UNIVERSITY OF ESSEX

CE812

PHYSICS-BASED GAMES

SAFE-DRIVING

GAME DEVELOPED IN JBOX2D

Submitted by:

Vandana Sharma 1807152

ABSTRACT

This report is used to describe the game “SAFE-DRIVING” developed in java using JBox2D libraries. This game is about moving a car to reach its destination without getting hit by obstacles. All these objects and obstacles are run under physics world to make them act like the original world.

CONTENT

INTRODUCTION

This Report is about the A game “Safe driving” which is develop in java using JBox2D libraries for implementing physics in the game. In this game we have a car like object make up of different object and a world for that car to move and reach its destination safely.

While developing this game we studied about Box2D and JBox2D as JBox2D is ported form Box2D. We have explained the features of Jbox2D in the report that how they work. And the attributes of JBox2D libraries. We have also defined the methods which can be used while developing the game.

In next section we have explained about the game we have developed so far using JBox2D libraries. We have described formation of the objects and properties used into them and the use of those used methods. We discussed about the world we used in this game, Also the working of each objecting the game simulation.

At last we have given idea about the extension of this game. If we have more time to learn more about the JBox2D features in detail the there are many changes, we can do to make game more interesting.

Box2D & JBox2D

Box2D is an open source library for simulating bodies in 2-Dimentions written in portable C++ computer programming language [1]. It is developed by Erin Catto and released as open source on September 11, 2007 as a part of GDC presentation. It is used to develop games like Crayon Physics, Angry Bird, Limbo, Rolando, Happy Wheels and more. This engine is distributed under the zlib license and is thus available on the internet as free software. Box2D is also been ported to other platform like Flash, C#, Javascript, Java [2]. Box2D is ported in Java is known as JBox2D [3]. It was designed as a combination of Box2D and LiquidFun physics engine [4].

LiquidFun

LiquidFun is a Box2D - based 2D rigid body and fluid simulation library C++. It Created by Google as a Box2D extender with additional soft body and particle - based fluid simulation [4]. LiquidFun make object move and act realistically by providing animation support to physical bodies [4].

JBox2D is like Box2D, as it is a ported version of Box2D in Java. There main difference is about objects which are supported by one library but not by other. For example, making of liquidFun in JBox2D by using small particles and connecting them together which change shape under pressure like water and other liquids [5].

As these libraries are keep on updating. We are using latest version of JBox2D which is having similar objects and other feature similar to Box2D latest version which were not present in last version of JBox2D like Rope joint, motor joint, and as box2Dc can create edges, last version of JBox2D was able to create special type of polygon shape not edge shape.

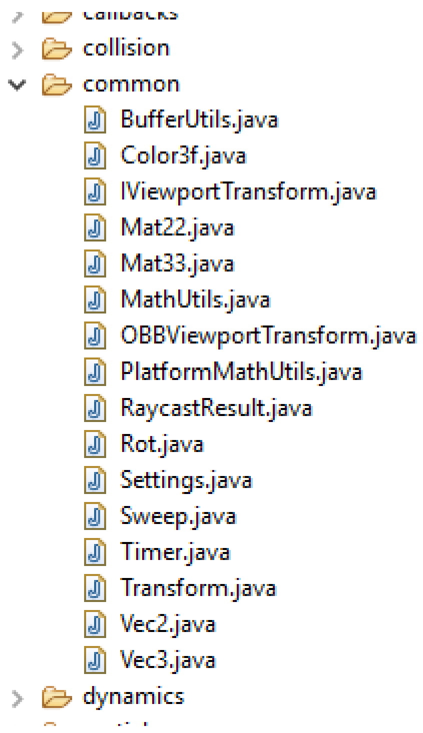
JBox2D also use other features like serialization. When the program that created it ends, a Java object normally expires. Since no variable refers to it, its storage is retrieved by the garbage collector [6]. Problem is when we want to make an object persistent to save it between executions of the program. Serialization is the mechanism of transforming an object into a byte stream for storing or transmitting the object to memory, database, or file [6].

