CSE230 Final Project

Building a Pong Game with Haskell

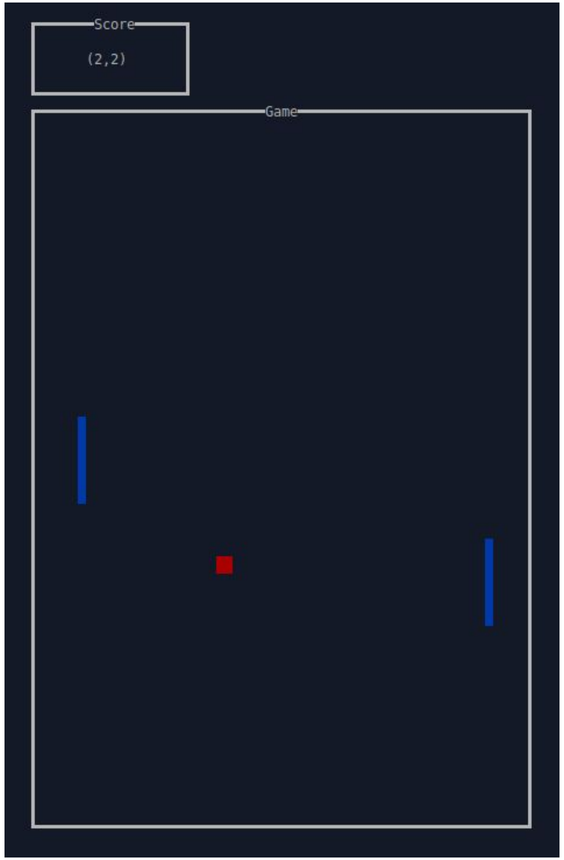
Group member:

Yuka Chu, Chi-Hsuan Lee,

Yi-Ting Wang, Heidi Cheng

Agenda

- Ideas
- Rules
- Program Architecture
- Demo & Testing
- Interesting Game Logic
- Difficulties
- Limitations



Ideas

- Goal: build a terminal user interface program
- Decided to build a TUI game
- Ping-Pong (Air Hockey)
- With some variation

Rules

- Two-player pong game
- Control vertical position of two rackets with keyboard
- One earns a point when the other player misses the ball
- The next ball is served towards the previous scored player
- The second ball is added after someone gets 3 points
- Game ends when one of the players hit a score of 5

Program Architecture

```
src
 ├── Model
 │    ├── Ball.hs
 │    ├── Player.hs
 │    └── Score.hs
 ├── Control.hs
 ├── Main.hs
 ├── Model.hs
 ├── Types.hs
 └── View.hs
```

Libraries: brick, vty
References: ranjitjhala/brick-tac-toe, samtay/snake

Program Architecture

```
data PlayState = PS
 { racket1     :: Racket       -- ^ racket on the left
 , racket2     :: Racket       -- ^ racket on the right
 , ball1       :: Ball         -- ^ properties of the ball1
 , ball2       :: Ball         -- ^ properties of the ball2
 , result      :: Maybe Turn   -- ^ game over flag
 , turn        :: Turn         -- ^ one of the player score, do nextServe.
 , score       :: Score        -- ^ score
 , secondBall  :: Bool         -- ^ whether the second ball has been added
 }
```

Program Architecture

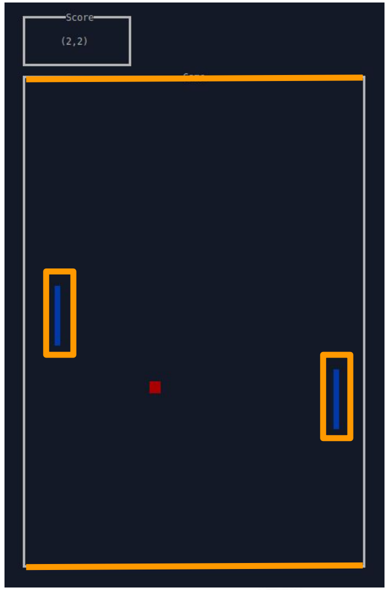
```
src
 ├── Model
 │    ├── Ball.hs
 │    ├── Player.hs
 │    └── Score.hs
 ├── Control.hs
 ├── Main.hs
 ├── Model.hs
 ├── Types.hs
 └── View.hs
```

