MIS6060 Distributed Systems

Class Project - Group 3

Classic Ping Pong Game Using HTML & Javascript

Distributed Systems Features Demostrated: Fault Tolerance, Asynchronous communication, Replication, Rendezvous, Master-Slave Communication, UDP Data Format, Failure Masking,

Introduction

This paper will introduce a distributed system, implemented in the form of the classic **Ping Pong** game using the javascript and processing language. The game in question will implement two players (computer and user) hitting a ball on the screen (ping pong), back and forth across a virtual table using a racket represented with solid lines (further described below). The rules of the game are quite simple: each player (computer or user), must return a ball played towards them by the opponent. A point is scored when a player fails to return the ball within the rules, i.e. before it hits the end of the canvas (described below). The speed of the ball is increased sequentially as the game continues, demanding quicker reactions from both players.

Processing Language

Processing provides a graphical library and an integrated development environment (IDE). Sketches can draw two- and three-dimensional graphics. The default renderer is for drawing two-dimensional graphics. The P3D renderer makes it possible to draw three-dimensional graphics, which includes controlling the camera, lighting, and materials. This project and program made use of the P2D renderer. The P2D renderer is a fast, but less accurate renderer for drawing two-dimensional graphics. The capabilities of the game were extended with Processing Libraries and Tools. Libraries make it possible for sketches to do things beyond the core Processing code.

Game Requirements

The game's requirements are broken down into two distinct sections. The first section outlines the visual specifications needed to accurately depict the players, ball, and surroundings. The functional specifications for how the various objects and the application should function are covered in the second section.

Visual Requirements

- A rectangle will represent the player’s object.
- A circle will represent the ball object.
- The game will contain two text elements showing each player’s score.

Functional Requirements

- The player object should only be able to move on the y-axis.
- The ball object should be able to move on the x-, and y-axis.
- The ball object should change its direction on the x-axis to the opposite if it overlaps with a player object.
- The ball object should change its direction on the y-axis to the opposite if it moves out of the screen on the y-axis.
- The ball object's position and direction should be reset if it moves out of the screen on the x-axis.
- A player gets a point whenever the ball moves out of the screen on the opposite side.
- The first player will be controlled using the mouse and the second player by the computer.

Game Objects

Two distinct classes will be present in the game, one to represent a player object and the other to represent a ball object. A UML class diagram with the two classes is shown in Figure 1. A UML class diagram is just a picture of a system's classes, attributes, methods, and relationships for those of you who have never heard of one.

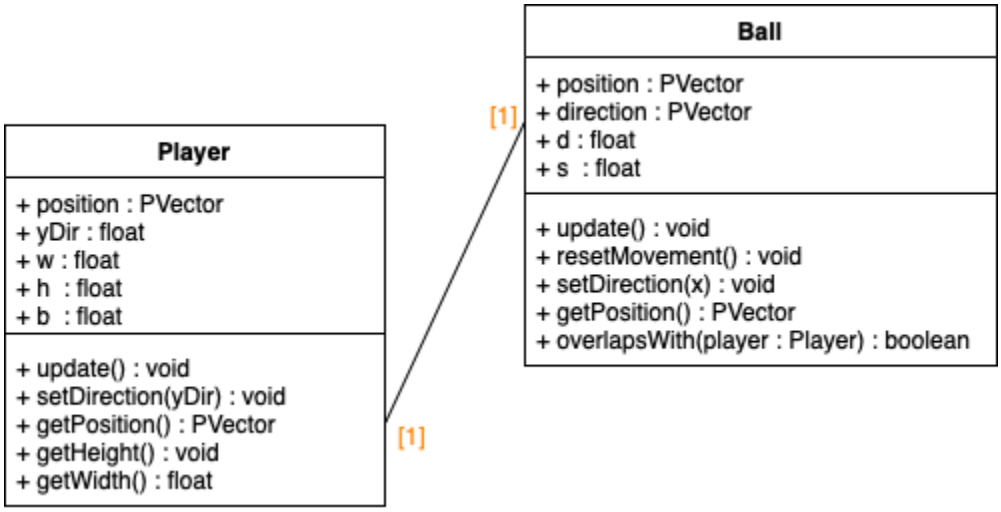

Figure 1: A UML Class diagram showing the two classes used in the Ping-pong game.