JBox2D FEATURES

JBox2D works on several concepts and objects. Like using rigidbody, shape fixtures etc. JBox2D generate rigid body using its shape, bind in a shape using fixture properties such as density, friction and restitution. Fixture make shape collide with other shapes in the world [2].

Common

It contains setting, memory management and vector math [1].

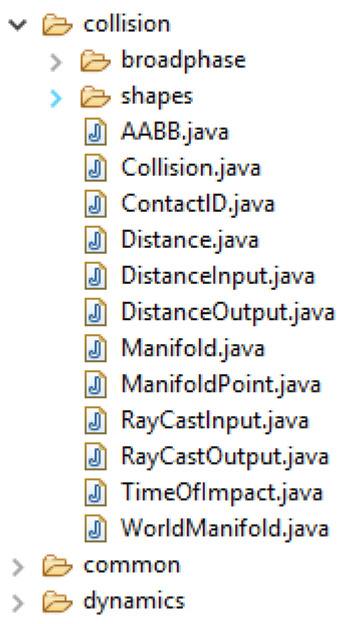


```
> callbacks
> collision
v common
      BufferUtils.java
      Color3f.java
      IViewportTransform.java
      Mat22.java
      Mat33.java
      MathUtils.java
      OBBViewportTransform.java
      PlatformMathUtils.java
      RaycastResult.java
      Rot.java
      Settings.java
      Sweep.java
      Timer.java
      Transform.java
      Vec2.java
      Vec3.java
> dynamics
```

It contains settings with different values that represent some maximum variables and constants. JBox2D use units like kg, meters, second not pixels [2]. In code, mostly variables are represented in floating point numbers instead of using double. Angles are calculated in radians and classes are in Vec2, Vec3, Mat22, Mat33. And we have used Vec2 only in our application as us was 2D. Vec3 is for 3D vectors similarly Mat2 and Mat3 are used to perform vector math using matrix module. It also keeps track of the positions of all the shapes which are moving in world using memory management.

Collision

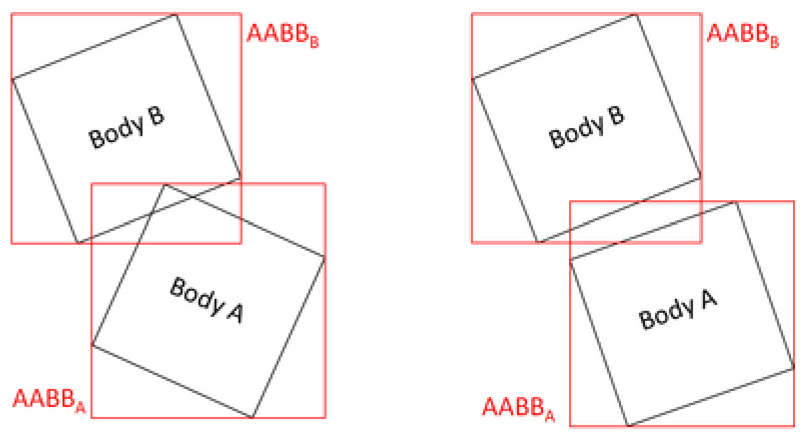
Collusion module contains shape and their functions used by them.



JBox2D detect collision between the shape even before they collide using its collision detection. The collision detection system calculates when and where the collision occurs and the normal vector between the touching objects [7]. It computes the point of contact and store in manifold. Then that data used by contact solver to calculate the collision. JBox2D calls a method to receive contact or collision point between the shapes. This method known as narrow-phase and this is called for each particle collides in the scene every time. To reduce the numbers of narrow phase calls, JBox2D use dynamic tree which organize large numbers of shapes efficiently. Class does not work on shape; it works on axis-aligned bounding boxes (AABB). In collision module of JBox2D it calculates distance between 2 shapes and time of their impact.

