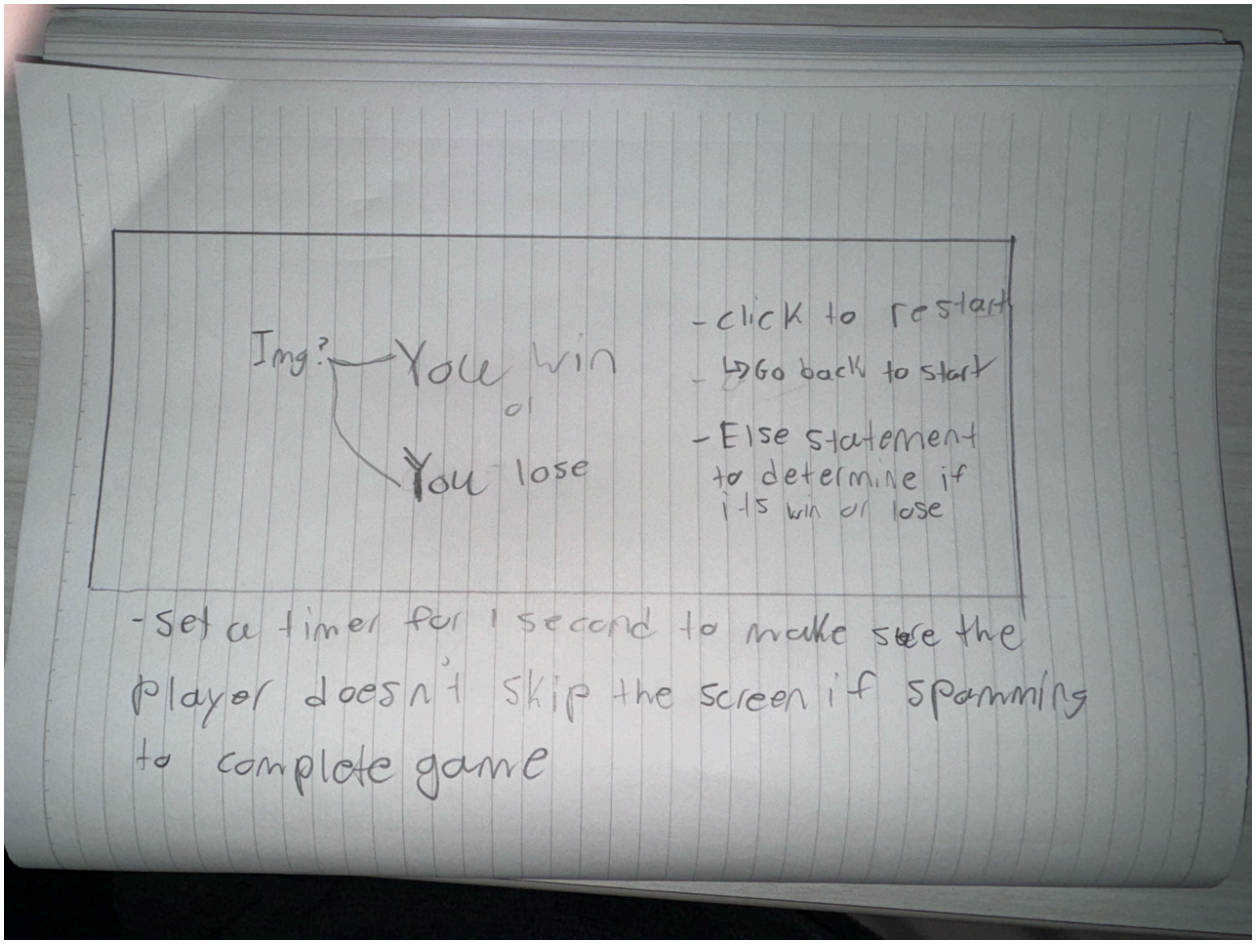
Testing and Iteration:
Sorting the machines required little iterations fortunately, the main changes I made were the amount of object machines within the array. Depending on the amount, I had to change the values of the gap between them to spread them out evenly.

Outcome:
The array made the background more organized and reusable. Adding more machines or modifying their positions is straightforward, as changes to the array are automatically reflected in the game

Pixel Paper Drawings:




"Img"
Start
Center of screen
"Do the task"
1/2 Text
- Have to make it back to this screen when done
Need:
- Boolean for on/off
- Resetting values
L Physics of ghost, score, and time
- Click screen stop showing this and go to game



- set a timer for 1 second to make sure the player doesn't skip the screen if spamming to complete game