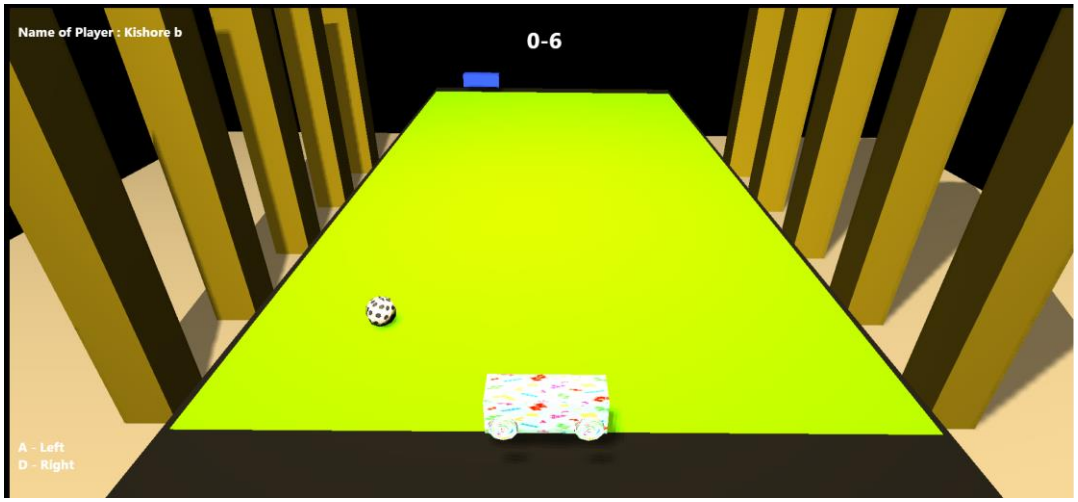
SAPIENZA
UNIVERSITÀ DI ROMA

PROJECT REPORT FOR INTERACTIVE GRAPHICS

Project Implemented – Ping Pong game

Instructor: Prof. Shaerf Marco



Group Members:

Kishore Bellada-1875384

July 13, 2019.

Table of Contents

Section A: Technical Presentation

Ping Pong game is one of the most famous indoor games and this inspired me to create a web version of it. The game starts with ball starting at randomly and rotating and moving from one side of table to another. To increase the user interaction I have various choices, so that user can choose, to enjoy his play. The game ends when either opponent or user scores the target points first.

1.0 DESCRIPTION OF THE ENVIRONMENT AND LIBRARIES USED IN THE PROJECT

The following libraries were used in the project:

ThreeJs: This was used to create and display animated 3D computer graphics in the web browser.

KeyboardJs: This was used to bind keys to perform actions on the scene.

Game.Js : The main file where game is setup and rendering occurs.

2.0 DESCRIPTION OF ALL THE TECHNICAL ASPECTS OF THE PROJECT

The game loads with Game.JS being the leading file for loading the scene and other models in the game. Keyboard.JS file mainly is used for binding the controls used to play the game. The step by step implementation from creating the dynamic world to starting the game is described as follows:

Creating the dynamic world

In order to create a dynamic world representation of the components used in the game scene (Ball and Paddles), I created a function call setup, which creates the background of the game, creates the table and add ball and paddles on it. The texture of the paddle, table and ball is chosen using a form at start of the game. The setup function is also used to load the name of

```
function setup() {
    score1 = 0
    score2 = 0
    document.getElementById("userName").innerHTML = "Name of Player : "+ (getUrlVars()["firstName"] + " "+ getUrlVars()["lastName"])
    getBackGrounds()
    createScene()
    draw()
}
```

Creating the scene rendering world

After we have the setup ready, the next task is of creating the scene, to load all the playable objects. The scene rendering is done using draw function. Before the draw function is called, the createScene function is called. Using this function, the scene is created with a particular camera angle. After the camera is set, the next important object is Plane and upon it the table. As the background is the created with idea of pillars and ground to have a feeling of playing a 3D game. We even add a ball and 2 paddles so as to start playing. The game starts with ball rotating on either side and rendering the ball rotation and paddle hitting the ball.

```
function draw() {
    renderer.render(scene, camera)
```

```
  requestAnimationFrame(draw)

  ballPhysics()
  paddlePhysics()
  cameraPhysics()
  playerPaddleMovement()
  opponentPaddleMovement()
}
```

