SPIN: A 2D PHYSICS-BASED GAME

submitted in partial fulfillment of the requirements

in

CE812-7-SP : Physics-Based Games

by

ISHWAR VENUGOPAL

(1906084)

School of Computer Science and Electronic Engineering
University of Essex

April 2020

CONTENTS

1. GAME DESCRIPTION

1.1 Synopsis

'Spin' is a 2-dimensional physics-based game that provides a simple and addictive experience for the user, by means of guiding a ball to a flagged destination. The current work is inspired from the original game with the same name, which is available online [2] for free. The original game is focused on implementing minimalist controls, and achieving a cross between Pinball and a 2-dimensional golf game. A sample gameplay video is available on YouTube [1]. Screenshots from the original game is shown in Fig:1.1, which has been used as an inspiration to create the environment and controls of the game presented

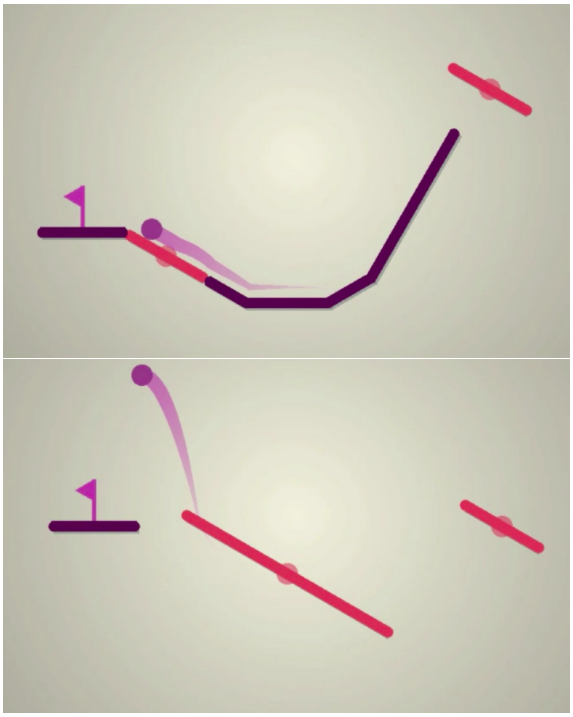

Figure 1.1: Screenshots from the original game which was used as an inspiration

in this report.

This game falls in the category of Arcade Puzzle games. The game mainly consists of controllable paddles which can be rotated either clockwise or anti-clockwise using the left and the right arrow keys. The main objective of the game is to guide the ball present on the screen towards the area denoted by a green flag. If the ball lands on any of the triangle-shaped 'thorns' during the course of the game, the player fails. Also the game ends if the ball goes out of the screen area. A pop-up message appears when the player either wins or loses the game. A reset button is available to restart the game from its initial configuration.

The game becomes interesting and challenging due to the fact that when the same amount of rotational force is applied to paddles with different lengths, they behave differently. As the arrow keys apply a fixed value of rotational force to all the paddles equally, the player has to find the optimum balance to control the ball as it reaches different parts of the screen. Fig:1.2 and Fig: 1.3 shows the screenshots during different trials of the game.

