Module: Problem Solving and Programming Concepts

Assignment: CA2 Individual Project Snap Application Game

By Colin Shaw

Introduction:

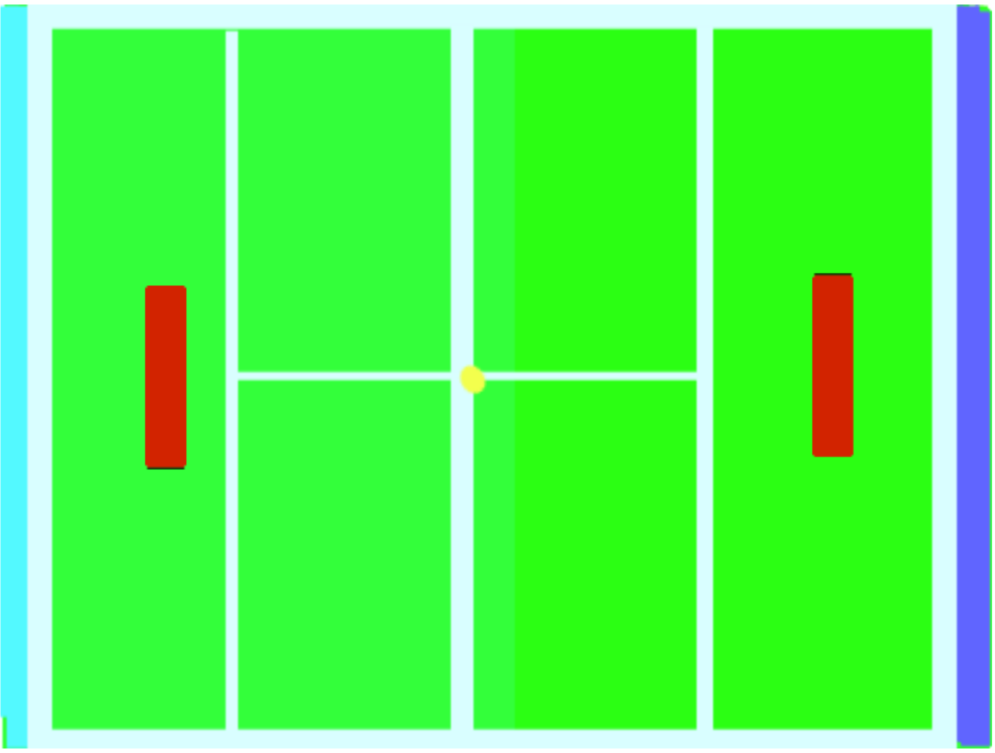
For CA2 we were tasked with building a Snap application puzzle or game. We had the choice when implementing a game that it could be an interactive, partially or fully "automated" interactive version. I chose a 2-player version of the Atari classic Pong. The reason I choose pong is as a kid I loved playing it in arcades and I wanted to test myself by building a version of pong I could enjoy playing against another person. With pong, the game rules are simple the person with the most points wins or who reaches the score level first wins. There are two paddles one on the left and one on the right and their job is to protect their goal line. You have one ball and if the ball hits the goal behind the player 1 goal that is one point for player 2 and if the ball the goal behind player 2 that is a point for player 1.

CA2 Pong 2player game!!

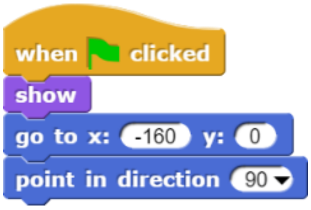
Documentation of implementing code for creating Pong.

I found it too difficult to create a game board that I actually liked for Pong using code. So, I found painting aboard looked best and created the best look board.

Painted game board

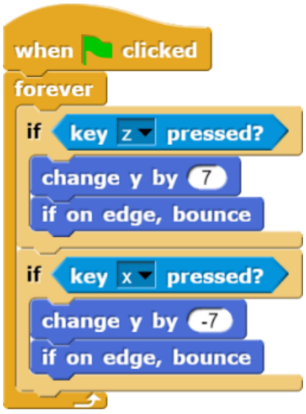


Giving instructions for player 1 to return to home position when clicked.



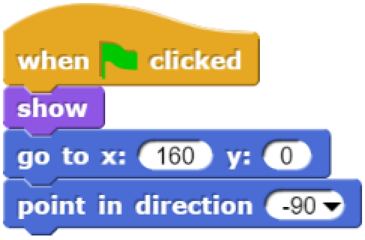
```
when [flag] clicked
show
go to x: -160 y: 0
point in direction 90
```

Giving player 1 paddle commands to move up and down on the y-axis. Use the z key to move up the y-axis and the x key to move down the y-axis.