AABB:

Dynamic tree uses the AABB method to detect the objects instead of using shape.



Shapes

Shape is unknown from knowledge of bodies and stand apart from the dynamics system [1]. Shapes in JBox2 are not easily moved, we need to move them manually by setting vertices position. To make body movement in the world we need to attach it with the fixture.

JBox2D supports circular, polygonal, edge and chain shape.

Circle Shape:

CircleShape circleShape = new CircleShape();

circleShape.m_radius = radius;

Circle shape can be defined by radius and position.

Polygon Shape:

In JBox2D, Polygon shape can be define in more than one way. Like

PolygonShape shapesss = new PolygonShape();

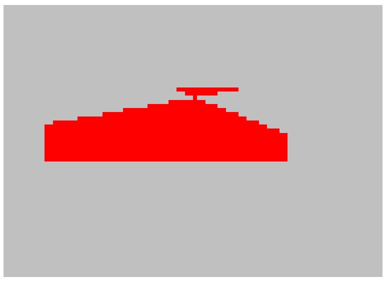
shape.setAsBox(width, height);

Above method is it set polygon shape setting as a box with width and height. In this method JBox2D will measure the side of polygon from centre and then both values will be half.

PolygonShape shape = new PolygonShape();

shape.set(vertices, vertices.length);

for convex polygon shape other than rectangle we can draw by providing array of three or more than three vertices. To make a convex polygon vertex define in the game should be in anti-clockwise direction. Drawing polygon in wrong direction may cause undesirable behaviour.



PolygonShape shape = new PolygonShape();

shape.setAsEdge(vertex1, vertex2);

In some versions of the JBox2D, above method is use to create edge between vertices. But in latest version there is a function edge. which we have used in the game as platform for the car movement.

Chain shape:

In the game we have used then chain shapes as platform for the car movement.

ChainShape chainShape = new ChainShape();

chainShape.createChain(vertices, vertices.length);

The chain shape provides an efficient way to build your static game worlds by connecting many edges together.

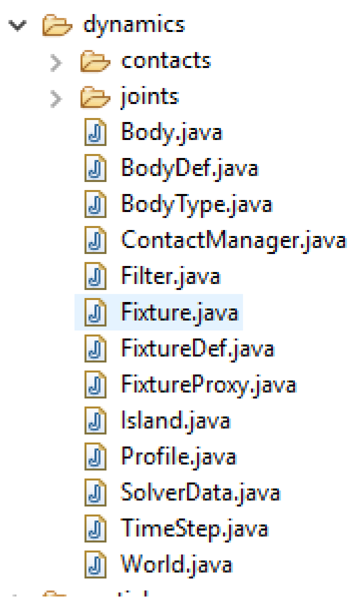
we can also use edge shape for the same.

Edge Shape:

They are line segments which can collide with all shapes except themselves. This is because as collision algorithm used in JBox2D requires at least one shape must have volume and as they are just line segments, they don't have volume, so they don't collide with themselves.

Dynamic Body

It is the most complex part of the JBox2D.



The Dynamics module contains:

- fixture class
- rigid body class
- contact class
- joint classes
- world class
- listener classes

Bodies:

BodyDef bodyDef = new BodyDef ();

bodyDef.type = BodyType.*STATIC*;

bodyDef.position = new Vec2(startx, starty);

Body is consisting of position and velocity, we can apply force, torque and impulse on body [1]. Body can be of three types STATIC, DYNAMIC AND KINEMATIC. STATIC bodies are those which do not move with any kind of force applied on them under the simulation and behaves like the have infinite weight. They can be move manually. KINEMATICS are those who moves under simulation according to its a velocity and if any force applied on them and Dynamic bodies are those which are fully simulated, they can be move manually as well but normally they move according to force on them [1], like randomly falling of object from top of the screen. Two fixtures attached to the same rigid body never move relative to each other and there is no collision between the fixtures attached to the same body [1].

