Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа программной инженерии

Отчет по курсовой работе

по дисциплине
«Объектно-ориентированное программирование»

Выполнил
студент гр.23534/1 Стойкоски Н.С.

Руководитель О.Ц. Аристаева
ст. преподаватель

«_26_» __мая_ 2018 г.

Санкт-Петербург
2018

Оглавление

Техническое задание

Создать приложение на базе платформы android, на языке java с подключением соответствующих библиотек.

Реализация

В качестве приложения была выбрана игра, в которой игрок с пушки выстреливает пули по различных конструкциях. Чем больше домики уничтоженние, тем больше полученных баллов.

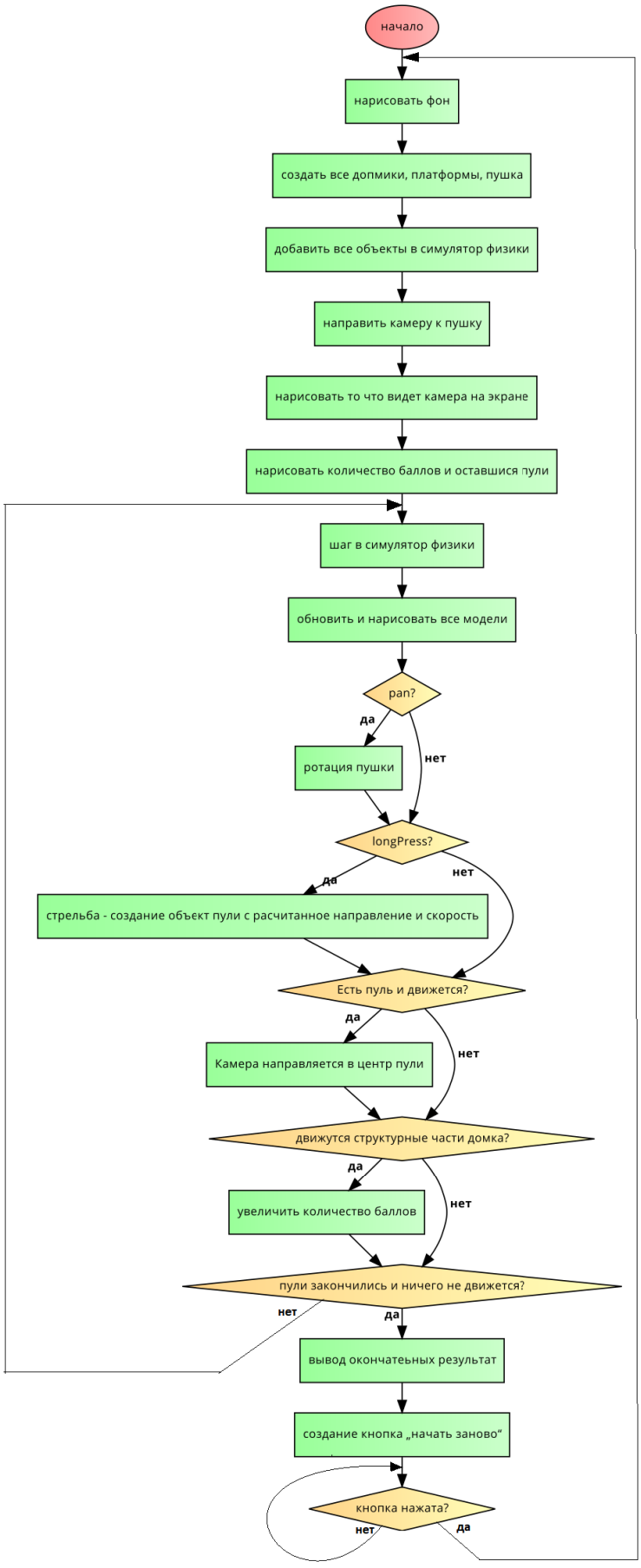
Приложение содержит:

- Пушка, которой можно управлять с помощю сенсорным экраном т.е. можно менять угол стельбы. Она находится в начало уровня, стоит на маленькую платформу в нижном левом угле. Она относительно неподвижная.
- Обьекты – рандомно сгенерированные конструкции (домики). Они состоят из несколько этажов, имеют разные размеры (высота, ширина, количество этажопв). Структурные элементы – деревянные доски разного размера, которые служат в качестве столбцов и платформах.