Creating a Hierarchical Model

To add complexity to my models, I created a hierarchical model. My pallet consists of a cube geometry with fixed length breath and width. The paddle moves in right and left axis, which is positive or negative Y axis. I am adding 4 wheels on user paddle, 2 wheels on either side, which rotate as the paddle moves. The main motive of adding rotation is to create a hierarchical object to be added so that there is multiple rendering object. Once all the objects are created, at end the wheels are added by calling this function. The function creates 4 wheels, which are added at fixed location on paddle, with 3.5 as its radius and 0.8625 as its thickness, this gives the exact structure of the wheel with paddle. It also makes the user paddle look differently than computers. The wheel is created using Cylinder Geometry and having same texture like the paddle, and using a looping of position of wheels, we add the wheels on paddle, by making the wheels, child of the paddle.

```
var wheelRadius = 3.5;
  var wheelThickness = 0.8625;
  var wheelSegments = 10;
  var wheelGeometry = new THREE.CylinderGeometry(
    wheelRadius,    // top radius
    wheelRadius,    // bottom radius
    wheelThickness,  // height of cylinder
    wheelSegments);
  var wheelMaterial = new THREE.MeshPhongMaterial({map:paddleTexture});
  var wheelPositions = [
    [ -paddleHeight/3 ,(paddleWidth) ,  2 ],
   [ paddleHeight/3,-(paddleWidth),  2 ],
   [ paddleHeight/3,(paddleWidth) , 2],
   [ -paddleHeight/3,-(paddleWidth), 2],
  ];
  count=0
  var wheelMeshes = wheelPositions.map((position) => {
   var mesh = new THREE.Mesh(wheelGeometry, wheelMaterial);
   mesh.position.set(...position);
   mesh.rotation.z = Math.PI * .5;
   mesh.castShadow = true;
   mesh.name="wheel"+count
   count+=1
   paddle1.add(mesh);
   return mesh;
  });
  scene.add(paddle1)
```

Updating the dynamic world

When the ball moves, there is a need to calculate the dynamic motion of the balls. There are various ways I am checking the dynamic motion like, when the ball hits the wall or when it hits the paddle, it reverses in translation direction so as to get the effect of bouncing back. The function which manages this is ball physics and paddle physics. The code below shows the logic, which I used to reverse the position of ball when it hits the side wall.

```
if(ball.position.y <= -fieldHeight / 2) {
    ballDirY = -ballDirY
  }
  if(ball.position.y >= fieldHeight / 2) {
    ballDirY = -ballDirY
  }
```

Updating the Paddle Physics

When the ball is moving in the game scene, it is necessary to update the ball's position and orientation. The user and opponent moves the paddle accordingly. The ball orientation and position is calculated using the ballPhysics, where as the paddle movement is calculated using the playerPaddleMovement and opponentPaddleMovement functions. The main rendering of the paddle happens when ball hits the paddle, where it reverses back to opposite side, this task is handled by paddlePhysics function. This function checks the ball movement with respect to paddle, and when the ball touches the respective paddle, the paddle scales in y axis so as to show that ball has hit the paddle, and the ball traverse in opposite direction. Below code shows one section of that logic:

```
if(ball.position.x <= paddle1.position.x + paddleWidth &&
    ball.position.x >= paddle1.position.x) {
      if(ball.position.y <= paddle1.position.y + paddleHeight / 2 &&
        ball.position.y >= paddle1.position.y - paddleHeight / 2) {
          if(ballDirX < 0) {
            paddle1.scale.y = 15
            ballDirX = -ballDirX
            ballDirY = paddle1DirY * 0.7
          }
        }
    }
```

3.0 DESCRIPTION OF THE IMPLEMENTED INTERACTIONS

This section explains the interactions starting from the modelling structure used to the animations that was implemented in this project.

