Project Report

Interactive Graphics

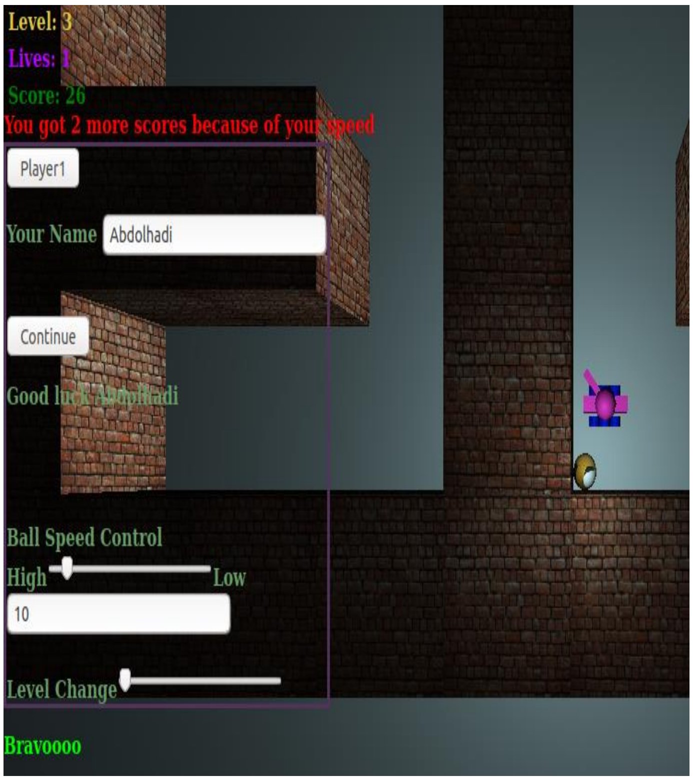
Bounce Game

Instructor: Prof. Shaerf Marco

Abdolhadi Rezaei

1837982

June 23, 2019.



Level: 3
Lives: 1
Score: 26
You got 2 more scores because of your speed

Player1

Your Name Abdolhadi

Continue

Good luck Abdolhadi

Ball Speed Control
High Low
10

Level Change

Bravoooo

Technical Presentation :

The game was inspired by a famous game called BOUNCE which was published and launched by Nokia.

The game developed in this project is similar to the original BOUNCE game and also simulates a lot of objects for gravity, friction, elasticity, joined bodies, collision, etc using some javascript libraries.
I have added also some functional for make it more user friendly, for example showing different messages in different situation.

Description of the environment and libraries used in the project :

I have used these libraries in the project :

- **ThreeJs**: This was used to create and display animated 3D computer graphics in the web browser.
- **Box2DWeb**: This is a 2D Physics Rendering Engine for creating a simulation of gravity, friction, collision, force etc.
- **JQuery**: This was used to perform HTML DOM Tree Traversal and Manipulation.
- **KeyboardJs**: This was used to bind keys to perform actions on the scene.

Description of all the technical aspects of the project :

The world is consist of the ball, maze, obstacles and pillet components in the game scene,In order to setup the dynamic world that would represent all mentioned objects I used ThreeJs (for rendering the scene) and Box2DWeb (to create a simulation of friction, density, force, restitution and collision between the ball and the maze). The step by step implementation from creating the dynamic world to starting the game is described as follows:

Creating the scene render world :

In order to render the game scene with all its components (light, camera, obstacle, pillet, ball, exit, maze and plane), I implemented a createSceneRenderWorld() function. This function is responsible for creating a ThreeJs Scene, creating a geometry of all the components (except light), assigns a texture to these components (except light), creates a ThreeJs Mesh of the components (except light) and places them at a predefined location in the rendered game scene. In case of the light, a white point light (using THREE.PointLight() function) is placed on the rendered game scene. A snippet of this function is shown below:

```
function createSceneRenderWorld() {
// Create the scene object.
scene = new THREE.Scene();
// Add the light.
scene.add(light);
// Add the Ball Mesh.
scene.add(ballMesh);
// Add the Obstacle Mesh
addObstacle();
//Addition of Pillet Mesh
addPillets(level/2);
//Adding the Exit Mesh
scene.add(exitMesh)
// Adding the Camera.
scene.add(camera);
// Adding the Maze Mesh.
scene.add(mazeMesh);
// Adding the Plane Mesh.
scene.add(planeMesh); }
```

Generating the maze mesh :