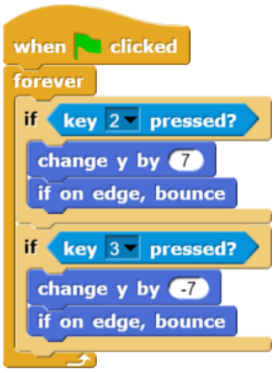
```
when [flag] clicked
forever
  if key z pressed?
    change y by 7
    if on edge, bounce
  if key x pressed?
    change y by -7
    if on edge, bounce
```

Instructions for player 2 to return home when clicked.



```
when [flag] clicked
show
go to x: 160 y: 0
point in direction -90
```

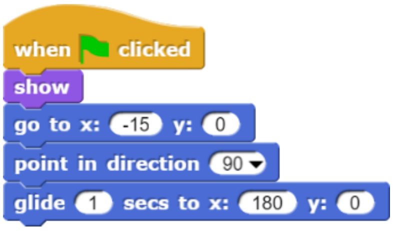
Giving player 2 paddle commands to move up and down on the y-axis. Use 2 to move up the y-axis and 3 to move down the y-axis.



```
when [flag] clicked
forever
  if key 2 pressed?
    change y by 7
    if on edge, bounce
  if key 3 pressed?
    change y by -7
    if on edge, bounce
```

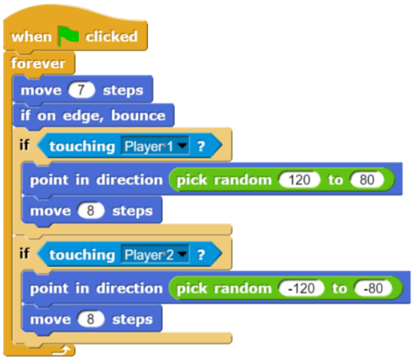
Starting with game instructions for the ball.

Building the blocks for the ball to return home and move in the direction of player 2 when clicked.



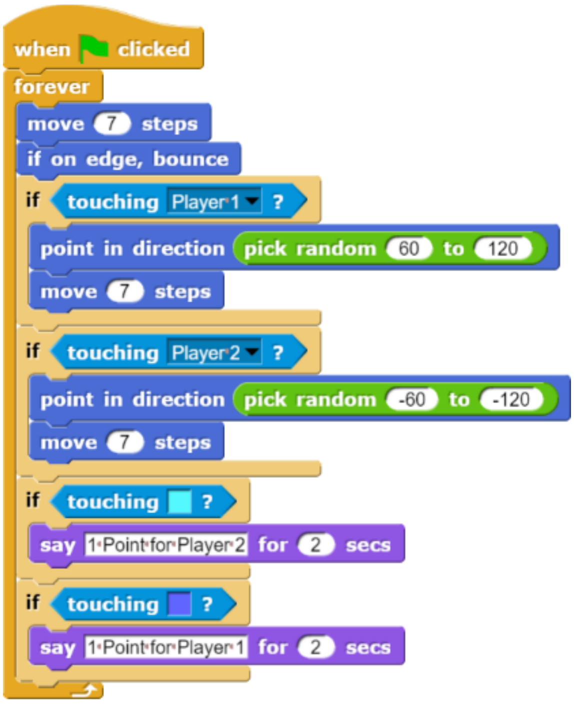
```
when 🏳 clicked
show
go to x: -15 y: 0
point in direction 90 ▾
glide 1 secs to x: 180 y: 0
```

Giving the ball instructions for when it hits the edges it bounces back onto the court. Also, for when it touches player 1 paddle it bounces in the direction back towards player 2 and when it touches players 2 paddles it bounces back in the direction of player 1.



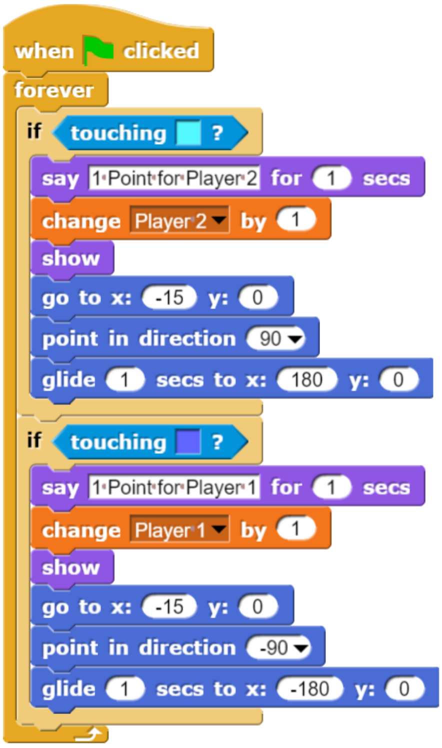
```
when 🏳 clicked
forever
  move 7 steps
  if on edge, bounce
  if touching Player1 ▾ ?
    point in direction pick random 120 to 80
    move 8 steps
  if touching Player2 ▾ ?
    point in direction pick random -120 to -80
    move 8 steps
```

Adding in the instructions for when player 1 scores a point by beating player 2 paddle and hitting the dark blue colour behind it saying 1 point for player 1 for 2 seconds. Also, visa Versa when player 2 beats player 1 paddle and hits the light blue colour behind, it says 1 point for player 2.