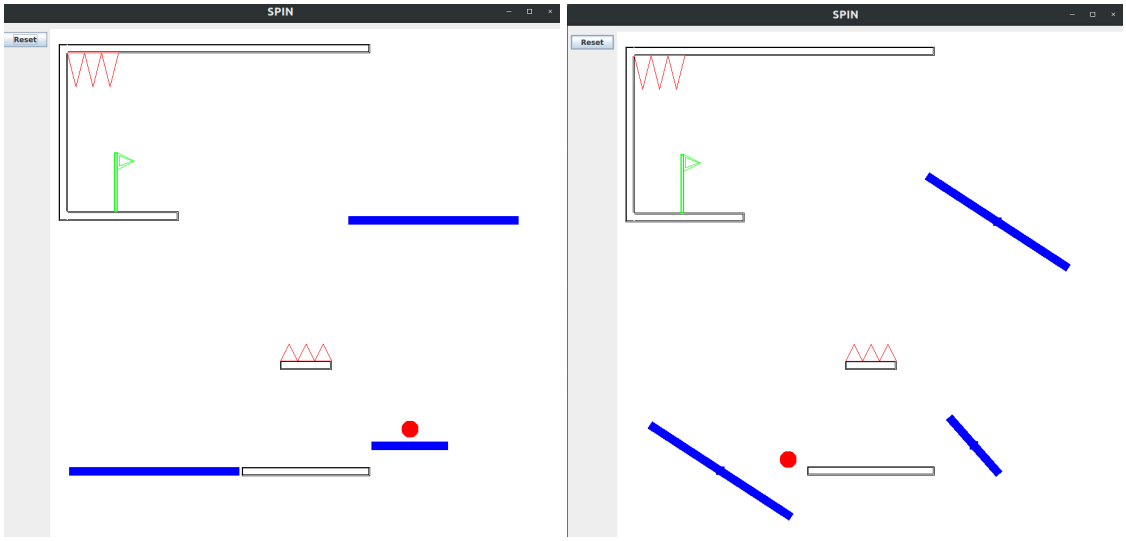

Figure 1.2: (Left) Initial configuration when the game starts, or when the user clicks on 'Reset' , (Right) A screenshot during the game

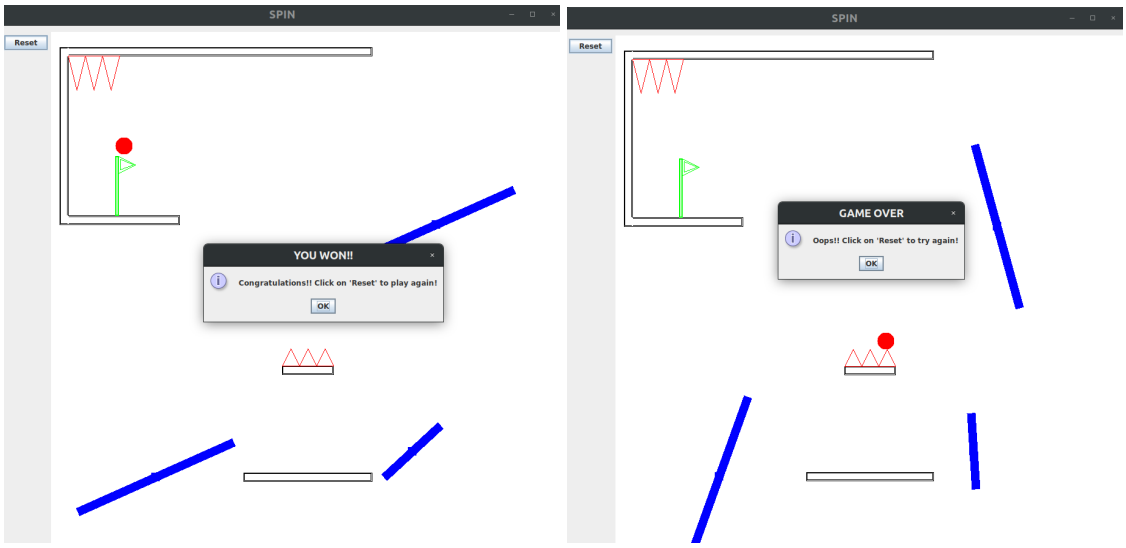

Figure 1.3: A screenshot when the player wins (Left) and when a player loses (Right)

1.2 Player instructions

- Click on ’Reset’ to begin.
- Use the left and the right arrow keys on the keyboard, to control the motion of the paddles. Left arrow key makes the paddle rotate anti-clockwise and the right arrow key makes the paddle rotate clockwise.
- The player loses if the ball goes out of the screen area, or if it lands on any of the triangle-shaped thorns.
- The player wins when the ball reaches the area marked by the green flag
- Click on ’Reset’ anytime to start from the initial configuration

2. TECHNICAL ISSUES

2.1 Physics Engine

In this work, the physics engine used is the open-source 2D Physics Engine called JBox2D [3]. It is the Java port of the original C++ version (Box2D). This physics engine was chosen as it could implement most of the features required for this project in the best possible way. The major features of JBox2D implemented for this project are as follows:

- Torque: The keyboard listener was linked to the application of torque of a fixed amount to each of the paddles. This caused a rotational motion corresponding to the length and mass of the paddles. The paddles were described as rigid bodies with different mass and dimensions.

```
        if (BasicKeyListener.isLeftArrowPressed()){
            body.applyTorque(1000);
        }
        if (BasicKeyListener.isRightArrowPressed()) {
            body.applyTorque(-1000);
        }

```

- A combination of STATIC and DYNAMIC body types were used to fix the paddle in space and to facilitate its rotation. Also, the ball becomes a STATIC object with a fixed position in space after the player fails. A function was created which does this job, only when the player fails in the game. The function is as follows:

```
        public void makeBallStatic(float x, float y) {
            this.body.setTransform(new Vec2(x,y),0);
            this.body.setType(BodyType.STATIC);
```

```
    }

```

- The paddle is connected to a small STATIC block in space through a Revolute joint. This facilitates the smooth control of the paddle and also makes it look hooked to a point in space. Three such paddles with different dimensions were placed in space [Fig:1.2].

- The condition for victory and loss was decided based on the position of the ball. If the ball travelled somewhere very close to the boundary of the thorns or if it moved away from the WORLD, the condition for loss was activated and a pop-up screen was made to appear acknowledging the defeat. Similarly, a region around the flag was identified for determining the victory and the position of the ball in this region was monitored. [Refer Appendix for the code]

Different concepts of physics that were discussed in the lectures, were implemented in this game. Some of the notable physics concepts that were implemented were:

- **Vectors**: Vectors have been used throughout the program using Vec2 class of JBox2D, to represent position and velocity of the different objects used. Objects of different classes like 'BasicParticle', 'BasicRectangle' and 'Controllable Rectangle' were created, having a position and velocity vector for each of them. Also during the application of the rolling friction force, a scalar multiplication with a vector is also implemented.

- **Centre of Mass**: The concept of Centre of Mass was important in the context of paddles, to identify the correct location to place the anchor for the Revolute joint. It was necessary to understand the centre of mass of a rectangular block, as the main objective was to rotate it around an axis perpendicular to its surface.

```
   //Lower left flipper

       rectangles.add(new BasicRectangle(WORLD_WIDTH/4 -
  WORLD_WIDTH/16 + 0.5f ,WORLD_HEIGHT/8 + 0.25f,0,0,0.5f,0.5f,
  Color.BLUE,10,rollingFriction,false,0.9f));
```

```
4        rectangles.add(new ControllableRectangle(WORLD_WIDTH/4 -
   WORLD_WIDTH/16 + 0.5f ,WORLD_HEIGHT/8 + 0.25f,0,0,10,0.5f,
   Color.BLUE,5,rollingFriction,true,0.9f));
5
6        BasicRectangle p1 = rectangles.get(0);
7        BasicRectangle p2 = rectangles.get(1);
8        RevoluteJointDef jointDef=new RevoluteJointDef();
9        jointDef.bodyA = p1.body;
10       jointDef.bodyB = p2.body;
11       jointDef.collideConnected = false; // this means we don't
   want these two connected bodies to have collision detection.
12       jointDef.localAnchorA=new Vec2(0,0); // the coordinates of
    the pivot in bodyA
13       jointDef.localAnchorB=new Vec2(0,0); // the coordinates of
    the pivot in bodyB
14       world.createJoint(jointDef);
15
```

- **Moment of Inertia and Torque**: As the paddles had different dimensions and mass, it was important to identify the correct combination of the values that would create a similar effect when a torque of equal magnitude was applied to all of the paddles. The optimum number was identified by considering the fact that torque was directly proportional to the Moment of Inertia as well as the mass. Calculated guesses were made to come up with the final values for the dimensions of the paddles. The final values magnitude of the torque used was 1000. The larger paddle had a length of 10 units, width of 0.5 units and a mass of 5 units. The smaller paddle had a length of 4.5 units, width of 0.5 units and a mass of 60 units.