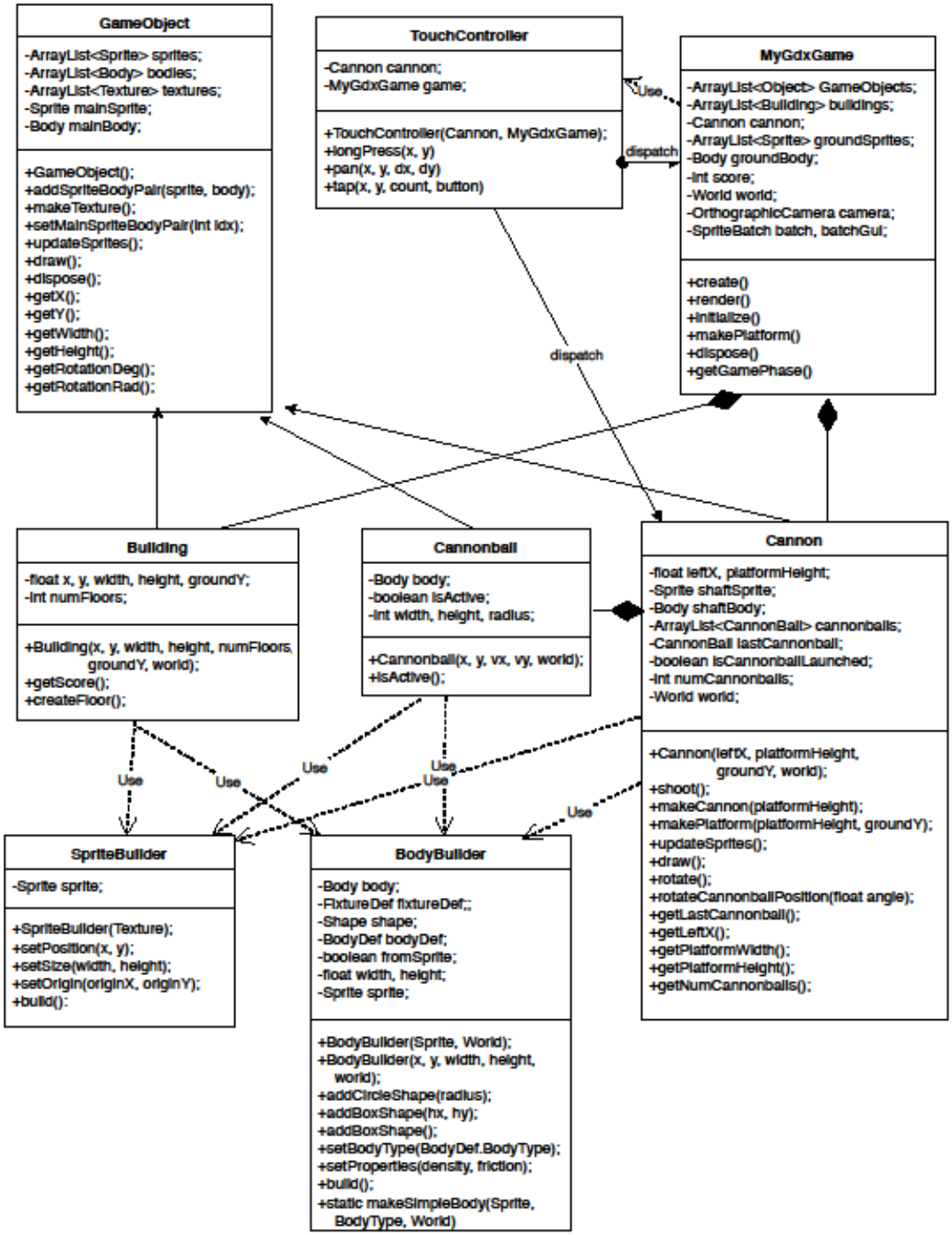
Когда все три пули потрачени, приложение завершается, на эране выводится количество полученных баллов, и кнопка которая позволяет заново запустить игру.

Задание – Создать возможность управления пушкой, ротация и стрельба пули. Применить разные методы управлением графикой с использованием библиотеки libgdx (фреймворк для создания приложения – игры). Применить класс камера – которая первоначально направлена к пушкой и будет следить за пулям когда они движутся. Создать соответствующие классы игровых обьектов – домики, платформы, пушка, пуль. Подключить и применить библиотека Box2d – физический симулятор который адекватно будеть управлять разным обьектам при их столкновение, и движение за счет силы гравитации. Ограничить количество пулей. Создать указатель количество оставщихся пули и количество баллов, которое возрастает при срушении домиков. Когда все пули потрачени, и больше не движутся – вывод окончательного количество баллов на экране и кнопка – возможность заново начать игру.

Алгоритм


начало
нарисовать фон
создать все допмики, платформы, пушка
добавить все объекты в симулятор физики
направить камеру к пушку
нарисовать то что видет камера на экране
нарисовать количество баллов и оставшися пули
шаг в симулятор физики
обновить и нарисовать все модели
pan?
да
нет
ротация пушки
longPress?
да
нет
стрельба - создание объект пули с расчитанное направление и скорость
Есть пуль и движется?
да
нет
Камера направляется в центр пули
движутся структурные части домка?
да
нет
увеличить количество баллов
пули закончились и ничего не движется?
нет
да
вывод окончатеьных результат
создание кнопка „начать заново“
кнопка нажата?
нет
да

Диаграммы классов



Модули

Основные методы:

MyGdxGame:

- initialize() – создает экземпляры всех нужных объектов. Устанавливает процессор обработки ввода. Создает камера, инициализирует симулятора физики. Загружает нужные текстуры. Обнуляет показатели количества оставщихся пули, количество баллов.
- render() – выполняет шаг симулятора физики, устанавливает/обновляет позицию камеры, рисует фон, все обьекты, количество баллов, пули на экране.
- makePlatform() – создает статическая платформа/земля.

GameObject:

Базобый класс всех игровых объектов. Содерьит графический модель (Sprite) и физический модель (body), а так же и все текстуры.