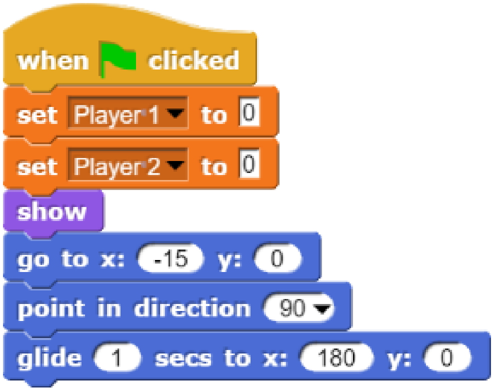
```
when 🏳 clicked
forever
  move 7 steps
  if on edge, bounce
  if touching Player1 ▾ ?
    point in direction pick random 60 to 120
    move 7 steps
  if touching Player2 ▾ ?
    point in direction pick random -60 to -120
    move 7 steps
  if touching 🟦 ?
    say 1 Point for Player 2 for 2 secs
  if touching 🟦 ?
    say 1 Point for Player 1 for 2 secs
```

Instructing the ball and adding it to the existing code. In the blocks below instructions for when player 1 scores a point the ball resets from its home position and goes in the direction of player 2 and one point is added to player 1 score. When player 2 scores a point the ball resets from its home position and goes towards player 1 and one point is added to player 2 scores.



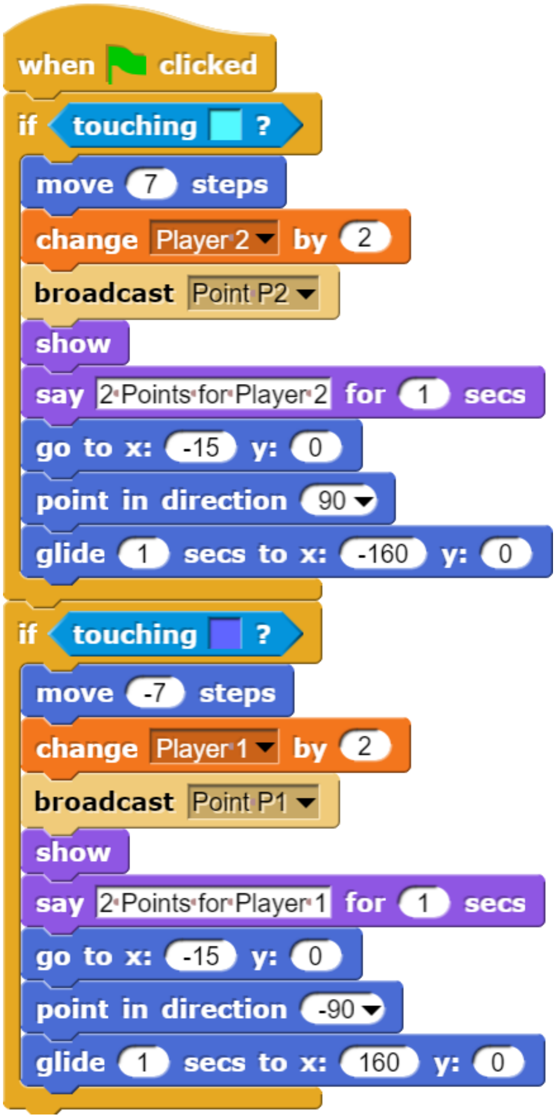
```
when ⚑ clicked
forever
  if touching 🟦 ?
    say 1 Point for Player 2 for 1 secs
    change Player 2 ▾ by 1
    show
    go to x: -15 y: 0
    point in direction 90 ▾
    glide 1 secs to x: 180 y: 0
  if touching 🟦 ?
    say 1 Point for Player 1 for 1 secs
    change Player 1 ▾ by 1
    show
    go to x: -15 y: 0
    point in direction -90 ▾
    glide 1 secs to x: -180 y: 0
```

Adding in the scoreboard variable to be reset to 0 when clicked



```
when ⚑ clicked
set Player 1 ▾ to 0
set Player 2 ▾ to 0
show
go to x: -15 y: 0
point in direction 90 ▾
glide 1 secs to x: 180 y: 0
```

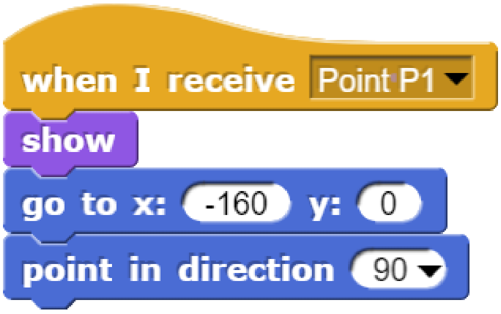
Adding in broadcast controls for when player 1 and player 2 score 2 points. I changed from one to two points as I found it made for a better game.