We can create a body by defining its positions, angle, velocity. You can set body's position for run time by setting as position of body and with velocity you can set linear velocity and angular velocity of a body which work with the starting of simulator. Beyond this, there are linear and angular damping. Damping is way of losing velocity of an object like in friction but damping loses velocity without collision unlike friction.

In body we can make an object keep on rotation with fixed velocity in the simulation by making Fixed Rotation of object true.

By making change in gravity scale of object we can make object use no gravity or inverse. We can also change bodies sleeping parameters.

If we want a body not to take part in any collision or simulation that's we can make that body off by using

bodydef. active = false;

there is one more feature of body is "userData". It is a void pointer. It can be used to call data of a body in any class.

bodyDef.bullet = true;

With above function, collision of body with other can be set as fast like bullet.

Fixtures:

One of the important parts of body is Fixture. A body can have multiple fixtures also. Fixture can be defined as:

```
FixtureDef fd = new FixtureDef();
fd.shape = circleShape;
```

There are some properties of fixture and those can be defined as:

```
fd.density = 1;
fd.friction = 0.3f;
fd.restitution = 0.5f;
fd.isSensor = false;
fd.filter = filter;
```

- Density: Its computer mass property of object. It can't be negative.
- Friction: As we know friction is a resistance force applied on object in opposite direction of there movement to stop their movement. It is used to reduce velocity of object. Its value can be between 0 to 1. If friction value is 0 the it means, there is no friction on body. And if two objects collide and have friction, then square root of both objects will applied.
- Restitution: It is bounciness of body. It is like friction, if restitution is 0 body will not bounce back.
- Filter: it is a method to make object pass through each other without any collision.
- Sensor: If sensor is true then it makes collision detection with actual collision. It is useful in the collision of complex shapes like concave.

Contacts:

When collision of objects occurs then contact occur between them. JBox2D calculate the collision between the objects in two ways. First method is to search all the contacts from the contact list and use accordingly or we have other method to do that which is using the contact listeners. In which we have four functions.

Begin Contact: It starts when two fixtures start to overlap.

End Contact: it starts at the end of contact.

Two others are call-back functions

Pre-solve event is called before the collision is resolved so that the velocities of fixtures can be changed, or the contact can be completely disabled. Post-solve event occurs when a collision is resolved and contains information about the impulses of the collision.

World:

```
public static final World WORLD = new World(new Vec2(0.0f, -9.8f ), false);
```

Whole game scene will be represented by the world, where other objects will fill gravity and all other forces and act like in real world. World is consisting of two values as define above, gravity and sleep.

```
world.Step(timeStep, velocityIterations, positionIterations);
```

The world defines time step for running a simulation that how often simulation should call.

```
float32 timeStep = 1.0f / 60.0f;
```

The function also has two attributes to set the accuracy of collision:

```
int32 velocityIterations = 6;
int32 positionIterations = 2;
```

AABB query:

This allow you to make rectangular box for collision detection. It is good method to use for concave shapes. It is done by setting lower and upper bound. And need call-back method to detect collision happen.

Ray Casting:

Ray Casing uses lines to detect the object they hit. Like AABB query, they also call call-back function you as the collision occur. This function contains four attributes that are fixture that got hit, point of contact, normal and frictional distance. Call-back return float value which is length of ray. Return 0 cancel the ray casting and return 1 is the original length of ray.

Impulse, torque and forces:

As bodies moves in the world with gravity, so they get effected by everything like impulse and torque like if we apply on any rotating body, angular velocity other forces.

Joints:

Joint are important part for a game, they help to attach objects to each other and reduce their degree of freedom. Joints use anchor point to connect bodies to each other. Joints and some attributes like Anchor point of A and B objects, angle limits. There are different types of joints.

