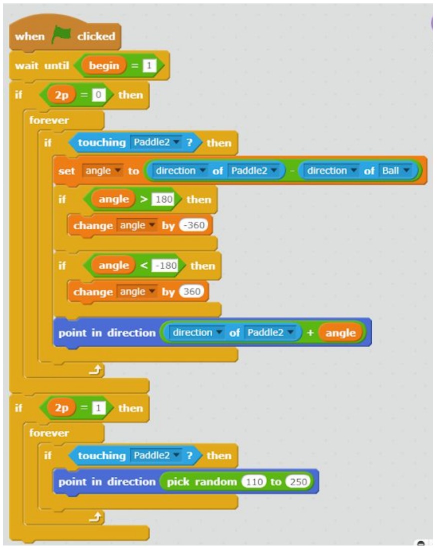
The function of my program goes as follows: the player selects a button on the main menu and each one of those buttons runs a different section of code. The high score button shows the high scores, the 2 player button runs the program with the A.I. deactivated, (replaced with a second player) the reset record button will reset the record, and the start button will run the program like normal.

During the development process, I first started with the movement of the player's paddle. I then added the A.I movement and ball angular momentum. The first day. The first testing started. I then added the menu screen, and more testing followed. The second day. After a few peer tests, I added 2 player mode, a record, and finally, after many fails, angle reflection with the ball, as per peer recommendation. The third day. I then added high scores, and final testing began. I then released the game as v. 0.1.0.

Two difficulties I had when developing the program were angle reflection (for the ball to act like real physics) and the high score list. The high score list was hard to fix since it required a new high score to see if it worked. The ball angle reflection was hard to develop since physics are hard to replicate in a non-physics game development workspace. I resolved the high score problem with a little help from TheRicks2, and the ball reflection was fixed with little Scratch research.



```
when clicked
wait until <(begin) = [1]>
if <(2p) = [0]> then
    forever
        if <touching [Paddle2] ?> then
            set [angle] to (([direction] of [Paddle2]) - ([direction] of [Ball]))
            if <(angle) > [180]> then
                change [angle] by (-360)
            end
            if <(angle) < [-180]> then
                change [angle] by (360)
            end
            point in direction (([direction] of [Paddle2]) + (angle))
        end
    end
end
if <(2p) = [1]> then
    forever
        if <touching [Paddle2] ?> then
            point in direction (pick random (110) to (250))
        end
    end
end
```

^^ This is the algorithm

This algorithm will test to see if the ball is touching the paddle, and if it is, it will set the angle of the ball to the direction of the paddle minus the direction of the ball, but if the angle is greater than 180, it will subtract 360 to the angle. If it is less than -180, it will add 360 to the angle. The ball will then point in the direction the computer decided before with the algorithm, so that the ball will reflect its angle toward the paddle making the pong game more realistic. This is all done by the computer.

The two algorithms within the algorithm are the testing to see if the angle is greater than 180 and then subtracting 360, or testing to see if it is less than -180 and then adding 360. This functions independently since there are a lot of variables used solely in this algorithm.

My abstraction used in this program was the high score segment.

The high score segment

This segment of code will wait until the ball hits the line, then tests to see if the current score is higher than any of the other scores. If it is, the score will replace the old score and then the lower scores will move down one. This manages the complexity since now there is a management of scores.