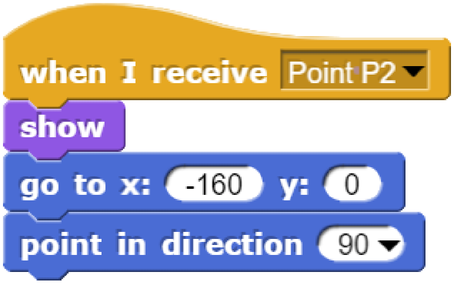
```
when ⚑ clicked
if < touching ■ ? >
  move (7) steps
  change [Player 2 ▾] by (2)
  broadcast [Point P2 ▾]
  show
  say [2 Points for Player 2] for (1) secs
  go to x: (-15) y: (0)
  point in direction (90 ▾)
  glide (1) secs to x: (-160) y: (0)
if < touching ■ ? >
  move (-7) steps
  change [Player 1 ▾] by (2)
  broadcast [Point P1 ▾]
  show
  say [2 Points for Player 1] for (1) secs
  go to x: (-15) y: (0)
  point in direction (-90 ▾)
  glide (1) secs to x: (160) y: (0)
```

Adding in the rules for each broadcast command.

Player 1

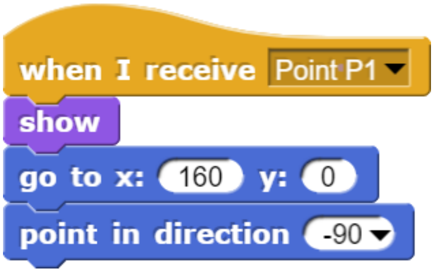


```
when I receive [Point P1 ▾]
show
go to x: (-160) y: (0)
point in direction (90 ▾)
```

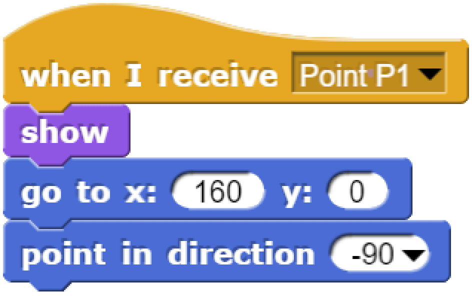


```
when I receive Point P2 ▾
show
go to x: -160 y: 0
point in direction 90 ▾
```

Player 2



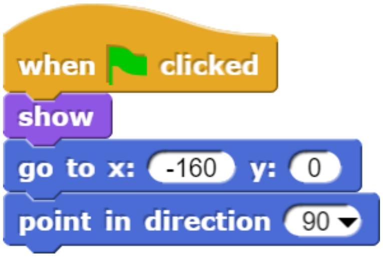
```
when I receive Point P1 ▾
show
go to x: 160 y: 0
point in direction -90 ▾
```



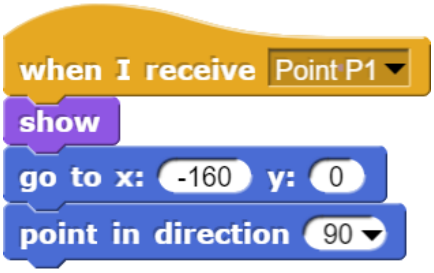
```
when I receive Point P1 ▾
show
go to x: 160 y: 0
point in direction -90 ▾
```

Code in its final cleaned up Game state when clicked!

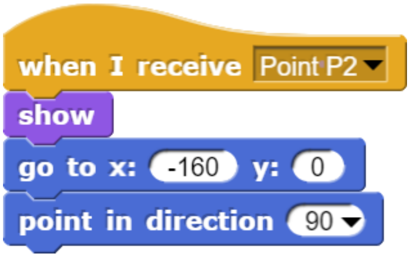
Player 1



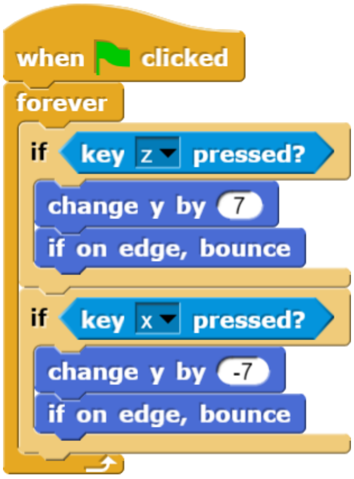
```
when ⚑ clicked
show
go to x: -160 y: 0
point in direction 90 ▾
```



```
when I receive Point P1 ▾
show
go to x: -160 y: 0
point in direction 90 ▾
```

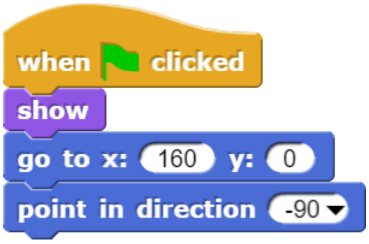


```
when I receive Point P2
show
go to x: -160 y: 0
point in direction 90
```