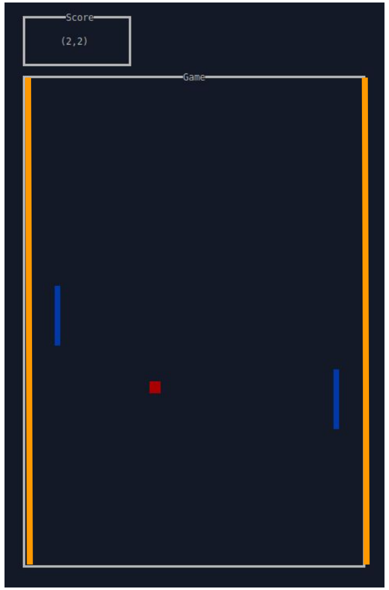
Libraries: brick, vty
References: ranjitjhala/brick-tac-toe, samtay/snake

Program Architecture

Ball reflects

The opponent scores





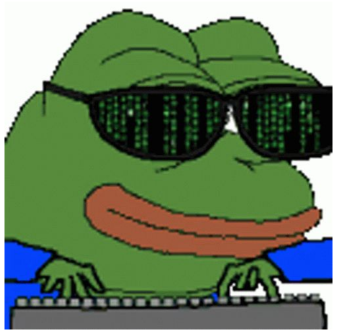
Program Architecture

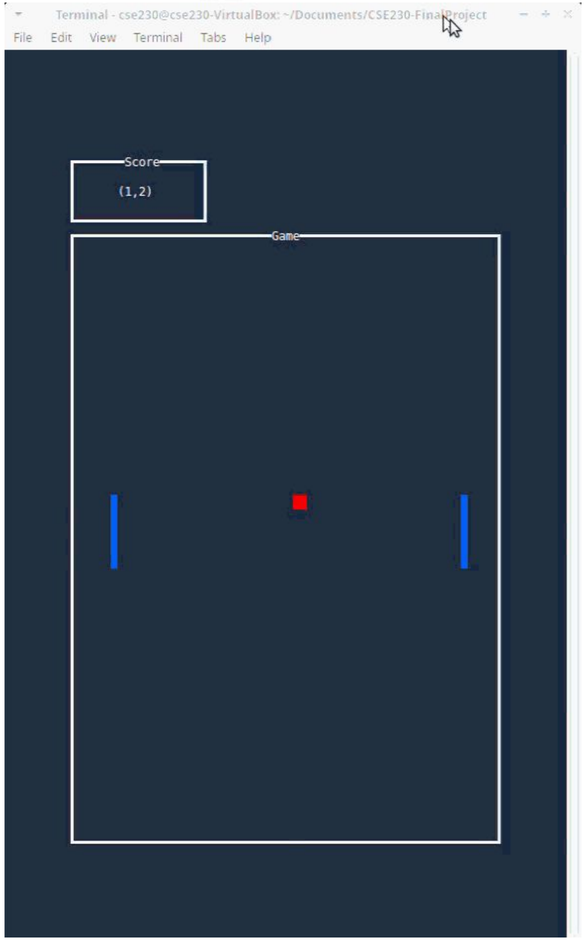
```
src
 ├── Model
 │    ├── Ball.hs
 │    ├── Player.hs
 │    └── Score.hs
 ├── Control.hs
 ├── Main.hs
 ├── Model.hs
 ├── Types.hs
 └── View.hs
```