The Player Object

Although it may be argued that width, height, and the border could be static variables given that the values are the same for all objects, the player class will be used to generate objects for both players, making it crucial that the position and y-direction are distinct for each player.

Player Properties

The class requires five properties (see figure 1), and each of these properties has the following purpose:

- position — A PVector that defines the player's x and y position.
- yDir — A float that defines the direction and speed of the player's movement.
- w — A float defining the width of the player's rectangle.
- h — A float defining the height of the player's rectangle.
- b — A float defining the boundaries of the player's movement.

The implementation of the properties in the player class can be seen in figure 2 below.

```
29  // user paddle
30  const user = {
31      x : 0,
32      y : (canvas.height - 100)/2,
33      width : 10,
34      height : 100,
35      score : 0,
36      color : "WHITE"
37  }
38
39  // computer paddle
40  const com = {
41      x : canvas.width - 10,
42      y : (canvas.height - 100)/2,
43      width : 10,
44      height : 100,
45      score : 0,
46      color : "GREEN"
47  }
```

Player Methods

The class also requires five methods (see figure 1), and each of these properties has the following purpose:

update() — A void method that is used to update the behaviour of the player related to movement, boundaries check, and drawing of the visual object.

setDirection(yDir) — A setter method that is used to update the player object's direction on the y-axis.

getPosition() — A getter method that returns the player object's position.

getHeight() — A getter method that returns the player object's height.

getWidth()— A getter method that returns the player object's width.

The Ball Object

The ball class is very similar to the player class, but it is allowed to move on both the x- and y-axis.

Ball Properties

The ball class makes use of four properties which are described as follows:

position — A PVector that defines the ball's x and y position.
direction — A PVector that defines the direction and speed of the ball's movement.
d — A float defining the diameter of the ball's circle.
s — A float defining the start speed of the ball.

The implementation of the ball class can be seen in the figure below:

```
8   // Ball object
9   const ball = {
10      x : canvas.width/2,
11      y : canvas.height/2,
12      radius : 10,
13      velocityX : 5,
14      velocityY : 5,
15      speed : 5,
16      color : "WHITE"
17  }
```

Figure 3. The ball class implementation

Ball Methods

The ball class should also have five methods as the player class which can be described as follows:

update() — A void method updating the movement, boundaries, and visuals of the ball.
getMousePos() — A getter method that returns the ball's position.
resetBall() — A void method that reset the ball's position to the centre of the screen and resets its direction to a random value.
drawNet() — A void function that is used to draw both the use and computer paddles.
drawArc() — a coid method used to draw the circle representing the ball.
setDirection(x) —A setter method that sets the x-direction of the ball.
Overlaps With(player) — A boolean method returning true if the ball overlaps with the given player.

Collision Methods

Setting up Processing control flow

The application needs three Processing methods, the setup() method which is executed once the application starts (Processing.org, 2022a), the draw() method which is executed once every frame until stopped (Processing.org, 2022b), and the keyPressed() method which is executed when a key is pressed (Processing.org, 2022c).

Update

The update() method is used for the game's repeating logic. Line 122–120 in figure 4 shows an if- and else if-statement checking if the ball is outside the screen, and if that is the case, one of the players gets one point and the ball's position and direction are reset.

```
109  function update(){
110
111
112      if( ball.x - ball.radius < 0 ){
113          com.score++;
114          comScore.play();
115          resetBall();
116      }else if( ball.x + ball.radius > canvas.width){
117          user.score++;
118          userScore.play();
119          resetBall();
120      }
```