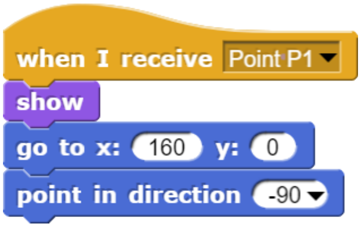


```
when ⚑ clicked
forever
  if key z pressed?
    change y by 7
    if on edge, bounce
  if key x pressed?
    change y by -7
    if on edge, bounce
```

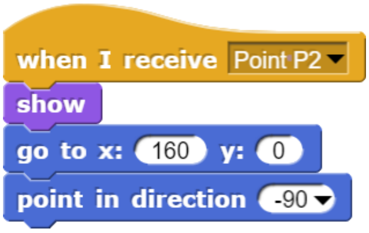
Player 2



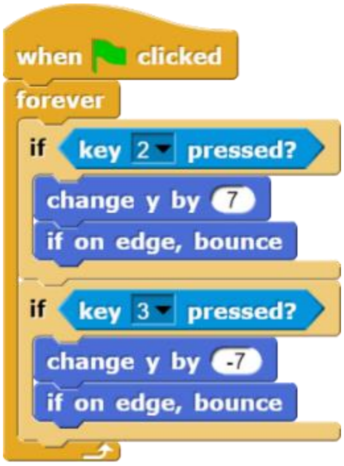
```
when ⚑ clicked
show
go to x: 160 y: 0
point in direction -90
```



```
when I receive Point P1
show
go to x: 160 y: 0
point in direction -90
```

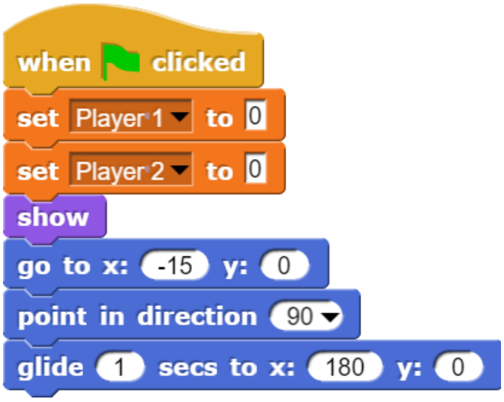


```
when I receive Point P2
show
go to x: 160 y: 0
point in direction -90
```

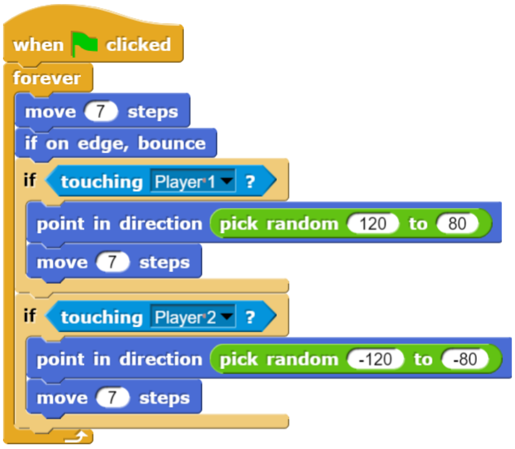


```
when [flag] clicked
forever
    if <key [2] pressed?>
        change y by (7)
        if on edge, bounce
    if <key [3] pressed?>
        change y by (-7)
        if on edge, bounce
```

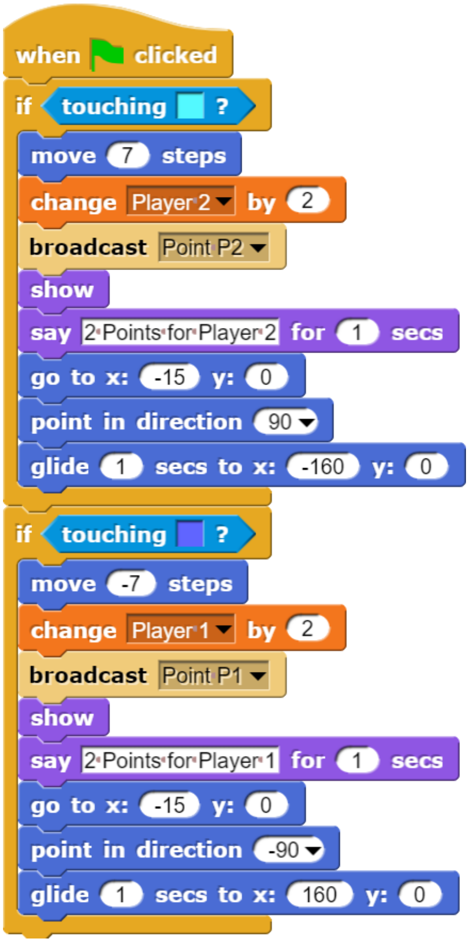
Ball commands



```
when [flag] clicked
set [Player 1] to [0]
set [Player 2] to [0]
show
go to x: (-15) y: (0)
point in direction (90)
glide (1) secs to x: (180) y: (0)
```