Libraries: brick, vty
References: ranjitjhala/brick-tac-toe, samtay/snake

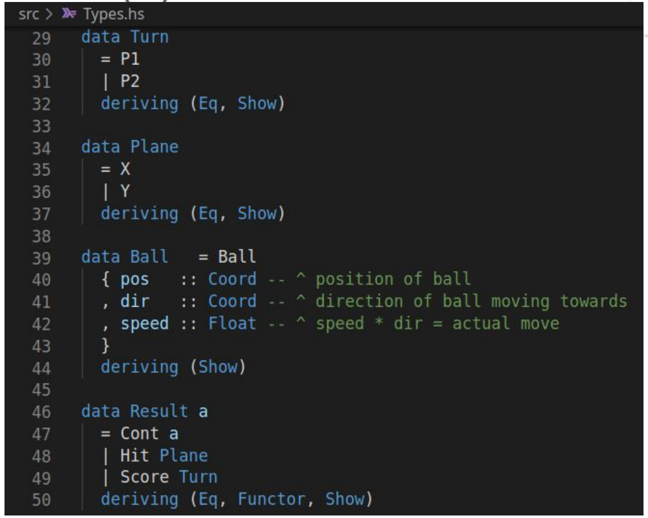
Demo & Testing

```
> stack run
```



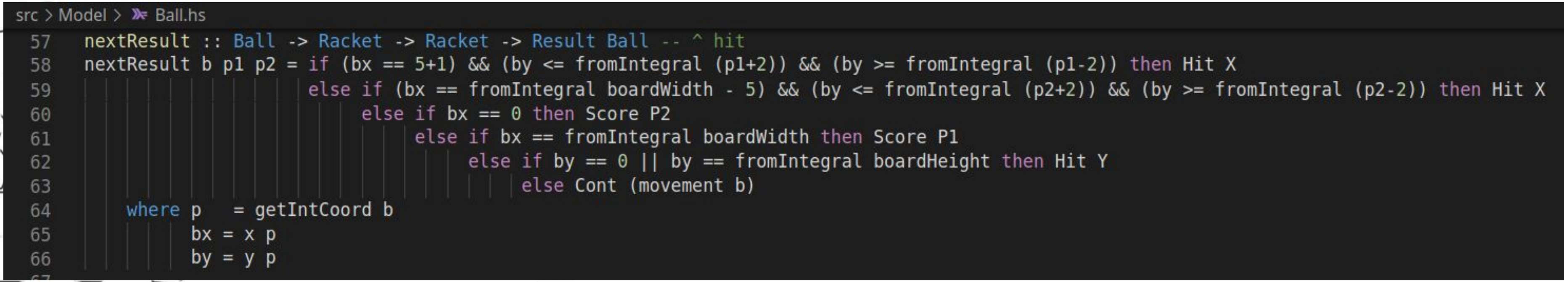

Terminal - cse230@cse230-VirtualBox: ~/Documents/CSE230-FinalProject
File
Edit
View
Terminal
Tabs
Help
Score
(1,2)
Game

