Assignment 1 - Pong

By Kushagra Arora

1. **Player - Rectangle Drawn using Element Buffer Object**
   `drawPlayer(GLfloat x, GLfloat y, GLfloat z)`
   This function takes the coordinates of the center of the rectangle and creates a rectangle around it. SInce the player moves by left and right arrow keys. The element buffers and vertex buffers are created again and again for the player. When the user presses LEFT arrow key or RIGHT arrow key, the x coordinate of the rectangle is shifted accordingly.

   These changes are done using the function:
   `update(double delta, GLfloat &x)`

2. **Ball - Circle drawn using Triangle Fan**
   `drawShape(GLfloat x, GLfloat y, GLfloat z, GLfloat R, GLint numSides)`
   This function takes in the center of the shape and radius as the argument. The last argument - numSides specifies the number of sides of the shape. When numSides is large, the shape tends to be a circle. Since, the ball is moving at all time, we again need to generate the buffers inside the loop.

3. **Ball - Movement**
   The movement of the ball is defined using an angle *theta.* At every iteration of the loop, the `x` and `y` coordinates are changed.

   ```
   x += cos(theta) * 0.01f
   y += sin(theta) * 0.01f
   ```

   Here, 0.01f is the scaling factor since we want small smooth movements between iterations.
   Acceleration Idea (Not Implemented): To accelerate the ball as the game progresses, we can increase this scaling factor.

4. **Collision Detection**
   AABB collision detection is followed considering the ball and player to be rectangles. The collision is detected using a simple formula that simply checks if both x-coordinate and y-coordinate of the ball and player are overlapping.

   ```
   bool inXdirection = ballX + ballR >= playerX && (playerX +
   playerSizeX >= ballX  && playerX - playerSizeX <= ballX) ;
   bool inYdirection = ballY + ballR >= playerY && (playerY +
   playerSizeY >= ballY  && playerY - playerSizeY <= ballY);
   ```

5. **What happens when Collision is detected?**
   When there is a collision, the direction of the ball needs to be changed. The direction of the ball is governed by `theta.` So, we need to change the value of `theta`. For a perfectly elastic collision, we can say that the value of `theta` changes as follows:

   ```
   theta = 2*PI - theta
   ```

   However, if there is always a perfect collision then, the ball will only move along a few axes. To bring in variation in the game, I have calculated the point of contact of player and the ball. We say that the perfect collision happens if the ball hits the center of the rectangle. But on any side of the center, there is an offset in the value of theta bringing in other axes into picture.

6. **Score keeping:**
   Whenever there is a collision, the score is incremented by 1.

7. **Not implemented:**
   Due to lack of time, the walls haven't been built.
   Wall building Idea : Build walls using GL_QUAD of width 0.01f.
   Check collisions: Using the same idea of AABB collisions, we can detect collisions of the ball and the walls.
   Collision Effect: We will always have elastic collision with the walls and the ball.