Pandemic Tennis

Team Members:

- Michael Inoue, michaelinoue@g.ucla.edu, GitHub: michaelinoue, UID: 405171527
- Utsav Munendra, utsavm9@g.ucla.edu, GitHub: utsavm9, UID: 805127226
- Da Yuen Kim, dayuen7@g.ucla.edu, GitHub: dayuenkim, UID: 305096821

Team Representative: Utsav Munendra

Pandemic Tennis is 3D single-player table tennis. Because it is hard to draw a mask in the game and the pandemic is still ongoing, the player will be playing alone, hitting the ball to a wall, trying to not miss it.

Gameplay:

- Through collision detection, the wall will send back the ball to the player.
- The goal of the game is to keep hitting the ball back for as long as possible, and the game ends when the player does not place the paddle at the right position and misses the ball.
- As the player keeps sending back the ball, the score will keep increasing.
- We can keep track of the high score for the current gameplay. However, it would not be persistent between page reloads.

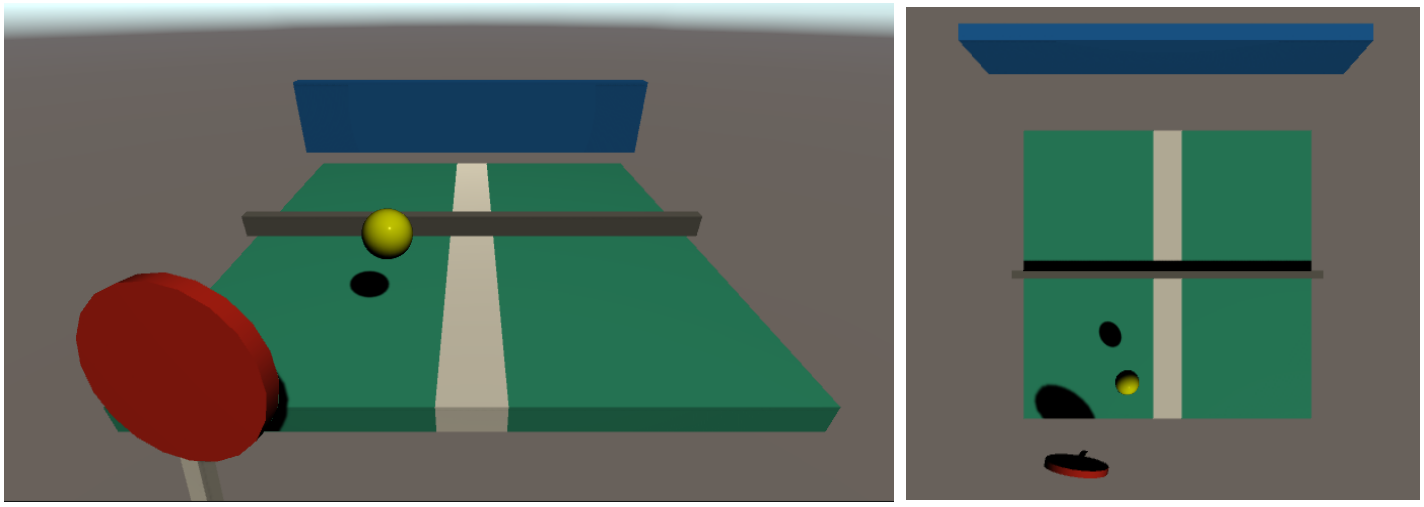

Figure 1: 3D concept design of Pandemic Tennis. Made with Unity

Game objects:

- Paddle:
    - A flat cylinder for paddle's blade, uniform color
    - A long cuboid for paddle's stick, wood texture
    - The paddle would only be movable in the up-down/left-right plane. We are not going to allow the paddle to go inside the table's edge for simplicity
    - The JavaScript object for paddle should provide functions to move the paddle while checking bounds
- Ball:
    - A yellow sphere
    - Javascript objects store the x,y,z limits of the ball, and check the location on the appropriate axis to find if there is a collision between the ball, paddle or wall.
    - Maintain variables for the current position, speed in each axis, and target position. Calculates new position coordinates with timestep.
    - Speed can change as time progresses.
- Table and Wall:
    - A green long cuboid.
    - The table is a stationary playing field.
    - Is not doing any calculations and only provides constants for the ball to access.

Collision detection:

- Collision between paddle/wall and ball:
    - As soon as a collision is detected, the ball will calculate a random point on the other side of the table and will then reset its speed vector to move toward that random point.
- Collision between table and ball:
    - When the ball collides with the table, the balls will retain its x,y velocity component but only change its z (up and down) velocity component.
- Collision between paddle's side and ball:
    - In the case where the ball's trajectory misses the center of the paddle and instead collides with the side of the paddle, the ball will move sideways out of view away from the paddle.

Calculating Ball Trajectory

- Parabolic curve:
    - The z-velocity would be constantly accelerating with *g*
- Random position section:
    - When the ball collides with the paddle or the wall, the ball will calculate a "random" position on the other side of the table to bounce from. This "random" position is picked from a fixed area on the other side of the table to prevent the ball from flying off the screen or to have an unreachable ball.
    - Paddle's velocity and direction does not affect on the ball's calculations.

Gameplay Interaction

- Keyboard interaction from the player. Able to move up and down, left and right to interact with the ball.
- Mouse clicking vs 3D Scene
    - We want to smoothly move the camera in case the ball's view might be obstructed due to the paddle. This also highlights the 3D nature of our world and is easy to transform the camera coordinates in tiny-graphics
    - We can make the paddle move with the mouse, but that would require us to fix the camera coordinates
    - In addition, calculating the damped motion of the paddle from mouse coordinates is trickier. Therefore, we are choosing to have a fixed scene with the paddle controlled solely by the keyboard.
    - We will, however, look into utilizing mouse input only if time permits.

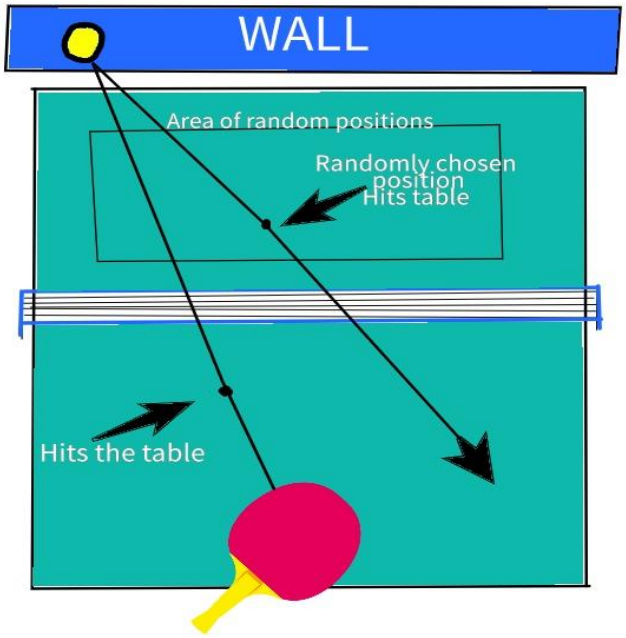

Figure 2: Top-down view of the game, with ball collisions marked