Interesting Game Logic



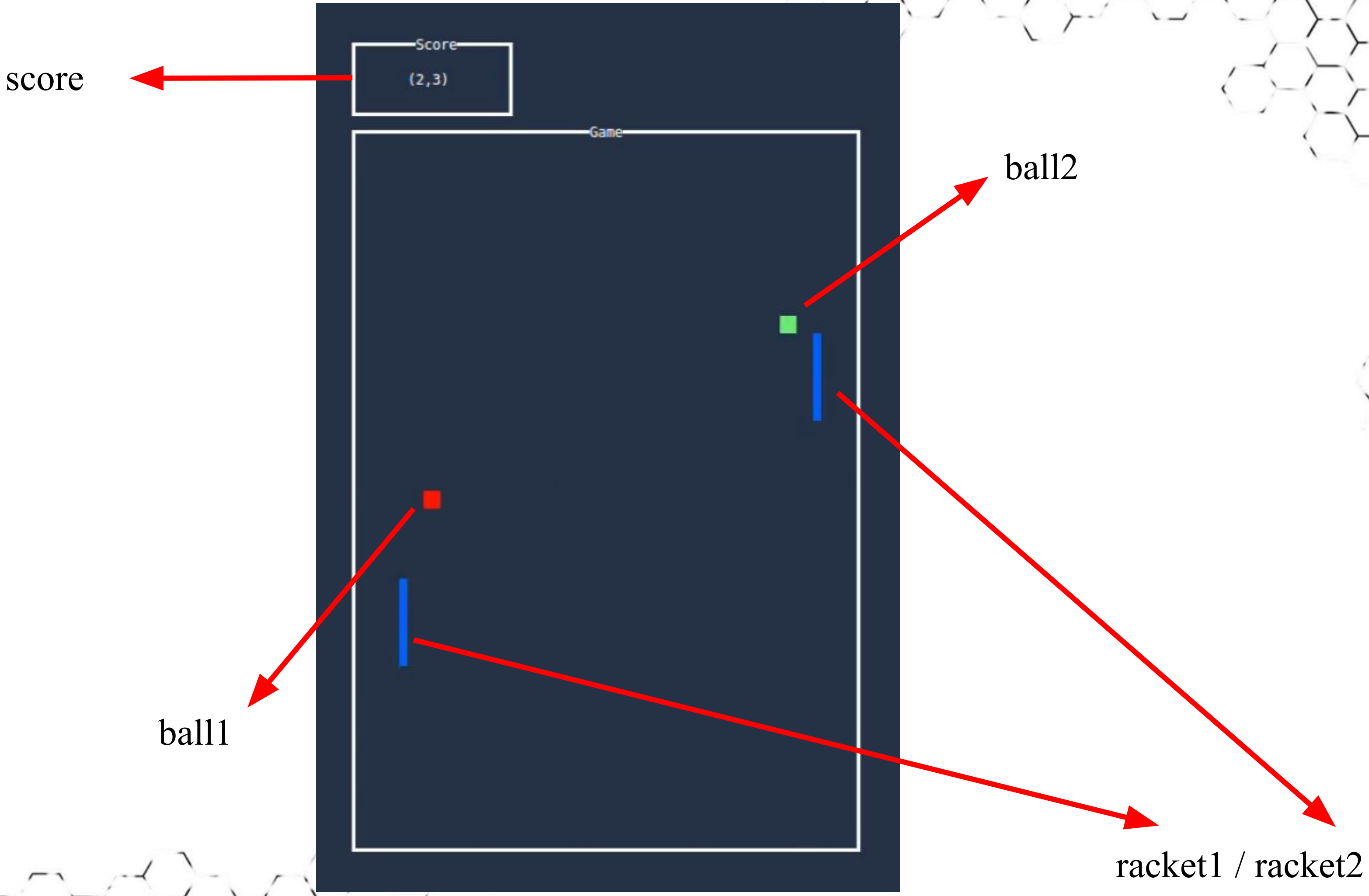
src > Types.hs

```haskell
data Turn
  = P1
  | P2
  deriving (Eq, Show)

data Plane
  = X
  | Y
  deriving (Eq, Show)

data Ball   = Ball
  { pos   :: Coord -- ^ position of ball
  , dir   :: Coord -- ^ direction of ball moving towards
  , speed :: Float -- ^ speed * dir = actual move
  }
  deriving (Show)

data Result a
  = Cont a
  | Hit Plane
  | Score Turn
  deriving (Eq, Functor, Show)
```



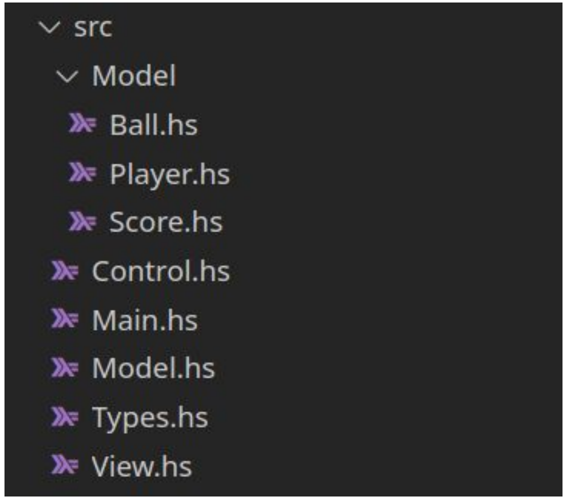
src > Model > Ball.hs

```haskell
nextResult :: Ball -> Racket -> Racket -> Result Ball -- ^ hit
nextResult b p1 p2 = if (bx == 5+1) && (by <= fromIntegral (p1+2)) && (by >= fromIntegral (p1-2)) then Hit X
                    else if (bx == fromIntegral boardWidth - 5) && (by <= fromIntegral (p2+2)) && (by >= fromIntegral (p2-2)) then Hit X
                        else if bx == 0 then Score P2
                            else if bx == fromIntegral boardWidth then Score P1
                                else if by == 0 || by == fromIntegral boardHeight then Hit Y
                                    else Cont (movement b)
   where p    = getIntCoord b
         bx = x p
         by = y p
```