- addSpriteBodyPair(Sprite sprite, Body body) – добавление нового элемента к этому обьекту.
- updateSprites() – обновляет позиция графического моделя в зависимости от физического моделя.
- draw(SpriteBatch batch) – рисуется на экране.

Cannon:

- shoot() – создаёт новый обьект – пуль с предварительно расчитанная позиция, направление и скорость
- rotate(float angle) – вращает пушка по заданным углом.

Cannonball:

- isActive() – дает информация про том – движется ли пуль или нет.

Building:

- getScore() – дает количество баллов в зависимости од скорость движения структурных элементов.

TouchController:

Обрабатывает ввод пользователя.

- longPress(x, y) – вызывает метод shoot() пушки, т.е. стреляет.
- pan(x, y, dx, dy) – вызывает метод rotate() пушки, т.е. вращает её.
- tap(x, y, count, button) – заново инициализирует игру при нажатие кнопки „играть заново“ по оканчания игры.

Инструкция пользователя

Для ротации пушкы – коснитесь экрана пальцем и перетащите по экрана чтобы получуить желаемый угол. Долгая задержка пальцом чтобы выстрелить, при этом число оставщися пули уменьшается (всего 3). При уничтожении домиков увеличивается число баллов. При окончании пули, появится возможность запустить игру заново – это делается при нажатие на соответствующей кнопки.

Выводы

Функционал приложения реализован полностью. Создана возможность управления пушкой и стрельба пули. Применены методы управленяи графикой с использованием библиотеки libgdx. Применен класс камера, которая первоначально направлена к пушку и следит за пулям когда они движутся. Созданы соответсвующие классы игровых объектов – домики, пушка, пуль. Использована библиотека Box2d в качестве симулятора физики (обработка столкновения и движения за счет силы гравитации). Создан графический указатель статуса игры – количество оставщихся пули, количество баллов. Добавлена возможность заново начать игру.