Hierarchical Model

To create a hierarchical model in the game, I have added wheels around the player paddle. There are 2 wheels added on both side, in total 4 wheels, which rotate as the paddle moves. The wheels rotate as the paddle moves left and right. The wheels rotation is done using rotateWheel function, which is called using playerPaddleMovement function, which makes the rendering having the hierarchical model to be called. The paddle and wheel is called hierarchical model because, the paddle is the parent of 4 wheels which are added to the paddle , and paddle is added to scene. Hence even the paddle acts as a parent of 4 wheels, which creates a hierarchy of model.

Lights and Textures

The game uses two types of lightning system, point and spotlighting, these 2-lightning mechanism help in easy highlighting the main focus of lights on ball, and table. The code below shows how I implement spot and point lightning:

```
pointLight = new THREE.PointLight(0xF8D898)
pointLight.position.x = -1000
pointLight.position.y = 0
pointLight.position.z = 1000
pointLight.intensity = 2.9
pointLight.distance = 10000
scene.add(pointLight)
spotLight = new THREE.SpotLight(0xF8D898)
spotLight.position.set(0, 0, 460)
spotLight.intensity = 1.5
spotLight.castShadow = true
scene.add(spotLight)
```

The textures that are used in various objects in the game are mainly using MeshPhongMaterial and MeshLambertMaterial materials. These materials help in loading the various texture in the objects in the game.

For the textures, various components were implemented:

Ball Texture: This is the game's ball texture which was loaded using THREE.ImageUtils. loadTexture() function with the image path (that contains the image to be used as the ball's texture) as its argument.

Player Paddle Texture: This is the user paddle texture which was loaded using THREE.ImageUtils. loadTexture() function with the image path (that contains the image to be used as the paddle texture) as its argument.

Plane Texture: This is the game's plane texture which was loaded using THREE.ImageUtils.loadTexture() function with the image path (that contains the image to be used as the plane's texture) as its argument.

Ground, Table and Pillar Texture: The texture is loaded by selecting solid colors, using the THREE.MeshLambertMaterial function, which takes as input colors and its corresponding values. The following lines show corresponding code:

```
var paddle2Material = new THREE.MeshLambertMaterial({
    color: 0x1B32C0
})
var paddle1Material = new THREE.MeshPhongMaterial({map:paddleTexture})
var planeMaterial = new THREE.MeshLambertMaterial({
    color: planeColor
})
var tableMaterial = new THREE.MeshLambertMaterial({
    color: 0x111111
})
var pillarMaterial = new THREE.MeshLambertMaterial({
    color: 0x534d0d
})
var groundMaterial = new THREE.MeshLambertMaterial({
    color: 0x888888
})
```

Player Interaction

The Login Page begins, with user giving a first name, which is mandatory, along with his last name and chooses the board color, paddle texture and ball texture before he begins the game. If the user doesn't feel like selecting any, he can play the game with default selections which are predefined. After the game starts, then ball starts moving accordingly towards one side of the board.

To navigate the paddle, two keys play a major role, mainly key "A" and "D" to move left and right accordingly. The 2 keys help in moving the paddle according to player input. The user presses the keys, depending on the ball movement. When the ball moves towards the paddle, the user will tend to get the paddle to hit the ball, eventually making the ball go in reverse direction. As the paddle moves, eventually the wheels attached to it also start rotating.

```
if(Key.isDown(Key.A)) {
    if(paddle1.position.y < fieldHeight * 0.45) {
        paddle1DirY = paddleSpeed * 0.5
    } else {
        paddle1DirY = 0
        paddle1.scale.z += (10 - paddle1.scale.z) * 0.2
    }
} else if(Key.isDown(Key.D)) {
    if (paddle1.position.y > -fieldHeight * 0.45) {
        paddle1DirY = -paddleSpeed * 0.5
    } else {
```

```
        paddle1DirY = 0
        paddle1.scale.z += (10 - paddle1.scale.z) * 0.2
    }
```

The player wins the game, if he scores 7 points before the opponent. The game doesn't end on draw. The game can be easily restarted using refresh button in browser page. The point is increased when a opposition is not able to hit the ball, and the ball passes after the paddle, which gives the player a point. Each point increases the chances of the player to win, until the player reaches 7 points first.