score
Score
(2,3)
Game
ball2
ball1
racket1 / racket2

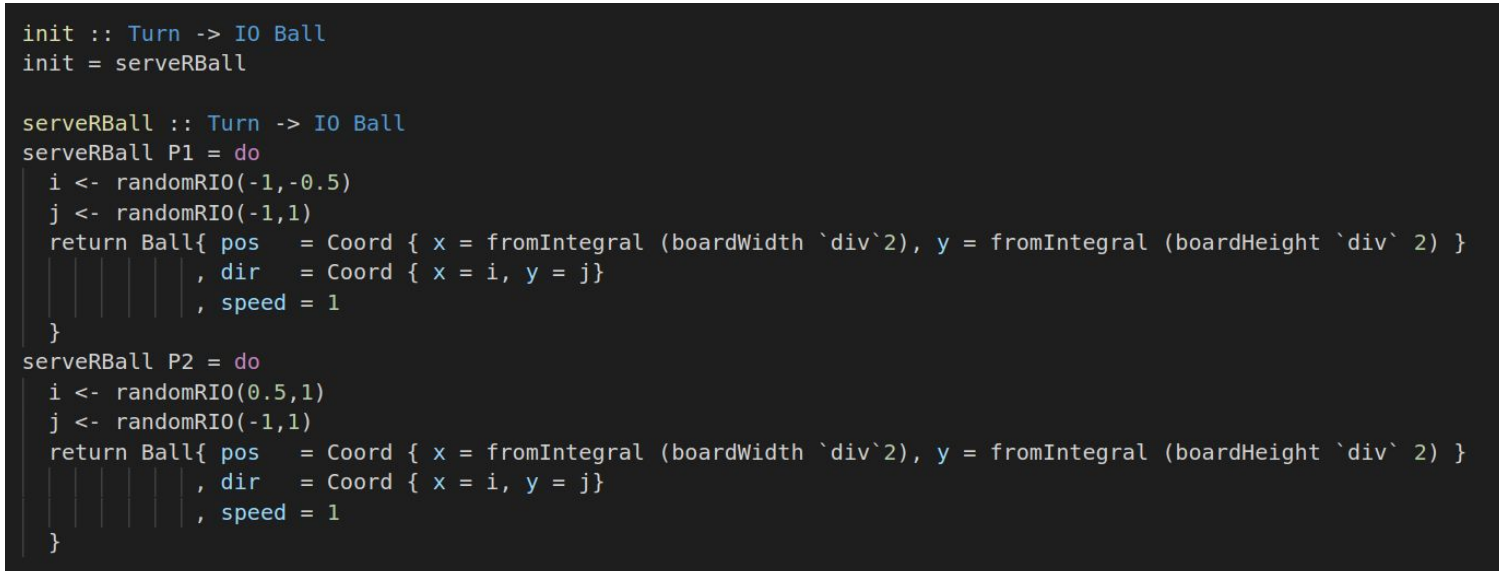
Difficulties

- How to assemble each part of our work and make the program executes correctly



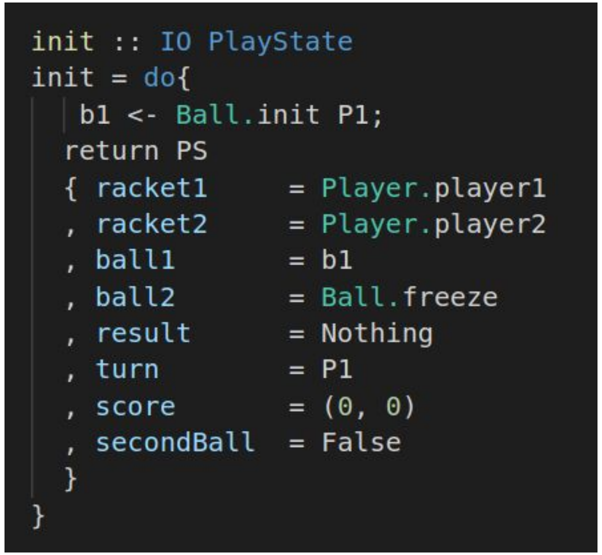
```
src
  Model
    Ball.hs
    Player.hs
    Score.hs
  Control.hs
  Main.hs
  Model.hs
  Types.hs
  View.hs
```

- To randomly serve balls, we had to deal with IO



```haskell
init :: Turn -> IO Ball
init = serveRBall

serveRBall :: Turn -> IO Ball
serveRBall P1 = do
  i <- randomRIO(-1,-0.5)
  j <- randomRIO(-1,1)
  return Ball{ pos   = Coord { x = fromIntegral (boardWidth `div`2), y = fromIntegral (boardHeight `div` 2) }
             , dir   = Coord { x = i, y = j}
             , speed = 1
  }
serveRBall P2 = do
  i <- randomRIO(0.5,1)
  j <- randomRIO(-1,1)
  return Ball{ pos   = Coord { x = fromIntegral (boardWidth `div`2), y = fromIntegral (boardHeight `div` 2) }