Приложение

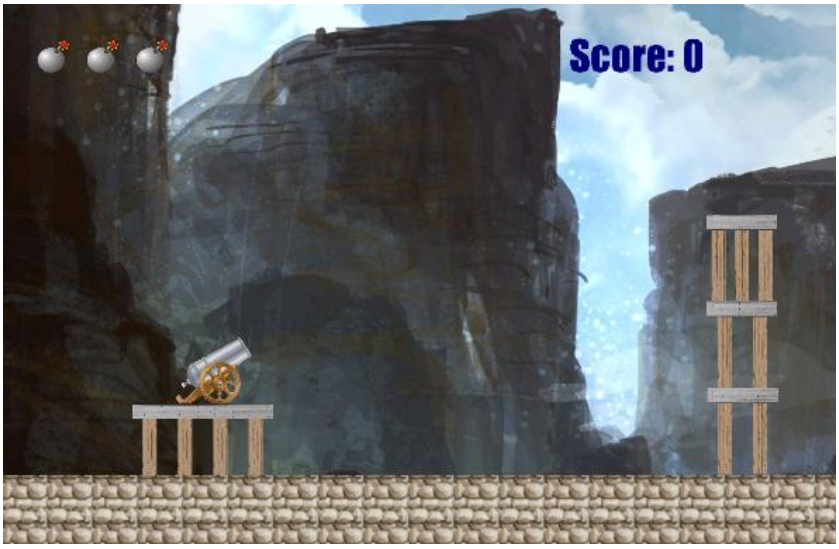


Рис.1 Начало игры

Рис.2 Разрушение домика

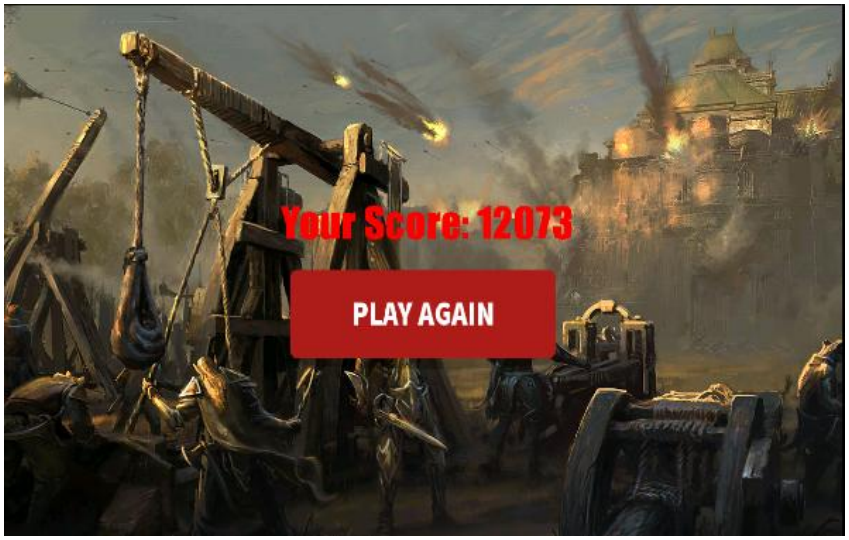


Рис. 3 Конец игры

Исходный код

MyGdxGame

```
package com.mygdx.game;

import com.badlogic.gdx.ApplicationAdapter;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.graphics.g2d.Sprite;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.input.GestureDetector;
import com.badlogic.gdx.math.Matrix4;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.physics.box2d.Body;
import com.badlogic.gdx.physics.box2d.BodyDef;
import com.badlogic.gdx.physics.box2d.Box2DDebugRenderer;
import com.badlogic.gdx.physics.box2d.World;

import java.util.ArrayList;


public class MyGdxGame extends ApplicationAdapter {

    public static final float PIXELS_TO_METERS = 100f;
    public enum GamePhase{
        CANNON_WAITING,
        PROJECTILE_FLYING,
        GAME_OVER
    }
    private GamePhase gamePhase;
    private SpriteBatch batch;
    private World world;
    private OrthographicCamera camera;

    //gui
    private SpriteBatch batchGui;
    private BitmapFont guiFont;
    private int totalScore = 0;
    private ArrayList<Sprite> remainingCannonballs;

    //Game Objects
    private ArrayList<GameObject> gameObjects;
    private ArrayList<Building> buildings;
    private Cannon cannon;
    private Sprite backgroundSprite;

    //Ground
    //private Terrain terrain;
    private ArrayList<Sprite> groundSprites;
    private Texture groundTexture, backgroundTexture;
    private Body groundBody;

    //Misc
    public static boolean drawSprite = true;
    private Box2DDebugRenderer debugRenderer;
    private Matrix4 debugMatrix;


    @Override
    public void create () {
        initialize();
    }
```

```
   @Override
   public void render () {

      world.step(1f/60f, 7, 7);

      for(GameObject gameObject: gameObjects){
         gameObject.updateSprites();
      }

      //camera follows cannon/cannonball
      if(cannon.isCannonBallLaunched() && cannon.getLastCannonball().isActive()){
         Cannonball lastCannonball = cannon.getLastCannonball();
         camera.position.set(Math.max(lastCannonball.getX(),
Gdx.graphics.getWidth() / 2),
               Math.max(Gdx.graphics.getHeight() / 2, lastCannonball.getY()), 0);
         Gdx.app.log("camera", "camera on ball");
      } else{
           camera.position.set(Math.max(cannon.getLeftX() +
cannon.getPlatformWidth() / 2, Gdx.graphics.getWidth() / 2),
               Math.max(Gdx.graphics.getHeight() / 2, cannon.getPlatformHeight() +
100 ), 0);
         Gdx.app.log("camera", "camera on cannon");
      }
      camera.update();


      //get Score
      for(Building b: buildings){
         totalScore += b.getScore();
      }

      Gdx.gl.glClearColor(1, 0, 0, 1);
      Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
      batch.setProjectionMatrix(camera.combined);

      batch.begin();

      //Draw Background
      backgroundSprite.draw(batch);

      //Draw ground
      for(Sprite groundSprite: groundSprites){
         groundSprite.draw(batch);
      }

      //Draw gameObjects
      for(GameObject gameObject: gameObjects){
         gameObject.draw(batch);
      }
      batch.end();

      //Draw gui
      batchGui.begin();

      guiFont.draw(batchGui, "Score: " + totalScore, Gdx.graphics.getWidth() - 380,
Gdx.graphics.getHeight() - 50);

      for(int i = 0; i < cannon.getNumCannonballs(); i++){
         remainingCannonballs.get(i).draw(batchGui);
      }

      if(cannon.getNumCannonballs() == 0 &&
!cannon.getLastCannonball().isActive()){
         gamePhase = GamePhase.GAME_OVER;
          batchGui.draw(new Texture("medieval-battle-wallpaper.jpg"), 0, 0);
          guiFont.setColor(Color.RED);
         guiFont.draw(batchGui, "Your Score: " + totalScore,
Gdx.graphics.getWidth() / 2 - 200, Gdx.graphics.getHeight()/2 + 100);
         batchGui.draw(new Texture("button-playagain.png"),
Gdx.graphics.getWidth()/2 - 200,
```

```
            Gdx.graphics.getHeight()/2 - 130, 400, 140);
    }

    batchGui.end();
}


public void initialize(){
    batch = new SpriteBatch();
    camera = new OrthographicCamera(Gdx.graphics.getWidth(),
          Gdx.graphics.getHeight());
    world = new World(new Vector2(0, -18.81f), true);
    debugRenderer = new Box2DDebugRenderer();
    gameObjects = new ArrayList<GameObject>();

    batchGui = new SpriteBatch();
    guiFont = new BitmapFont(Gdx.files.internal("impactFont.fnt"), false);
    guiFont.setColor(Color.NAVY);
    guiFont.getData().setScale(0.8f);
    remainingCannonballs = new ArrayList<Sprite>();

    Texture cannonBallTexture = new Texture("cannonball2.png");
    for(int i = 0; i < 3; i++){
        Sprite cannonball = new Sprite(cannonBallTexture);
        cannonball.setPosition(50+70*i, Gdx.graphics.getHeight()-100);
        cannonball.setSize(50,50);
        remainingCannonballs.add(cannonball);
    }

    backgroundTexture = new Texture("background-cliffs.jpg");
    backgroundSprite = new Sprite(backgroundTexture);
    backgroundSprite.setPosition(0, -600 );


    //Terrain
    //terrain = new Terrain(36, backgroundSprite.getWidth(), world);
    makePlatform();

    //Buildings
    buildings = new ArrayList<Building>();
    for(int i = 0; i < 5; i++){
        Building b = new Building(1000 + i*450, 100,
              100+(i%5)*50, 100 + i*50,
              3+i%3, 100, world);
        buildings.add(b);
        gameObjects.add(b);
    }

    //Cannon
    cannon = new Cannon(200, 100, 100, world);
    gameObjects.add(cannon);
    gamePhase = GamePhase.CANNON_WAITING;

    totalScore = 0;

      Gdx.input.setInputProcessor(new GestureDetector
              (new TouchController(cannon,this)));

}

private void makePlatform(){
    groundTexture = new Texture("brown-platform.png");
    groundSprites = new ArrayList<Sprite>();


    groundBody = new BodyBuilder(0,0,backgroundSprite.getWidth(), 100, world)
          .setBodyType(BodyDef.BodyType.StaticBody)
          .addBoxShape()
          .setProperties(0.1f,10f)
          .build();
```

```
        groundTexture = new Texture("rock-block.jpg");
        float totalWidth = 0;
        while(totalWidth < backgroundSprite.getWidth()){
            Sprite groundBlock = new Sprite(groundTexture);
            groundBlock.setPosition(totalWidth, 36);
           groundSprites.add(groundBlock);

           Sprite groundBlock2 = new Sprite(groundTexture);
           groundBlock2.setPosition(totalWidth, -28);
           groundSprites.add(groundBlock2);

           totalWidth += groundBlock.getWidth();
        }
    }

    @Override
    public void dispose () {
        batch.dispose();
        world.dispose();
        backgroundTexture.dispose();
        groundTexture.dispose();

        for(GameObject gameObject: gameObjects){
            gameObject.dispose();
        }
    }

    public GamePhase getGamePhase(){
        return gamePhase;
    }
}
```