- Prismatic joint, while preventing rotation, allows two connected bodies to move along one axis.
- Revolute joint allows the rotation of two connected bodies around one point.
- Distance joint join two bodies that are defined by a space between them. Even they are at distance their distance between the anchor point should be constant.
- Gear joint can connect more than on joint together to make them affect each other.
- Wheel joint is combination of revolute and prismatic joint. It works perfectly for wheel to make wheel move up and down on bumps and same time rotate to make wheel move forward in the direction.
- Mouse joint is used make object attached to mouse and make them move.
- Pully joint is used to make pully from two body by connecting them through pully. They are also connected to the ground as well.
- Weld joint is to make different shapes from the basic shapes and prevent motion of objects with respect to each other.
- Rope joint connect two bodies together to make chain like structure at the same time It doesn’t allow to go far from maximum distance from each other but can mover freely like a chain.
- Motor joint set position and angle of body which are the limit where body tries to reach. It can have limited torque and force.

GAME APPLICATION

This game is a demonstration of using JBox2D for game development. Game is about the moving objects attached to each other by joints and act like car in the world. And the motive of object is to reach its destination without getting hit or collide with the obstacles. This game is developed using many objects and attributes.

JAVA libraries:

As we know JBox2D does not support any graphical user interface, so to represent our simulator we need to use some external libraries. Which is used to create applet in the game where we run the Game. In this game we have used external libraries to create applet which is divide into panel, where there is one button and text to represent the win and lose, and other side panel to make game simulation. Which is drawn in the “GameView.java”. This java file is used to excess draw() function of all objects. Which help in drawing all the objects. And by using button we call all the functions again and reset all values using threads.

WORLD:

To create world, we need to set gravity. For this game have set gravity value in BasicPhysicsEngineUsingBox2D.java. We are using gravity as 9.8.

```
public static final float GRAVITY=9.8f;
```

and constant set as a 0.

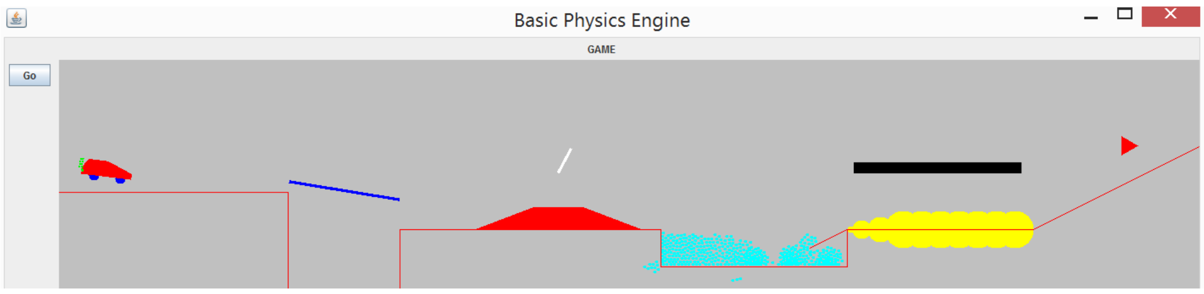
```
world = new World(new Vec2(0, -GRAVITY));
```

As we describe above in report that to create world, we also need iteration values in the game, so we have set them in update method of the file so every time it updates the values as

```
public static final int NUM_EULER_UPDATES_PER_SCREEN_REFRESH=10;

int VELOCITY_ITERATIONS=NUM_EULER_UPDATES_PER_SCREEN_REFRESH;
```

int POSITION_ITERATIONS=***NUM_EULER_UPDATES_PER_SCREEN_REFRESH***;

In world we also need to set its world dimension which also help to make frame for the game to display. We have set Screen height as 250 and width as 220. And the dimension of the screen is set to 6 times of screen width and one time of screen height.



As we know that JBox2D use world dimensions [2] and units, we need to convert them to screen dimensions in subject to use those in on the screen. So, we need to convert the world value to screen value and first convert the world axis to screen axis and then world height and width to screen height and width and vice-versa.