In order to generate a maze, I implemented a generate_maze_mesh() function. This function is responsible for creating a cube geometry using ThreeJs which accepts the field parameter (this contains a maze coordinate) to generate different cube geometries and merges these geometries to form a single maze.this function is shown below:

```
function generate_maze_mesh(field) {
var emptyGeometry = new
HREE.Geometry();
for (var i = 0; i < field.dimension;
i++) {
for (var j = 0; j < field.dimension;
j++) {
if (field[i][j]) {
var cubeGeometry = new
THREE.CubeGeometry(1,1,1,1,1,1);
var cubeMesh_ij = new
THREE.Mesh(cubeGeometry);

cubeMesh_ij.position.z = 0.5;

cubeMesh_ij.position.y = j;

cubeMesh_ij.position.x = i;

THREE.GeometryUtils.merge(emptyGeomet
ry, cubeMesh_ij);
            }
        }
    }
var material = new
THREE.MeshPhongMaterial({map:
meshTexture});
var mesh = new
THREE.Mesh(emptyGeometry, material)
return mesh;
}
```

Updating the dynamic world :

There is a need to update the ball's dynamic world properties when the ball position changes such as friction and force.The updateDynamicWorld() function updates these after taking a time step. A snippet of this function is shown below:

```
function updateDynamicWorld() {

document.getElementById("ballSpeedControl").onchange = function() {//For controlling ball speed
        ballSpeed = event.srcElement.value;

document.getElementById("txtBallSpeed").value=ballSpeed;
    };

document.getElementById("levelChange").onchange = function() {//For changing the level of game
            mazeDimension = event.srcElement.value;
            initializeGame();
    };
    // Applying Friction to the ball
    var velocity = newBall.GetLinearVelocity();
    velocity.Multiply(0.95);

newBall.SetLinearVelocity(velocity);

    // Applying Force to the ball controlled by User
    var force = new box2DVec2(animationAxis[0]*newBall.GetMass()*0.25, animationAxis[1]*newBall.GetMass()*0.25);
    newBall.ApplyImpulse(force, newBall.GetPosition());
    animationAxis = [0,0];

    // Taking a time step.
    newWorld.Step(1/ballSpeed, 8, 3);
    animatingObstacle()
}
```

As you see in this function I also implemented 2 events for changing the speed of the ball and the level of the game.

Updating the scene render world :

I have to update the ball’s position and orientation when the ball is moving in the game scene.I also need to check if the ball has collided with an object or picked a pillet in the scene. Why the collision of the ball with an object is important?because it determines the lives and reward that would be assigned to the player playing the game. Finally, I then update the camera and light’s position as the ball rotates in the scene. The updateSceneRenderWorld() function was implemented for this purpose.This function can be seen below:

```
function updateSceneRenderWorld() {
    // Updating Ball position.
var xIncrement = newBall.GetPosition().x - ballMesh.position.x;
var yIncrement = newBall.GetPosition().y - ballMesh.position.y;
ballMesh.position.x += xIncrement;
ballMesh.position.y += yIncrement;
actualBallPosX = Math.round(ballMesh.position.x*100)/100
actualBallPosY = Math.round(ballMesh.position.y*100)/100
    // Updating Lives and Score
checkBallCollisionWithObjects(actualBallPosX,actualBallPosY);
if (lives == 0) {
        $('#gameover').show()
        $('#try').show()
        $('#panelControl').hide();
        $('#help').hide();
        $('#level').hide();
        $('#lives').hide();
        $('#score').hide();
        $('#totalscore').html('Total Score: ' + score).show();
        $('html, body').css({
            overflow: 'hidden',
            height: '100%',    });
        scene.remove(planeMesh);
        scene.remove(light);
        scene.remove(ballMesh);
        scene.remove(exitMesh);
        scene.remove(mazeMesh);
        scene.remove(camera);  }
    // Updating Ball rotation.
var tempMat = new THREE.Matrix4();
tempMat.makeRotationAxis(new THREE.Vector3(0,1,0),xIncrement/ballRadius);
tempMat.multiplySelf(ballMesh.matrix);
ballMesh.matrix = tempMat;
tempMat = new THREE.Matrix4();
tempMat.makeRotationAxis(new THREE.Vector3(1,0,0), -yIncrement/ballRadius);
tempMat.multiplySelf(ballMesh.matrix);
ballMesh.matrix = tempMat;
ballMesh.rotation.getRotationFromMatrix(ballMesh.matrix);
// Updating Camera Position and Light Position.
camera.position.x += (ballMesh.position.x - camera.position.x) * 0.1;
camera.position.y += (ballMesh.position.y - camera.position.y) * 0.1;
camera.position.z += (5 - camera.position.z) * 0.1;
light.position.x = camera.position.x;
light.position.y = camera.position.y;
light.position.z = camera.position.z - 3.7;
}
```

Game State Animation :