GameObject

```
package com.mygdx.game;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Sprite;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.physics.box2d.Body;

import java.util.ArrayList;

public class GameObject {

    public static final float PIXELS_TO_METERS = 100f;

    protected ArrayList<Sprite> sprites;
    protected ArrayList<Body> bodies;
    protected ArrayList<Texture> textures;
    protected Sprite mainSprite;
    protected Body mainBody;

    public GameObject(){
        sprites = new ArrayList<Sprite>();
        bodies = new ArrayList<Body>();
        textures = new ArrayList<Texture>();
    }
    public int addSpriteBodyPair(Sprite sprite, Body body){
        sprites.add(sprite);
        bodies.add(body);
        return sprites.size() - 1;
    }
    public Texture makeTexture(String internalPath){
        Texture texture = new Texture(internalPath);
        textures.add(texture);
        return texture;
    }
    public void setMainSpriteBodyPair(int idx){
```

```
        mainSprite = sprites.get(idx);
        mainBody = bodies.get(idx);
    }
    public void updateSprites(){

        for(int i = 0; i < sprites.size(); i++){

            Sprite sprite = sprites.get(i);
            Body body = bodies.get(i);

            sprite.setPosition((body.getPosition().x * PIXELS_TO_METERS) -
sprite.getWidth() / 2,
                    (body.getPosition().y * PIXELS_TO_METERS)-
sprite.getHeight()/2);
            sprite.setRotation((float)Math.toDegrees(body.getAngle()));
        }
    }

    public void draw(SpriteBatch batch){
        for(Sprite sprite: sprites){
            batch.draw(sprite, sprite.getX(), sprite.getY(),
                    sprite.getOriginX(), sprite.getOriginY(),
                    sprite.getWidth(), sprite.getHeight(),
                    sprite.getScaleX(), sprite.getScaleY(),
                    sprite.getRotation());
        }
    }

    public void dispose(){
        for(Texture texture: textures){
            texture.dispose();
        }
    }

    public float getX(){return mainSprite.getX();}
    public float getY(){return mainSprite.getY();}
    public float getWidth(){return mainSprite.getWidth();}
    public float getHeight(){return mainSprite.getHeight();}
    public float getRotationDeg(){return mainSprite.getRotation();}
    public float getRotationRad(){return mainBody.getAngle();}
}
```