- **Forces**: A rolling friction force is applied to the ball in every time step. This is calculated by multiplying the linear velocity of the ball with it's mass.

```
1    Vec2 rollingFrictionForce=new Vec2(body.getLinearVelocity());
2      rollingFrictionForce=rollingFrictionForce.mul(-
   rollingFriction*mass);
3      body.applyForceToCenter(rollingFrictionForce);
4
```

- **Angular Damping**: Angular damping was an important concept to implement in this game as we didn't want the paddle to rotate continuously when the arrow key is pressed once. Due to this fact, the constructor of the 'ControllableRectangle' was made in such a way that it also considers value for angular damping as an input, among other parameters. A trial-and-error approach was followed until a desirable behaviour was obtained.

- **Coefficient of Restitution**: To make the game a little interesting and difficult to play, it was taken care that the ball doesn't bounce too much when it hits the paddles. This was achieved by controlling the coefficient of restitution of the rectangles. The coefficient of restitution was kept as 0.5 units.

- **Transforms**: A linear transform was applied to the ball, to place it statically at a desired position after the game ends.

```
1    this.body.setTransform(new Vec2(x,y),0);
2
```

2.2 Graphics

- The components of the game, like the platforms, flag and the thorns were created using the AnchoredBarrier_StraightLine class. Different shapes were created by aligning them as closed figures and defining them in an anti-clockwise fashion. Each compound shape that appears on the screen is a well-structured combination of straight lines.

- **Pop-up screens** were implemented to create an interactive atmosphere when the player wins or loses the game [Fig:1.3]. This was achieved by using the JOptionPane function in the Java.Swing class.

- A square screen of dimension 840 units was used. Delta T used in this game is 0.002.

3. PERSONAL APPRAISAL

This module was very interesting to me as I was always fascinated by the concepts in Physics and how well they relate to the things we experience in real-life. Incorporating concepts from physics into games was a challenging task to me in the beginning, as I had very little experience with developing games in Java. Most of my previous works were focused on programming languages like Python and C++. I developed all my skills required for this project through the completion of the lab sessions, which in itself posed a new challenge to me every week.

I chose to work on this particular game for my project as it was a simple implementation of basic physics concepts in an engaging and addictive way. I believe that I have developed and used a good amount of skills in implementing concepts of physics using Java in this project, which I wouldn't have been able to when the module had started. I am happy with the fact that I was able to develop this into a game with proper pop-ups and other additional factors, which I wasn't able to do in any of the lab sessions.

My lack of experience with using Java might have been the reason that I was only able to make a game with a single level. I believe that with further practise and experience, I will be able to develop games consisting of multiple levels too.

BIBLIOGRAPHY

[1] Spin! - a very satisfying physics-based paddle-spinning blend of 2d golf pinball! https://www.youtube.com/watch?v=C4I1hY2yd3g.

[2] Spin! - download game. https://www.freegameplanet.com/spin-download-game/.

[3] D Murphy. Jbox2d. https://github.com/jbox2d/jbox2d.

Appendices

A. CLASSES

The program consists of the following classes (The classes which were written completely from scratch or developed by making significant changes to previously existing classes are written in **bold**):

- AnchoredBarrier
- AnchoredBarrier_StraightLine
- BasicKeyListener
- **Basic Particle**
- **BasicRectangle**
- BasicView
- **ControllableRectangle**
- **Gameplay**
- JEasyFrame
- **PopUpScreen**
- ThreadedGuiForPhysicsEngine

A.1 Newly created classes

Out of the classes highlighted in **bold**; BasicRectangle, ControllableRectangle and PopUpScreen were written from scratch. The class Gameplay was modified from the BasicPhysicsEngineUsingBox2D class, that was originally used in the lab sessions.