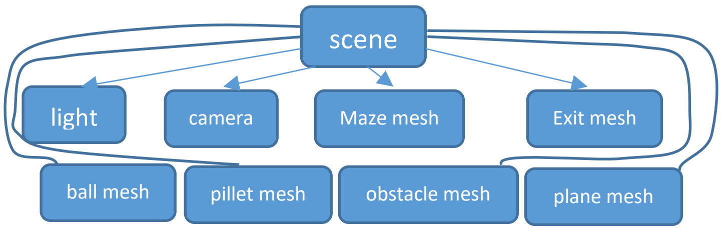
I first initialize the game scene (using the initializeGame() function) in order to start the game, setup the game scene (setupGameScene() function) before actually playing the game (playGame() function). There are different levels in the game and transition between levels is done by the requestAnimationFrame() function. All the functions described in the game animation section was implemented except the requestAnimationFrame() function that is inbuilt in ThreeJs. A snippet of the gameAnimation() function is shown below:

```
function gameAnimation() {

    if (gameState == "initialize"){
        initializeGame();
    }
    else if (gameState == "setup"){
        setupGameScene();
    }
    else if (gameState == "play"){
        playGame();
    }
    else if (gameState == "next"){
        nextGameLevel()
    }

requestAnimationFrame(gameAnimation);
}
```

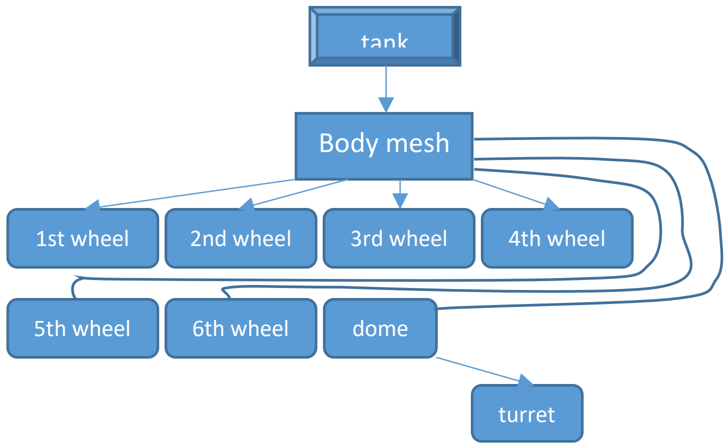
Hierarchical Models :

There are 2 different hierarchical model as below :

I modeled the game using hierarchical modeling according to the structure below. I used Scene.add() function (into the ThreeJs library) to add all these components to the Scene in a hierarchical manner.



I have created the tanks in a hierarchical way. I have considered the tank as the root and make the body attached to it. Other parts which include six wheels and a dome are the children of bodyMesh and the turret is the children of dome.



Textures :

6 components were implemented in order to make the texture:

1. ballTexture : 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.
2. obstacleTexture:This is the game's obstacle texture which was loaded using THREE.ImageUtils.loadTexture()function with the image path (that contains the image to be used as the obstacle's texture) as its argument.
3. planeTexture :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.
4. meshTexture :This is the game's maze texture which was loaded using THREE.ImageUtils.loadTexture() function with the image path (that contains the image to be used as the maze texture) as its argument.
5. pilletTexture :This is the game's pillet texture which was loaded using THREE.ImageUtils.loadTexture() function with the image path (that contains the image to be used as the pillet texture) as its argument.
6. exitTexture :This is the game's exit texture which was loaded using THREE.ImageUtils.loadTexture() function with the image path (that contains the image to be used as the exit texture) as its argument.

The interaction between the player and the game :

For moving the balla round the maze I have implemented the onMoveKey() function which detects what key direction that was pressed by the player.

```
function onMoveKey(axis) {
    animationAxis = axis.slice(0);
}
```

I have bounded the arrow keys to the keyboard.

```
KeyboardJS.bind.axis('left', 'right',
'down', 'up', onMoveKey);
```

Depending on the key that was pressed, a force is applied to the ball to move it in that particular direction.

```
var force = new
box2DVec2(animationAxis[0]*newBall.Ge
tMass()*0.25,
animationAxis[1]*newBall.GetMass()*0.
25);
```

```
newBall.ApplyImpulse(force,
newBall.GetPosition());
```

A player has 3 lives in the game and I have considered the score in this manner :

- 16 scores after passing each level
- 2 more scores for whom pass each level with high speed(under 15)
- - 5 points for colliding with an obstacle

In order to update the player's points and lives I have implemented the function checkCollisionWithObstacle() that detect a collision with obstacle.

```
function
checkCollisionWithObstacle(ball_Pos_X
,ball_Pos_Y){
    if (obstacleLocation.length<=0)
        return false;
    for( i =
0;i<obstacleLocation.length;i+=2)
    {
if((ball_Pos_X<=(obstacleLocation[i]+
0.3) &&
ball_Pos_X>=(obstacleLocation[i]-0.3)
)      &&
(ball_Pos_Y<=(obstacleLocation[i+1]+0
.3) &&
ball_Pos_Y>=(obstacleLocation[i+1]-
0.3)))
         return true
    }
    return false;
}
```