TouchController

```
package com.mygdx.game;

import com.badlogic.gdx.Application;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.input.GestureDetector;
import com.badlogic.gdx.math.Vector2;

public class TouchController implements GestureDetector.GestureListener{

    private Cannon cannon;
    private MyGdxGame game;

    private final Texture blockTexture = new Texture("block.png");

    public TouchController(Cannon cannon, MyGdxGame game){
        super();
        Gdx.app.setLogLevel(Application.LOG_DEBUG);
        this.cannon = cannon;
        this.game = game;
    }

    @Override
    public boolean longPress(float x, float y) {
        MyGdxGame.drawSprite = !MyGdxGame.drawSprite;
        cannon.shoot();
```

```
        return true;
    }
    @Override
    public boolean pan(float x, float y, float deltaX, float deltaY) {

        //cannon.rotate( 4*10^{-6}x^2-0.0038x+1 * distance(touch, cannon) * deltaY
* 0.24f);
        if(game.getGamePhase() == MyGdxGame.GamePhase.CANNON_WAITING){
            Gdx.app.log("Touch-Pan", "x,y,dx,dy: " + x +  ", " + y + ", " + deltaX
+ ", " +deltaY);
            cannon.rotate(deltaY * 0.24f );
            return true;
        }else{
            return false;
        }
    }
    @Override
    public boolean tap(float x, float y, int count, int button) {
        if(game.getGamePhase() == MyGdxGame.GamePhase.GAME_OVER){
            game.dispose();
            game.initialize();
        }
        return true;
    }

    @Override
    public boolean touchDown(float x, float y, int pointer, int button) {
        return false;
    }
    @Override
    public boolean fling(float velocityX, float velocityY, int button) { return
false; }

    @Override
    public boolean panStop(float x, float y, int pointer, int button) {
        return false;
    }
    @Override
    public boolean zoom(float initialDistance, float distance) {
        return false;
    }
    @Override
    public boolean pinch(Vector2 initialPointer1, Vector2 initialPointer2, Vector2
pointer1, Vector2 pointer2) {
        return false;
    }
    @Override
    public void pinchStop() {

    }
}
```

Building

```
package com.mygdx.game;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Sprite;
import com.badlogic.gdx.physics.box2d.Body;
import com.badlogic.gdx.physics.box2d.BodyDef;
import com.badlogic.gdx.physics.box2d.World;
import java.util.Random;

public class Building extends GameObject {

    private float x, y, width, height, groundY;
    private int numFloors;

    public Building(float x, float y, float width, float height, int numFloors,
```

```
float groundY, World world){
        super();
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.numFloors = numFloors;
        this.groundY = groundY;

        float nextFloorY = groundY;
        for(int i=0; i< numFloors; i++){
            nextFloorY = createFloor(nextFloorY, world);
        }
    }

    public int getScore(){
        int score = 0;
        for(Body b: bodies){
            score += b.getLinearVelocity().dot(b.getLinearVelocity());
        }
        //Gdx.app.log("SCORE", "score is: " + score);

        if(score < 100)
            score = 0;

        return score;
    }

    private float createFloor(float groundHeight, World world) {

        Random rand = new Random();
        int numColumns = Math.max(2, rand.nextInt() % 4);

        float curSectionLeft = x;
        float supportingWidth = width / numColumns;
        float columnWidth = 20, columnHeight = 100;

        Texture woodTexture = makeTexture("wood1.jpg");

        //Add wood columns
        for(int i=0; i<numColumns; i++){

            float curColx = curSectionLeft + (supportingWidth / 2);
            Sprite sprite = new SpriteBuilder(woodTexture)
                    .setPosition(curColx - columnWidth/2, groundHeight)
                    .setSize(columnWidth, columnHeight)
                    .setOrigin(columnWidth/2, columnHeight/2)
                    .build();
            Body body = BodyBuilder.makeSimpleBody(sprite,
BodyDef.BodyType.DynamicBody, world);
            addSpriteBodyPair(sprite, body);
            curSectionLeft += supportingWidth;

            Gdx.app.log("Building", "add column at: " + (curColx - columnWidth/2) +
", " + groundHeight);
        }

        //Add wood ceiling
        int ceilingHeight = 20;
        Texture stoneTexture = makeTexture("stone.jpg");

        Sprite plankSprite = new SpriteBuilder(stoneTexture)
                .setPosition(x, groundHeight + columnHeight + 5)
                .setSize(width, ceilingHeight)
                .setOrigin(width/2, ceilingHeight/2)
                .build();
        Body plankBody = new BodyBuilder(plankSprite, world)
                .addBoxShape()
                .setBodyType(BodyDef.BodyType.DynamicBody)
                .setProperties(3f, 15f)
```

```
                .build();

        addSpriteBodyPair(plankSprite, plankBody);

        Gdx.app.log("Building", "add plank at: " + x + ", " + (groundHeight +
columnHeight));

        //Return highest point
        return (groundHeight + columnHeight + ceilingHeight + 10);
    }
}
```