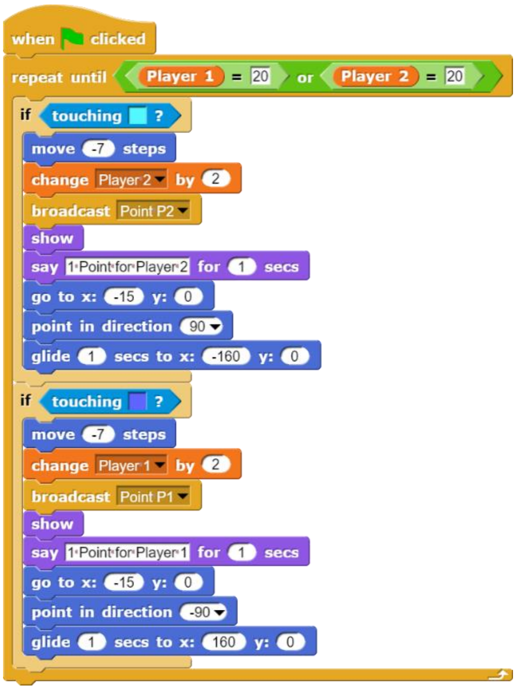


```
when [flag] clicked
forever
    move (7) steps
    if on edge, bounce
    if <touching [Player 1] ?>
        point in direction (pick random (120) to (80))
        move (7) steps
    if <touching [Player 2] ?>
        point in direction (pick random (-120) to (-80))
        move (7) steps
```



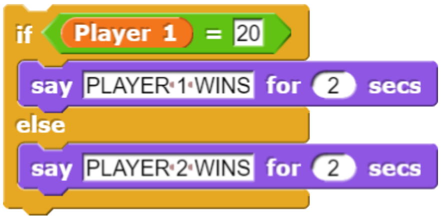
```
when 🏳 clicked
if <touching 🟦 ?>
    move (7) steps
    change [Player 2 ▾] by (2)
    broadcast [Point P2 ▾]
    show
    say [2 Points for Player 2] for (1) secs
    go to x: (-15) y: (0)
    point in direction (90 ▾)
    glide (1) secs to x: (-160) y: (0)
if <touching 🟦 ?>
    move (-7) steps
    change [Player 1 ▾] by (2)
    broadcast [Point P1 ▾]
    show
    say [2 Points for Player 1] for (1) secs
    go to x: (-15) y: (0)
    point in direction (-90 ▾)
    glide (1) secs to x: (160) y: (0)
```

Adding in the final rules for final Game state when a player scores 20 points.

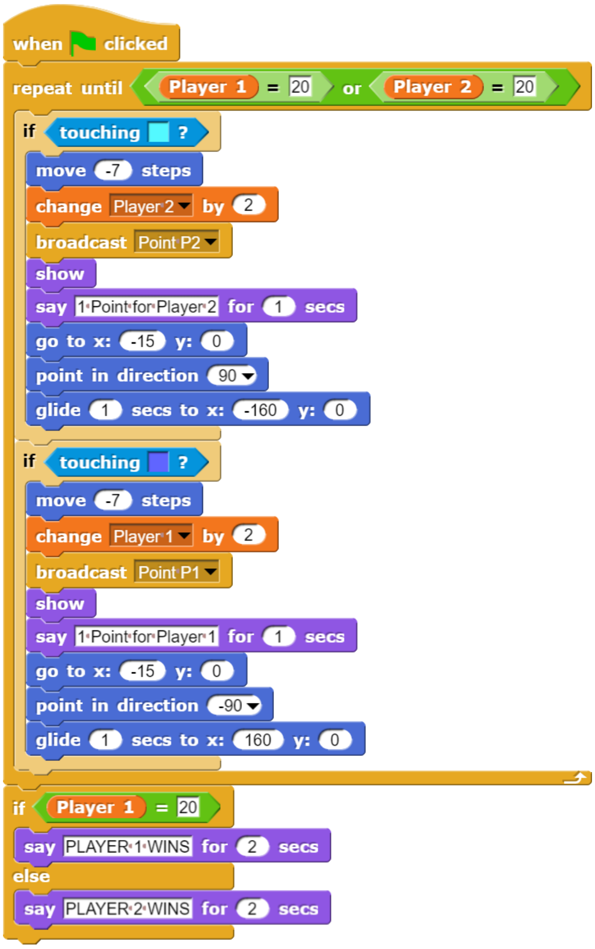


```
when 🏳 clicked
repeat until <<(Player 1) = [20]> or <(Player 2) = [20]>>
    if <touching 🟦 ?>
        move (-7) steps
        change [Player 2 ▾] by (2)
        broadcast [Point P2 ▾]
        show
        say [1 Point for Player 2] for (1) secs
        go to x: (-15) y: (0)
        point in direction (90 ▾)
        glide (1) secs to x: (-160) y: (0)
    if <touching 🟦 ?>
        move (-7) steps
        change [Player 1 ▾] by (2)
        broadcast [Point P1 ▾]
        show
        say [1 Point for Player 1] for (1) secs
        go to x: (-15) y: (0)
        point in direction (-90 ▾)
        glide (1) secs to x: (160) y: (0)
```