CAR:

In this game main object is car which is drawn using three small objects. One polygon shape as car body and two circle shape as wheels for the car. They are defined in the classes named as BasicParticles and BasicPolygon respectively. All three objects are called in world and they are joint using revolute joint in the game.

Calling of objects in game:

BasicParticle.add (new BasicParticle(0.3f,3.3f,0,3f, r,Color.BLUE, 3, rollingFriction, BodyType.DYNAMIC, 2f, 10f));

BasicParticle.add(new BasicParticle(0.3f,3.3f,0,3f, r,Color.BLUE, 3, rollingFriction, BodyType.DYNAMIC, 2f, 10f));

```
BasicPolygon.add(new  BasicPolygon(WORLD_WIDTH/10,4f,0,0,  r*6,r*4,Color.RED,  30, rollingFriction,4));
```

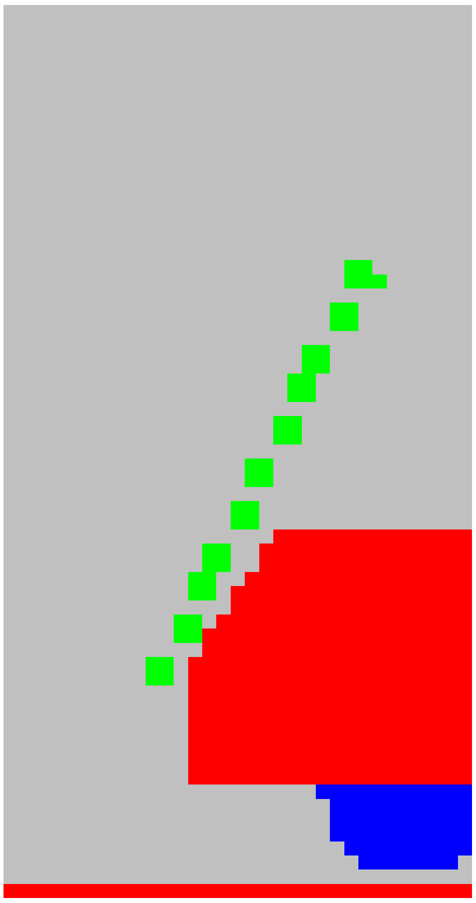
Creating joints between car and wheels

```
 BasicPolygon p1 = BasicPolygon.get(0);

BasicParticle p2 =BasicParticle.get(0);

RevoluteJointDef jointDef=new RevoluteJointDef();

jointDef.bodyA = p1.body;

jointDef.bodyB = p2.body;

jointDef.collideConnected = false;  // this means we don't want these two connected bodies to have collision detection.

jointDef.localAnchorA=new Vec2(-0.7f,-r);

jointDef.localAnchorB=new Vec2(0f,0f);

world.createJoint(jointDef);

BasicParticle p3 = BasicParticle.get(1);

RevoluteJointDef jointDef2=new RevoluteJointDef();

jointDef2.bodyA = p1.body;

jointDef2.bodyB = p3.body;

jointDef2.collideConnected = false;

jointDef2.localAnchorA=new Vec2(2.5f*r,-r);

jointDef2.localAnchorB=new Vec2(0f,0f);

world.createJoint(jointDef2);
```

These both objects use different value of density, friction and restitution.

To move the car, we have applied torque on the wheels which works according to the key press. They are defined in the BasicPhysicsEngineUsingBox2D.java.

Chain: there is one chain, which we have attached to car. That chain is created by using polygon shape in loop a joint to each other using revolute joint and join with car using same joint. Chain is also having opposite gravity then the world.