Cannon

```
package com.mygdx.game;

import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Sprite;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.physics.box2d.Body;
import com.badlogic.gdx.physics.box2d.BodyDef;
import com.badlogic.gdx.physics.box2d.World;
import java.util.ArrayList;
import box2dLight.PointLight;
import box2dLight.RayHandler;

public class Cannon extends GameObject {

    //Constants
    static final int numColumns = 4;
    static final float platformWidth = 200;

    //Position
    private float leftX, platformHeight;
    private Sprite shaftSprite;
    private Body shaftBody;

    //Cannonballs
    private ArrayList<Cannonball> cannonballs;
    private float nextCannonballX, nextCannonballY;
    private float nextCannonballVX = 13, nextCannonballVY = 0;
    private Cannonball lastCannonball;
    private boolean isCannonBallLaunched = false;
    private int numCannonballs = 3;

    //World reference
    private World world;

    public Cannon(int leftX, int platformHeight, int groundY, World world){
        super();
        this.leftX= leftX;
        this.platformHeight = platformHeight;
        this.world = world;

        cannonballs = new ArrayList<Cannonball>();
        //cannonballLight = new PointLight(rayHandler, 1000, Color.GOLD, 300, 0,0);

        makePlatform(platformHeight, groundY);
        makeCannon(groundY + platformHeight);

        //todo: make method - setupInitialCannonballPosition
        nextCannonballX = shaftSprite.getX() + shaftSprite.getWidth() / 2 + 75;
        nextCannonballY = shaftSprite.getY() + shaftSprite.getHeight() / 2 - 25;
        rotateCannonballPosition(-45);
    }

    public void shoot(){

        if(numCannonballs == 0)
```

```
            return;

        lastCannonball = new Cannonball((int) nextCannonballX,
(int)nextCannonballY,
                nextCannonballVX * 1.4f, nextCannonballVY * 1.4f, world);
        cannonballs.add(lastCannonball);
        isCannonBallLaunched = true;
        numCannonballs--;
    }
    private void makeCannon(int platformHeight){

        shaftSprite = new SpriteBuilder(makeTexture("cannon-shaft.png"))
                .setPosition(leftX + platformWidth / 2 - 55, platformHeight)
                .setSize(111, 96)
                .setOrigin(111/2, 96/2)
                .build();
        shaftBody = BodyBuilder.makeSimpleBody(shaftSprite,
BodyDef.BodyType.StaticBody, world);
        addSpriteBodyPair(shaftSprite, shaftBody);

        Sprite baseSprite = new SpriteBuilder(makeTexture("cannon-base.png"))
                .setPosition(leftX + platformWidth / 2 - 55, platformHeight)
                .setSize(120, 108)
                .build();
        Body baseBody = BodyBuilder.makeSimpleBody(baseSprite,
BodyDef.BodyType.StaticBody, world);
        addSpriteBodyPair(baseSprite, baseBody);

    }

    //todo: dali treba ova?
    //bodyDef.angle = (float)Math.toRadians(sprite.getRotation());

    private void makePlatform(int platformHeight, int groundY){

        //todo: cannon platform should be a Building
        Texture woodTexture = makeTexture("wood1.jpg");

        float plankHeight = 20;
        float columnWidth = 20, columnHeight = platformHeight - plankHeight;
        float currentX = leftX;
        float supportingWidth = platformWidth / numColumns;

        for(int i = 0; i < numColumns; i++){
            Sprite columnSprite = new SpriteBuilder(woodTexture)
                    .setPosition(currentX, groundY)
                    .setSize(columnWidth, columnHeight)
                    .build();
            Body columnBody = BodyBuilder.makeSimpleBody(columnSprite,
BodyDef.BodyType.StaticBody, world);
            addSpriteBodyPair(columnSprite, columnBody);
            currentX += supportingWidth;
        }

        Texture stoneTexture = makeTexture("stone.jpg");
        Sprite plankSprite = new SpriteBuilder(stoneTexture)
                .setPosition(leftX - 15, groundY + columnHeight)
                .setSize(platformWidth, plankHeight)
                .build();
        Body plankBody = BodyBuilder.makeSimpleBody(plankSprite,
BodyDef.BodyType.StaticBody, world);
        addSpriteBodyPair(plankSprite, plankBody);
    }

    @Override
    public void updateSprites() {
        super.updateSprites();
        for(Cannonball cannonball: cannonballs){
            cannonball.updateSprites();
        }
```

```
        if(isCannonBallLaunched()){
            //cannonballLight.setPosition(lastCannonball.getX(),
lastCannonball.getY());
        }
    }

    @Override
    public void draw(SpriteBatch batch) {
        super.draw(batch);
        for(Cannonball cannonball: cannonballs){
            cannonball.draw(batch);
        }
    }

    public void rotate(float angle){
        //todo: dali mora na sekoe da se mnozi so pixels to meters?
        shaftBody.setTransform(shaftBody.getPosition(), shaftBody.getAngle() -
(float) Math.toRadians(angle));

        rotateCannonballPosition(angle);

    }
    private void rotateCannonballPosition(float angle){
        float cosA = (float)Math.cos(Math.toRadians(-angle));
        float sinA = (float)Math.sin(Math.toRadians(-angle));

        float tx = nextCannonballX - shaftBody.getPosition().x * PIXELS_TO_METERS;
        float ty = nextCannonballY - shaftBody.getPosition().y * PIXELS_TO_METERS;

        nextCannonballX = tx*cosA - ty*sinA + shaftBody.getPosition().x *
PIXELS_TO_METERS;
        nextCannonballY = tx*sinA + ty*cosA + shaftBody.getPosition().y *
PIXELS_TO_METERS;

        nextCannonballVX = 0.185f * (nextCannonballX - shaftBody.getPosition().x *
PIXELS_TO_METERS);
        nextCannonballVY = 0.185f * (nextCannonballY - shaftBody.getPosition().y *
PIXELS_TO_METERS);
    }

    public Cannonball getLastCannonball(){
        return lastCannonball;
    }
    public boolean isCannonBallLaunched(){
        return isCannonBallLaunched;
    }
    public float getLeftX(){
        return leftX;
    }
    public float getPlatformWidth(){
        return platformWidth;
    }
    public float getPlatformHeight(){
        return platformHeight;
    }
    public int getNumCannonballs(){return numCannonballs;}
}
```