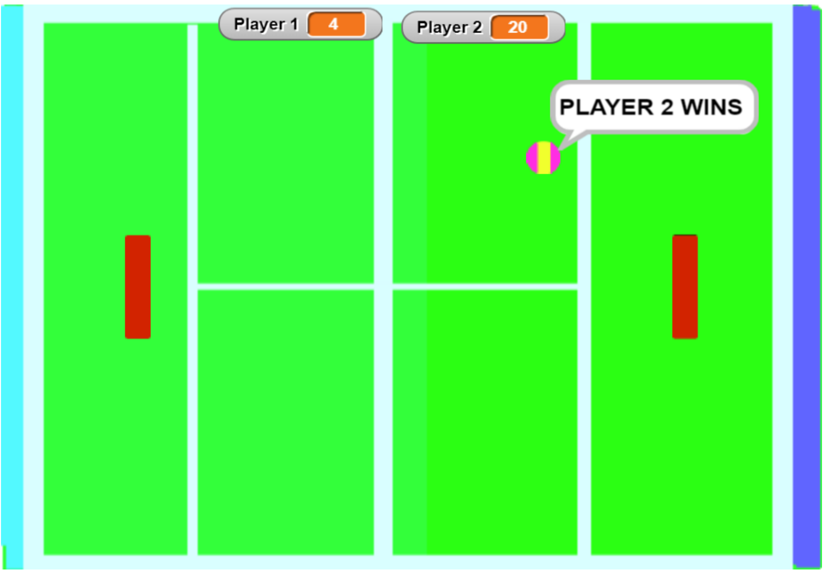
Adding in rules for if player 1 scores 20 points say player one wins and if-else player 2 wins



```
if < (Player 1) = 20 >
    say PLAYER 1 WINS for 2 secs
else
    say PLAYER 2 WINS for 2 secs
```



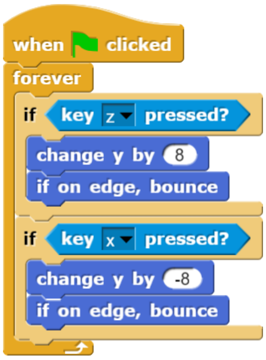
```
when 🏁 clicked
repeat until < < (Player 1) = 20 > or < (Player 2) = 20 > >
    if touching 🟦 ?
        move -7 steps
        change Player 2 ▾ by 2
        broadcast Point P2 ▾
        show
        say 1 Point for Player 2 for 1 secs
        go to x: -15 y: 0
        point in direction 90 ▾
        glide 1 secs to x: -160 y: 0
    if touching 🟦 ?
        move -7 steps
        change Player 1 ▾ by 2
        broadcast Point P1 ▾
        show
        say 1 Point for Player 1 for 1 secs
        go to x: -15 y: 0
        point in direction -90 ▾
        glide 1 secs to x: 160 y: 0
if < (Player 1) = 20 >
    say PLAYER 1 WINS for 2 secs
else
    say PLAYER 2 WINS for 2 secs
```



To speed up the ball and paddles we just increase the number of steps, in the case of the above code the number of steps is 7, so if we increase this to 8 the ball and paddles move faster and makes for a quicker and harder game. The same principle is applying if we want to slow the game down and make the game easier, we would decrease the steps by depending on how slow we wanted the ball and paddles to move.

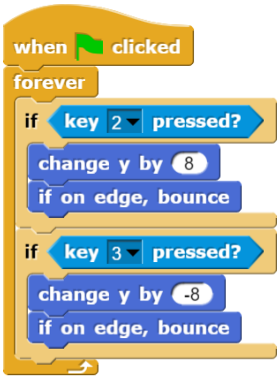
Example of the Game State if we increase the number of steps from 7 to 8.

Player 1 paddle commands.



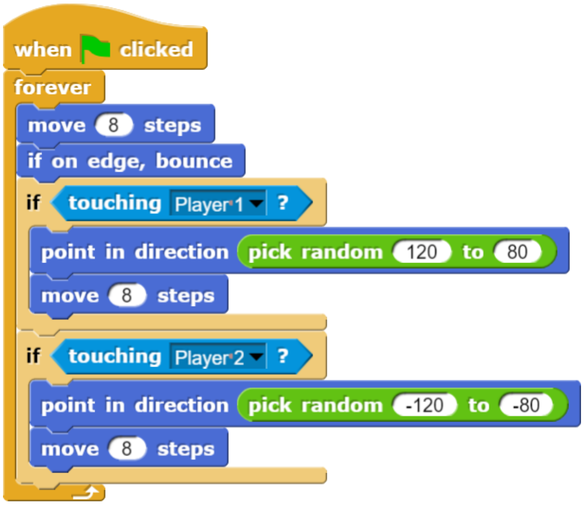
```
when ⚑ clicked
forever
  if <key z pressed?>
    change y by 8
    if on edge, bounce
  if <key x pressed?>
    change y by -8
    if on edge, bounce
```

Player 2 paddle commands.