The next task was to reset the game after every time someone loses a game. This is done by restarting the game by loading the ball at center location and every alternate game, to move in a opposite direction, so its always equal number of chances to both the players.

```
function resetBall(loser) {
    ball.position.x = 0
    ball.position.y = 0

    if (loser == 1) {
        ballDirX = -1
    } else {
        ballDirX = 1
    }
    ballDirY = 1
}
```

Animations

The game has many animations which makes it an interesting game to play. I have created many animations for the game. Some of the animations include ball rotation, wheel rotation, paddle scaling, etc. I will be describing the animations which we have created in the game below:

Ball Movement and Rotation

Making the ball rotate was one of the important animations as it was vital to the fun of the game. I am not using any Physics library for rendering and making the ball move. I am using a basic logic, where in the position of the ball updates with ball speed and its direction. This gives us the movement required for making the ball move. But I also had to make the ball bounce from walls, paddles, and for that I also added various condition statement, which on checking those conditions, make the ball bounce back in the board. With all these logics, the ball would move/ translate but it was not rotating . Hence I added 4 lines for making it rotate:

```
if (rot>100)
rot=0.01
rot+=0.5
ball.rotation.y = rot
ball.rotation.z = rot
```

These lines the make ball rotate in y as well as z axis, giving the user the feeling as if the ball is rotating as it is moving. This rotation movement, is not affecting the translation of the ball, and hence as the ball translates, the rotation makes the ball look as if its moving because of rotation.

Paddle Scaling

To create a visual of ball hitting the paddle, I have implemented a scaling animation. The animation is activated when the ball hits the paddle, either player or opponent. Whenever the ball hits the paddle, the respected paddle scales itself in y axis to indicate the ball has been hit in the respected direction. The paddle also scales in Z axis, when the paddles reaches the end of the board.

Wheel Rotation

The Wheel Rotation animation is a hierarchical model animation. The wheels are child objects of its parent : user paddle object. The user paddle consists of 4 wheels, 2 on each side, and they rotate as the paddle is in movement. The rotateWheel function is used to rotating the wheel with a constant speed. The texture of the wheel is same as player paddle.

Section B: User Manual

This section describes the requirements before a player can play the game and also the instructions on how to play the game.

1.0 REQUIREMENTS

The player needs a web browser to run the game. The game has been tested on Google Chrome, Microsoft Edge, and Mozilla Firefox, and have been found to be successfully running without any glitches.

The player can run the game in 2 ways. The first step is a developer method, where player runs the game following the steps below:

Open cmd and traverse till the path of the game folder.

Start \python -m SimpleHTTPServer" in your shell (for Python 3.0 and above type \python -m http.server" in your shell)

Open browser and type : localhost:8080/

And these steps will help in loading the game locally. But if u intend to run the game online then you can use the following link for playing the game: https://kishorebelladasapienza.github.io/PingPongGame/

2.0 HOW TO PLAY

Keyboard Controls

The game is intended to attract young kids as well as current generation students. The game being easy to play doesn't have much trouble in learning it. The controls are very basic and has multiple controllers too.

The basic controllers being the arrow keys, i.e for moving left and right. The table below shows the respected keys for playing the game depending on the player's preference:

RIGHT	LEFT
d	a

Scorecard

The simplicity of the game makes the scoring simple as well. The player has the main task of winning the game, by making the opposition miss the ball hitting the paddle. The player receives a point every time, the opposition misses hitting the ball with the paddle. The player to score 7 points win the game.

3.0 TROUBLESHOOTING

The troubleshooting of the game can be done by:

Closing the terminal, and restarting the game.

Clearing your cookies and browser history, and then performing the steps for running the game.

References

http://threejs.org
https://en.wikipedia.org/wiki/Three.js