A.2 BasicParticle

There were a few additions to the BasicParticle class that was used in the lab sessions. The code snippet with the major additions that were made is given below:

```
//Additional functions added to the original class //

public boolean gameover=false;

public boolean checkGameover() {

  //Checking world boundary conditions

  if((body.getPosition().x < 0)||(body.getPosition().x > WORLD_WIDTH))
    gameover=true;
  if((body.getPosition().y < 0)||(body.getPosition().y > WORLD_HEIGHT))
    gameover=true;

  //Checking thorn in the middle

  if((body.getPosition().x + 0.5f >= WORLD_WIDTH/2 - 1.5f)&&(body.getPosition().x + 0.5f <= WORLD_WIDTH/2 + 1.5f)&&(body.getPosition().y >= WORLD_HEIGHT/2 - WORLD_HEIGHT/6 + 0.5f)&&(body.getPosition().y <= WORLD_HEIGHT/2 - WORLD_HEIGHT/6 + 1.5f))
      gameover=true;

  if((body.getPosition().x - 0.5f <= WORLD_WIDTH/2 - 1.5f)&&(body.getPosition().x-0.5f >= WORLD_WIDTH/2 - 1.5f)&&(body.getPosition().y >= WORLD_HEIGHT/2 - WORLD_HEIGHT/6 + 0.5f)&&(body.getPosition().y <= WORLD_HEIGHT/2 - WORLD_HEIGHT/6 + 1.5f))
    gameover=true;

  if((body.getPosition().y - 0.5f <= WORLD_HEIGHT/2 - WORLD_HEIGHT/6 + 1.5f)&&(body.getPosition().y - 0.5f >= WORLD_HEIGHT/2 - WORLD_HEIGHT/6 + 0.5f)&&(body.getPosition().x >= WORLD_WIDTH/2 - 1.5f)&&(body.getPosition().x <= WORLD_WIDTH/2 + 1.5f))
```

```
23      gameover=true;
24
25    //Checking thorn at the top
26
27    if((body.getPosition().y + 0.5f >= WORLD_WIDTH*0.97f - 2.5f)&&(body
      .getPosition().y + 0.5f <= WORLD_WIDTH*0.97f - 0.5f)&&(body.
      getPosition().x >= 1f)&&(body.getPosition().x <= 4f))
28      gameover=true;
29
30    if((body.getPosition().x - 0.5f <= 4f)&&(body.getPosition().x - 0.5
      f >= 1f)&&(body.getPosition().y >= WORLD_WIDTH*0.97f - 2.5f)&&(body
      .getPosition().y <= WORLD_WIDTH*0.97f - 0.5f))
31      gameover=true;
32
33    return  gameover;
34  }
35
36  public boolean victory=false;
37
38  public boolean checkVictory(){
39
40    if((body.getPosition().x + 0.5f >= WORLD_WIDTH/8 - 2f)&&(body.
      getPosition().x + 0.5f <= WORLD_WIDTH/8+1.2f)&&(body.getPosition().
      y >= WORLD_HEIGHT/2 + WORLD_HEIGHT/8 + 0.5f)&&(body.getPosition().y
       <= WORLD_HEIGHT/2 + WORLD_HEIGHT/8 + 4f))
41      victory=true;
42
43    if((body.getPosition().x - 0.5f <= WORLD_WIDTH/8+1.2f)&&(body.
      getPosition().x-0.5f >= WORLD_WIDTH/8 - 2f)&&(body.getPosition().y
      >= WORLD_HEIGHT/2 + WORLD_HEIGHT/8 + 0.5f)&&(body.getPosition().y
      <= WORLD_HEIGHT/2 + WORLD_HEIGHT/8 + 4f))
44      victory=true;
45
46    if((body.getPosition().y - 0.5f <= WORLD_HEIGHT/2 + WORLD_HEIGHT/8
      + 4.1f)&&(body.getPosition().y - 0.5f >= WORLD_HEIGHT/2 +
      WORLD_HEIGHT/8 + 0.5f)&&(body.getPosition().x >= WORLD_WIDTH/8 - 2f
      )&&(body.getPosition().x <= WORLD_WIDTH/8+1.2f))
47      victory=true;
```

```
48
49      return victory;
50    }
51
52    public void makeBallStatic(float x, float y) {
53      this.body.setTransform(new Vec2(x,y),0);
54      this.body.setType(BodyType.STATIC);
55    }
56
```

B. GITLAB LINK

The complete code along with the README file can be accessed through GitLab via the following link: `https://cseegit.essex.ac.uk/2019_ce812/ce812_venugopal_i/-/tree/master/FinalProject`.