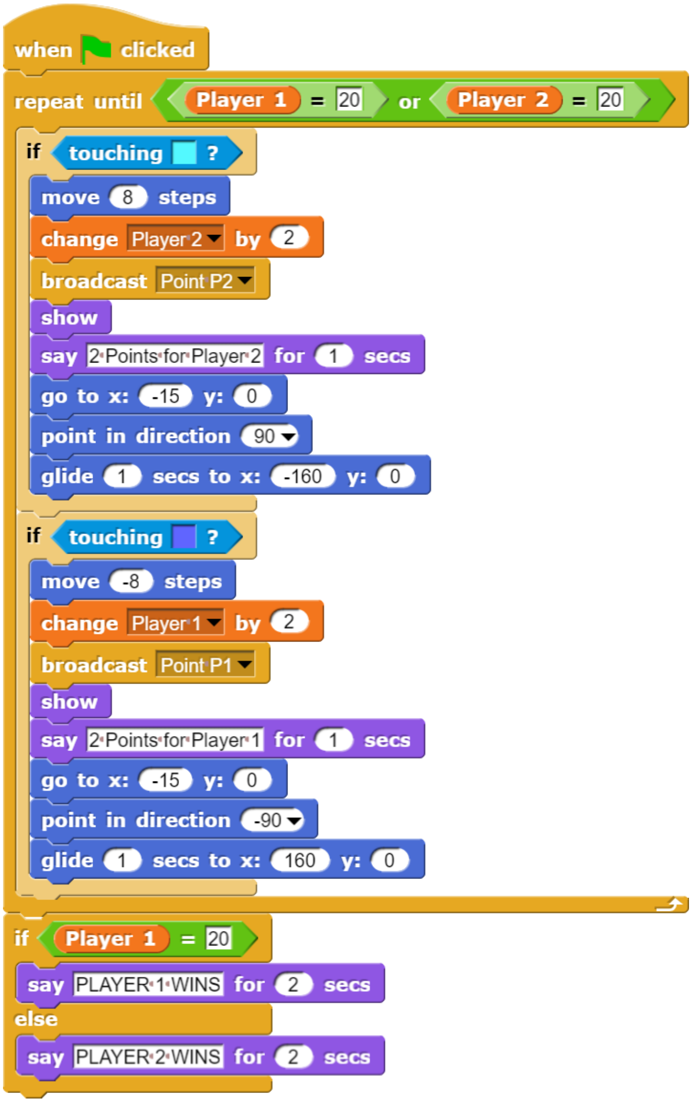


```
when ⚑ clicked
forever
  if <key 2 pressed?>
    change y by 8
    if on edge, bounce
  if <key 3 pressed?>
    change y by -8
    if on edge, bounce
```

Ball commands.



```
when ⚑ clicked
forever
  move 8 steps
  if on edge, bounce
  if <touching Player1 ?>
    point in direction (pick random 120 to 80)
    move 8 steps
  if <touching Player2 ?>
    point in direction (pick random -120 to -80)
    move 8 steps
```



```
when 🏁 clicked
repeat until < (Player 1) = 20 > or < (Player 2) = 20 >
    if touching 🟦 ?
        move 8 steps
        change Player 2 ▾ by 2
        broadcast Point P2 ▾
        show
        say 2 Points for Player 2 for 1 secs
        go to x: -15 y: 0
        point in direction 90 ▾
        glide 1 secs to x: -160 y: 0
    if touching 🟦 ?
        move -8 steps
        change Player 1 ▾ by 2
        broadcast Point P1 ▾
        show
        say 2 Points for Player 1 for 1 secs
        go to x: -15 y: 0
        point in direction -90 ▾
        glide 1 secs to x: 160 y: 0
if < (Player 1) = 20 >
    say PLAYER 1 WINS for 2 secs
else
    say PLAYER 2 WINS for 2 secs
```

Conclusion:

After testing the game in multiple states, I found moving the ball and paddle 7 steps each time made for the best medium level gaming experience. I also found using the commands of the Z and X keys for player 1 and the 2 and 3 keys for player 2 the best commands because they are the opposite side of the keyboard and most user-friendly. I tested this with my parents and friends, and it was the best outcome each time. I used the mouse for player 1 and I found this to be the to difficult due to the sensitivity of the mouse and left one player depending on their experience at a huge advantage or disadvantage. Overall this was a success and I'm happy I made Pong as my application because now I have my own working version of one of my favourite childhood games.