If it returns true,the player's points is reduced by 5 and lives is reduced by 1. Afterwards, his score and lives would be updated. The player would also have to restart the current level because of the collision. If the lives has been reduced to zero, then the game is over and the total score is shown.I also checked if the player picked a pillet with the ball. If he does, he is rewarded with 8 points and his overall score is updated.
Finally, if the ball gets to the exit of the maze, the scene moves to the next level.

```
function nextGameLevel(){
    scene.remove(exitMesh)
    updateDynamicWorld();
    updateSceneRenderWorld();
    light.intensity += 0.07 * (0.0 -
light.intensity);
    renderer.render(scene, camera);
    if (Math.abs(light.intensity -
0.0) < 0.1) {
        light.intensity = 0.0;
        renderer.render(scene,
camera);
        gameState = 'initialize'
    }
}
```

The movement of the ball and rotation :

One of the important animations is the ball rotation, I have used the Box2DWeb library for making the ball rotate and move ahead. The game starts with ball at location <1,1> which is our initial position for all levels, and when the player loses a life. I have created ball mesh with the initial position. After the initial position, when the player presses the required keys set up for the game, I get the velocity using the GetLinearVelocity() function and get it multiplied with a constant of 0.95 to keep the movement steady. This velocity is the new position of our ball and when subtracted with old position gives us the original position of the ball. I have also considered the force, which is applied when the player presses the key to keep the ball moving, this is also calculated by using the Box2DWeb library. Once I find the updated dynamics for movement, I then make the ball rotate using the following code:

```
var tempMat = new THREE.Matrix4();
tempMat.makeRotationAxis(newTHREE.Vec
tor3(0,1,0), xIncrement/ballRadius);
tempMat.multiplySelf(ballMesh.matrix);
ballMesh.matrix = tempMat;
tempMat = new THREE.Matrix4();
tempMat.makeRotationAxis(new
THREE.Vector3(1,0,0), -
yIncrement/ballRadius);
tempMat.multiplySelf(ballMesh.matrix);
ballMesh.matrix = tempMat;
ballMesh.rotation.getRotationFromMatr
ix(ballMesh.matrix)
```

Removal of Pillet on Ball Collision :

Each even level of the game consists of half the amount of pillets.When an even number of level is loaded, addPillet() function is called which calculates and stores the locations for adding the pillet in an array called pilletLocation. When the ball reaches a pillet, the location of ball is matched
with the pillet location to check whether the ball has touched the pillet, this task is done by pickingPilletByBall() function. If the ball location is found to be at pillet location, then this function returns the array index of the pillet, which helps the game in choosing which pillet to be removed. By using the getChildByName() function, I achieve that particular pillet mesh, by providing the pilletMesh + its index location as its name. This helps in finding the pillet to be removed. The location of the pillet in array pilletLocation is made to -1 for that level and the pillet is removed from scene, simultaneously the score is updated with 10 more points.

ExitMesh Animation :

The ExitMesh Animation consists of decreasing opacity of exit mesh as the ball reaches the end of the maze. The mesh opacity continuously ecreases as the ball reaches the end of maze upto value of 0.2 and it becomes 1 the moment ball leaves and enters the maze again. The exit mesh is removed from scene once the ball exits the maze.

Wheels rotation :

I make the wheels rotate using following loop. Using wheelObject, I get each wheel and rotate it by 0.08 in x direction

```
function rotateWheel(bodyObject)
{
    for(s=0;s<6;s++)
    {
        wheelObject =
bodyObject.getChildByName("wheel"+s)
        last_rot =
wheelObject.rotation.x
        last_rot+=0.08
        if (last_rot>15)
            last_rot=0
wheelObject.rotation.x=last_rot;
        }
    return;
}
```

Turret rotation :

To make turret rotate, I use annimatingTurret() function. In this function, I make the turret rotate by 0.09 each time in y direction.

```
function
animatingTurret(obstacleObjectMesh)
{
  bodyObject =
obstacleObjectMesh.getChildByName("bo
dyMesh")
    turretPivot =
bodyObject.getChildByName("turretPivo
t")
    prev_rot = turretPivot.rotation.y
    prev_rot+=0.09
    turretPivot.rotation.y=prev_rot
    return;
}
```

References :

https://api.jquery.com/

https://github.com/hecht-software/box2dweb

http://threejs.org/

https://github.com/wwwtyro/Astray-2

https://www.w3schools.com/html/html_css.asp