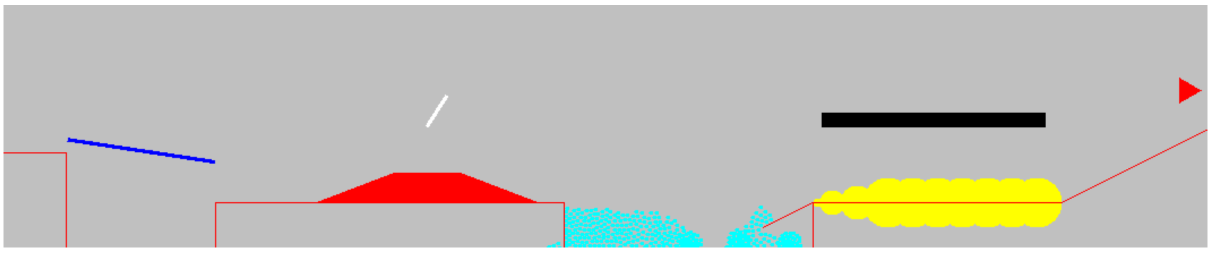


Barriers:

There are some barriers in the game:

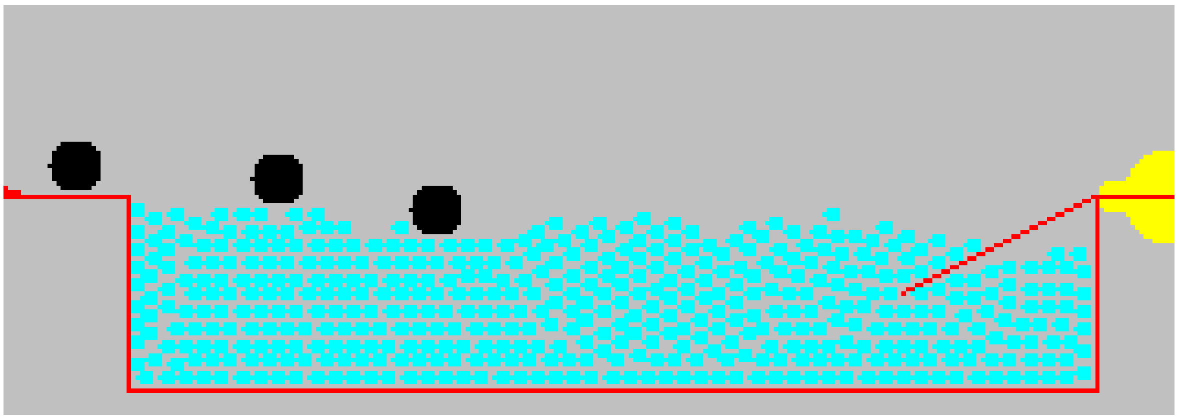
Rolling sticks: they are created as polygon shape with attributes like angular velocity with 0 rolling friction which make these objects rotation continually. Also set the fixed rotation true.

These objects are colliders. If car's body hit them the call back function will be called and according to this game, game will terminate by sending "lose" message in panel.



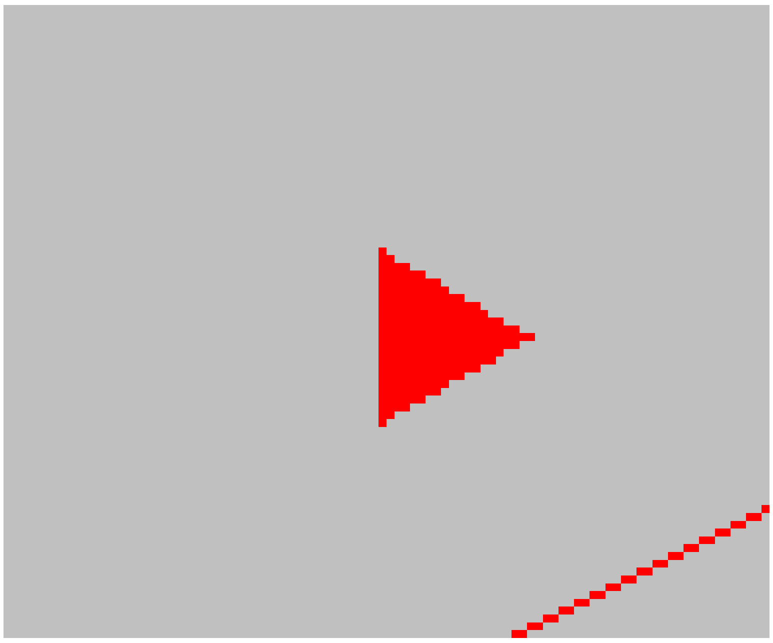
Falling objects: they are small particles falling from top of world at random position. They are also creating using loop. They have only basic particle properties. They work as small barriers for the car to move.