Cannonball

```
package com.mygdx.game;

import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.g2d.Sprite;
import com.badlogic.gdx.physics.box2d.Body;
import com.badlogic.gdx.physics.box2d.World;

import box2dLight.PointLight;
import box2dLight.RayHandler;
```

```
public class Cannonball extends GameObject {

    static final int width = 20;
    static final int height = 20;
    static final int radius = 10;

    private Body body;
    private boolean isActive = true;

    public Cannonball(int x, int y, float vx, float vy, World world){
        super();

        Sprite sprite = new SpriteBuilder(makeTexture("cannon-ball.png"))
                .setPosition(x, y)
                .setSize(width, height)
                .setOrigin(width/2, height/2)
                .build();

        body = new BodyBuilder(sprite, world)
                .addCircleShape(radius)
                .setProperties(7.8f, 4f)
                .build();

        body.setLinearVelocity(vx, vy);

        setMainSpriteBodyPair(addSpriteBodyPair(sprite, body));
    }

    public boolean isActive(){
        if(isActive && (Math.pow(body.getLinearVelocity().x, 2) +
Math.pow(body.getLinearVelocity().y, 2) < 0.025)){
            isActive = false;
        }
        return isActive;
    }
}
```

SpriteBuilder

```
package com.mygdx.game;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Sprite;

public class SpriteBuilder {

    private Sprite sprite;

    public SpriteBuilder(Texture texture){
        sprite = new Sprite(texture);
    }
    public SpriteBuilder setPosition(float x, float y){
        sprite.setPosition(x, y);
        return this;
    }
    public SpriteBuilder setSize(float width, float height){
        sprite.setSize(width, height);
        return this;
    }
    public SpriteBuilder setOrigin(float originX, float originY){
        sprite.setOrigin(originX, originY);
        return this;
    }
    public Sprite build(){
        return sprite;
    }

}
```

BodyBuilder

```
package com.mygdx.game;

import com.badlogic.gdx.graphics.g2d.Sprite;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.physics.box2d.Body;
import com.badlogic.gdx.physics.box2d.BodyDef;
import com.badlogic.gdx.physics.box2d.CircleShape;
import com.badlogic.gdx.physics.box2d.FixtureDef;
import com.badlogic.gdx.physics.box2d.PolygonShape;
import com.badlogic.gdx.physics.box2d.Shape;
import com.badlogic.gdx.physics.box2d.World;

public class BodyBuilder {

    private Body body;
    private FixtureDef fixtureDef;
    private Shape shape;
    private BodyDef bodyDef;
    private boolean fromSprite;
    private float width, height;

    //references
    private World world;
    private Sprite sprite;

    public BodyBuilder(Sprite sprite, World world){
        this.world = world;
        this.sprite = sprite;
        bodyDef = new BodyDef();
        bodyDef.type = BodyDef.BodyType.DynamicBody;
        bodyDef.position.set((sprite.getX() + sprite.getWidth() / 2) /
MyGdxGame.PIXELS_TO_METERS,
                (sprite.getY() + sprite.getHeight()/2 ) /
MyGdxGame.PIXELS_TO_METERS);
        fromSprite = true;
    }
    public BodyBuilder(float x, float y, float width, float height, World world){
        this.world = world;
        this.width = width;
        this.height = height;
        bodyDef = new BodyDef();
        bodyDef.type = BodyDef.BodyType.DynamicBody;
        bodyDef.position.set((x+width/2) / MyGdxGame.PIXELS_TO_METERS,
                (y+height/2) / MyGdxGame.PIXELS_TO_METERS);
        fromSprite = false;
    }
    public BodyBuilder addCircleShape(int radius){
        CircleShape shape = new CircleShape();
        shape.setRadius(radius / MyGdxGame.PIXELS_TO_METERS);
        shape.setPosition(new Vector2(0,0));
        this.shape = shape;
        return this;
    }
    public BodyBuilder addBoxShape(float hx, float hy){
        PolygonShape shape = new PolygonShape();
        shape.setAsBox(hx / MyGdxGame.PIXELS_TO_METERS,hy /
MyGdxGame.PIXELS_TO_METERS);
        this.shape = shape;
        return this;
    }
    public BodyBuilder addBoxShape(){
        if(fromSprite){
            return addBoxShape(sprite.getWidth() / 2, sprite.getHeight() / 2);
        }else{
            return addBoxShape(width/2, height/2);
        }

    }
    public BodyBuilder setBodyType(BodyDef.BodyType type){
```

```
        bodyDef.type = type;
        return this;
    }
    public BodyBuilder setProperties(float density, float friction){
        fixtureDef = new FixtureDef();
        fixtureDef.shape = shape;
        fixtureDef.density = density;
        fixtureDef.friction = friction;
        fixtureDef.restitution = 0.1f;
        return this;
    }
    public Body build(){
        body = world.createBody(bodyDef);
        body.createFixture(fixtureDef);
        shape.dispose();
        return body;
    }

    public static Body makeSimpleBody(Sprite sprite, BodyDef.BodyType bodyType,
World world){
        return new BodyBuilder(sprite, world)
                .addBoxShape()
                .setBodyType(bodyType)
                .setProperties(0.53f, 15f)
                .build();
    }

}
```