             , dir   = Coord { x = i, y = j}
             , speed = 1
  }
```

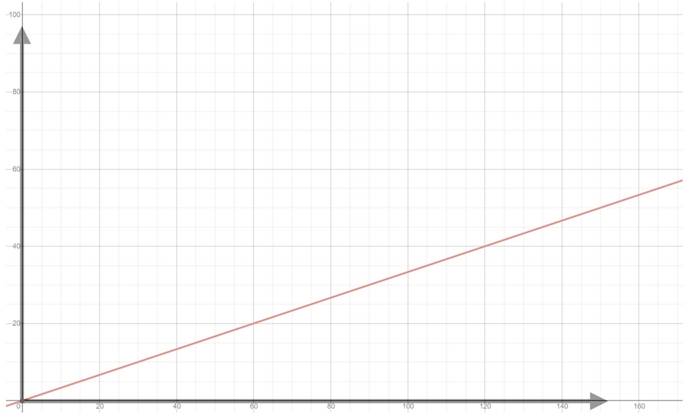
- We needed to deal with the movement of two balls separately and defined when to consider the second ball

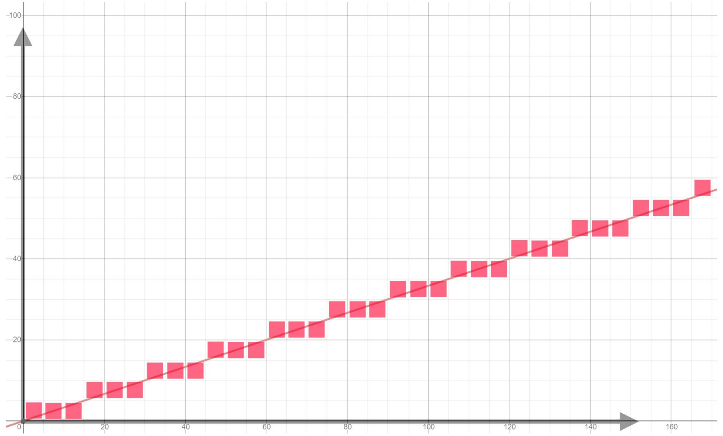


```haskell
init :: IO PlayState
init = do{
    b1 <- Ball.init P1;
  return PS
  { racket1     = Player.player1
  , racket2     = Player.player2
  , ball1       = b1
  , ball2       = Ball.freeze
  , result      = Nothing
  , turn        = P1
  , score       = (0, 0)
  , secondBall  = False
  }
}
```

Limitations

- Due to the nature of pixel games, it is inevitable that the ball moves discretely





Limitations

- Both players have to press and release the keyboard to move (when two players press to move at the same time, one player gets stuck)