Pool: In the game there is a pool filled with heavy balls. Which need to cross by the car to achieve the target. They are small in size and heavy that car can cross it if go in a flow.



Bumps: there are some yellow bumps created by circle shape. Which are difficult to cross due to the friction they apply on the car and above that there is a black collider. If car hit jumped from bump and hit the collider gave over.

Rewards: There is a reward at the end of this game is a red reward in the triangle shape. It is also defined as polygon shape. Its s also a collider which if collide with car call call back function and announce "WIN" message and exit the game.



Boundaries: As game will start, we have straight edges for car to walk and one hole and if car fall user will lose. Them we have more edges with some mountain, ball pool and bumps and at the end reward. Boundaries, mountain and bumps are STATIC objects.

Contact Listener: In this simulation, contact Listener is used to detect the collision between card body and other objects. Call-back function is define in the file collision.java.

Key Listener: for this game we have key listener only. Which uses both side keys? And on key press we apply torque on the wheels of the car. And this is defined in BasicPhysicsEngineUsingBox2D.java.

Feature missing in the game:

There many more things can be done this game all the features missing can be added in future.

Camera movement: this game does not consist of any camera movement. We can use that feature in future development of game.

Collecting more coins: we have only one reward in this game to demonstrate hoe it can be used. For future work we can use more and different rewards in the game.

Barriers: We have used some barriers in game we can have more.

Endless edges: We can also make this game endless by making some un-parallel edges and continue generation of random edges.

Time or score: we can also use the feature off limited time in the game to make it interesting.

Graphics: As we know JBox2D does not support graphics, we can make own graphics using images in the game, which look more attractive to user.

Joints: Joints can make simple structure more complicated; we can use different joints to make different type of objects.

CONCLUSION

In this report I have mention about the game I developed to demonstrate using JBox2D, which is ported form Box2D in java. Both are almost similar with some small differences due to language difference. It is possible to learn and work with JBox2D with the help of Box2D user manual. As there is no manual is available for JBox2D.

As we have described about the Box2D and Jbox2D in the report that we can make any possible body structure in JBox2D and can adjust different properties of body to make them work as required.

JBox2D have some problems like It do not support any graphical user interface. So, we used different library of java. Still its easy to lean and interesting to make game with JBox2D. It needs some time to learn making proper games with JBox2D but its fun.

References

[1] E. CATTO, “Box2D v2.3.0 User Manual,” 2007-2013. [Online]. Available: : http://box2d.org/manual.pdf.

[2] D. M. Fairbank, *Physics-Based Games, Lecture 5: JBox2D,* 2019.

[3] D. Murphy, “JBox2D: A Java Physics Engine,” 2014. [Online]. Available: : http://www.jbox2d.org/.

[4] “LiquidFun Overview,” [Online]. Available: http://google.github.io/liquidfun/.

[5] D. Kratochv´ıl, “Pinball game in,” MASARYK UNIVERSITY, 2017.

[6] K. Slonneger, 2007. [Online]. Available: http://homepage.divms.uiowa.edu/~slonnegr/wpj/Serialization.pdf.

[7] I. MILLINGTON, Game Physics Engine Development: How to Build a Robust Commercial-Grade Physics Engine for your Game, 2010.