Санкт-Петербургский Политехнический Университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Программирование

Отчет по курсовой работе
2D-игра жанра Платформер

Работу выполнил:
Осипов А. Ю.
Группа: 23501/4
Преподаватель:
Вылегжанина К.Д.

Санкт-Петербург
2017

Содержание

1 2D-игра жанра Платформер

1.1 Концепция приложения

Платформер (англ. platformer) — жанр компьютерных игр, в которых основной чертой игрового процесса является прыгание по платформам, лазанье по лестницам, собирание предметов, обычно необходимых для завершения уровня. Некоторые предметы, называемые пауэр-апами (англ. power-up), наделяют управляемого игроком персонажа особой силой, которая обычно иссякает со временем (к примеру: силовое поле, ускорение, увеличение высоты прыжков). Коллекционные предметы, оружие и «пауэер-ап» собираются обычно простым прикосновением персонажа и для применения не требуют специальных действий со стороны игрока. Реже предметы собираются в «инвентарь» героя и применяются специальной командой (такое поведение более характерно для аркадных головоломок). Сходный жанр компьютерных игр SideScroller. Противники (называемые «врагами»), всегда многочисленные и разнородные, обладают примитивным искусственным интеллектом, стремясь максимально приблизиться к игроку, либо не обладают им вовсе, перемещаясь по круговой дистанции или совершая повторяющиеся действия. Соприкосновение с противником обычно отнимает жизненные силы у героя или вовсе убивает его. Иногда противник может быть нейтрализован либо прыжком ему на голову, либо из оружия, если им обладает герой. Смерть живых существ обычно изображается упрощённо или символически (существо исчезает или проваливается вниз за пределы экрана). Уровни, как правило, изобилуют секретами (скрытые проходы в стенах, высокие или труднодоступные места), нахождение которых существенно облегчает прохождение и подогревает интерес игрока. Игры подобного жанра характеризуются нереалистичностью, рисованной мультяшной графикой. Героями таких игр обычно бывают мифические существа (к примеру: драконы, гоблины) или антропоморфные животные. Платформеры появились в начале 1980-х и стали трёхмерными ближе к концу 1990-х. Через некоторое время после образования жанра у него появилось данное название, отражающее тот факт, что в платформерах геймплей сфокусирован на прыжках по платформам в противовес стрельбе. Правда, во многих платформерах присутствует стрелковое оружие, в таких, например, как Blackthorne или Castlevania.

1.2 Задание

Разработать 2D игровое приложение под ОС Windows и Android. Приложение представляет собой 2D-экшн игру. Пользователю предлагается управлять двухмерным персонажем в аналогичном мире, разбитом на уровни. Для прохождения каждого уровня необходимо выполнить задание уровня, например: доставить персонажа в финальную точку уровня, уничтожить всех противников, выживать в течение некоторого времени и.т.д. Основное отличие жанра "Платформер"состоит в том, что игровой мир плоский (вид сбоку) и разбит на несколько уровней по вертикали, на каждом из которых может находиться наш песонаж.

1.3 Минимально работоспособный продукт

Небольшая игра, которая позволяет пользователю управлять персонажем в двухмерном мире с предусмотренной физикой и возможностью окончания игры.

1.4 Вывод

Пояснён выбор темы курсового проекта. Описана концепция игрового положения "Платформер". Определено задание.

2 Проектирование приложения

2.1 Архитектура приложения

Приложение было разбито на 2 модуля:

- core - основной набор классов для реализации игровой логики
- desktop - лаунчер для запуска приложения на ОС Windows

Архитектура ядра выглядит следующим образом:

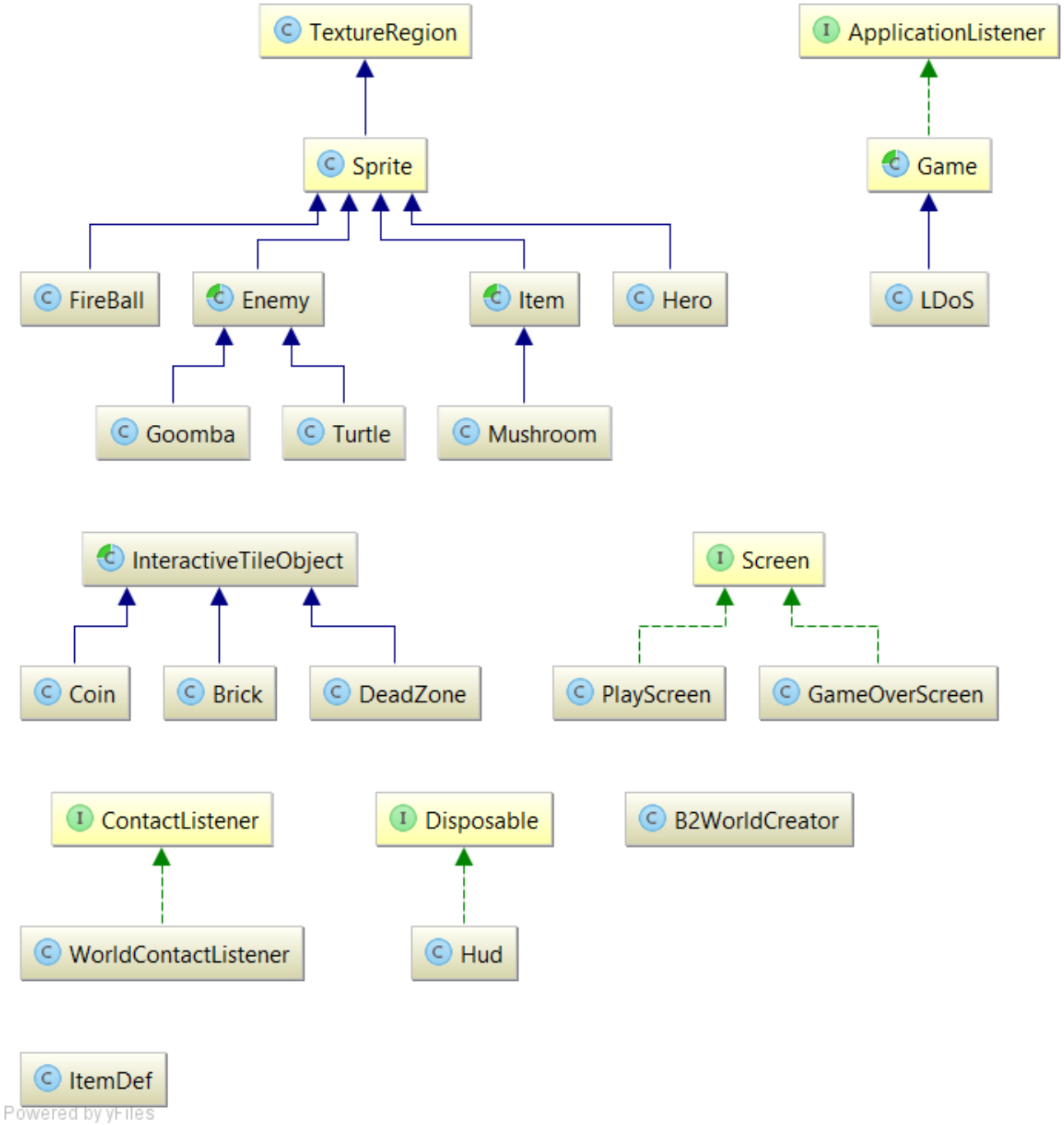

Рис. 1: Диаграма классов ядра

2.2 Карты

В качестве карт для приложения был выбран тип Tiled-Maps как самый простой в создании, а также обладающий широкими возможностями для редактирования.
В интернете есть большое количество Tiled-Map редакторов. Для проекта был выбран бесплатный редактор Tiled. Карты формата .tmx загружаются в проект средствами библиотеки GDX.

3 Реализация приложения

3.1 Используемые версии

- IntelliJ IDEA 2016.3.1
  Build IU-163.9166.29
  For educational use only.
  JRE: 1.8.0 102-b14 amd64
  JVM: Java HotSpot(TM) 64-Bit Server VM by Oracle Corporation

- Java language level: 6
- Операционная система: Windows 10 x64
- LibGDX 1.9.5
- gdx-texturepacker-3.2.0
- Tined Map Editor v3.1
- Система автоматической сборки: Gradle 2.14

3.2 LibGDX и его использование при разработке игрового приложения

LibGDX[1] — фреймворк для создания игр и приложений, написанный на Java с использованием C и C++ (для более быстрой работы). Он позволяет писать кроссплатформенные игры и приложения используя один код. [2]

Решение, о его использование было принято по трём причинам:

1 Кроссплатформенность, именно благодаря этому параметру была достигнута цель создать приложение сразу под две операционные системы

2 Удобство создания графических объектов и анимации

3 Изучение нового

3.3 Проектирование приложения

Классы, обеспечивающие работу приложения находятся в пакете com.mygdx.game. Для многих классов в качестве родителя был выбран класс Sprite из библиотеки LibGDX для удоства последующей отрисовки объектов этих классов. Их список приведен ниже:

- Hero - класс главного героя нашей игры
- Enemy - класс противника, от которого наследуются классы конкретных врагов (Turtle и Goomba)
- Item - класс для представления игровых предметов. От него наследуются следующие классы: Mushroom (на данный момент новых игровых предметов не добавлено)
- InteractiveTileObject - класс для представления интерактивных игровых объектов, т. е. объектов, с которыми может взаимодействовать игровой персонаж

Также было создано 2 класса, обеспечивающие генерацию мира и обработку событий, которые в нем происходят:

- B2WorldCreator - класс, генерирующий игровой мир: на основе загруженной .tmx карты создаются игровые объекты, спавнятся противники.
- WorldContactListener - класс, использующий средства библиотеки LibGDX, обрабатывающий столкновения на основе Collision-моделей игровых объектов.

[1] https://libgdx.badlogicgames.com
[2] https://ru.wikipedia.org/wiki/LibGDX

3.4 Процесс создания приложения

Процесс создания описан пошагово, затронуты ключевые пункты разработки:

1 Создано игровое окно

2 Создан HUD (интерфейс пользователя)

3 На основе AssetManager из LibGDX реализована подгрузка материалов

4 Создана карта с помощью Tiled Map Editor

5 Созданы спрайты игровых объектов

6 Создан персонаж и описана его основная функциональность

7 Добавлена анимация бега и прыжков

8 Реализован игровой мир на основе созданной карты, добавлено взаимодействие с интерактивными объектами

9 Добавлены звуки

10 Добавлены противники и взаимодействие с ними

11 Добавлена генерация предметов и взаимодействие с ними

12 Добавлены эффекты Power-up

13 Добавлен Game Over экран

14 Реализована смерть персонажа от противников и по истечению времени

3.5 Скриншоты процесса разработки

Скриншоты процесса разработки приведены ниже:

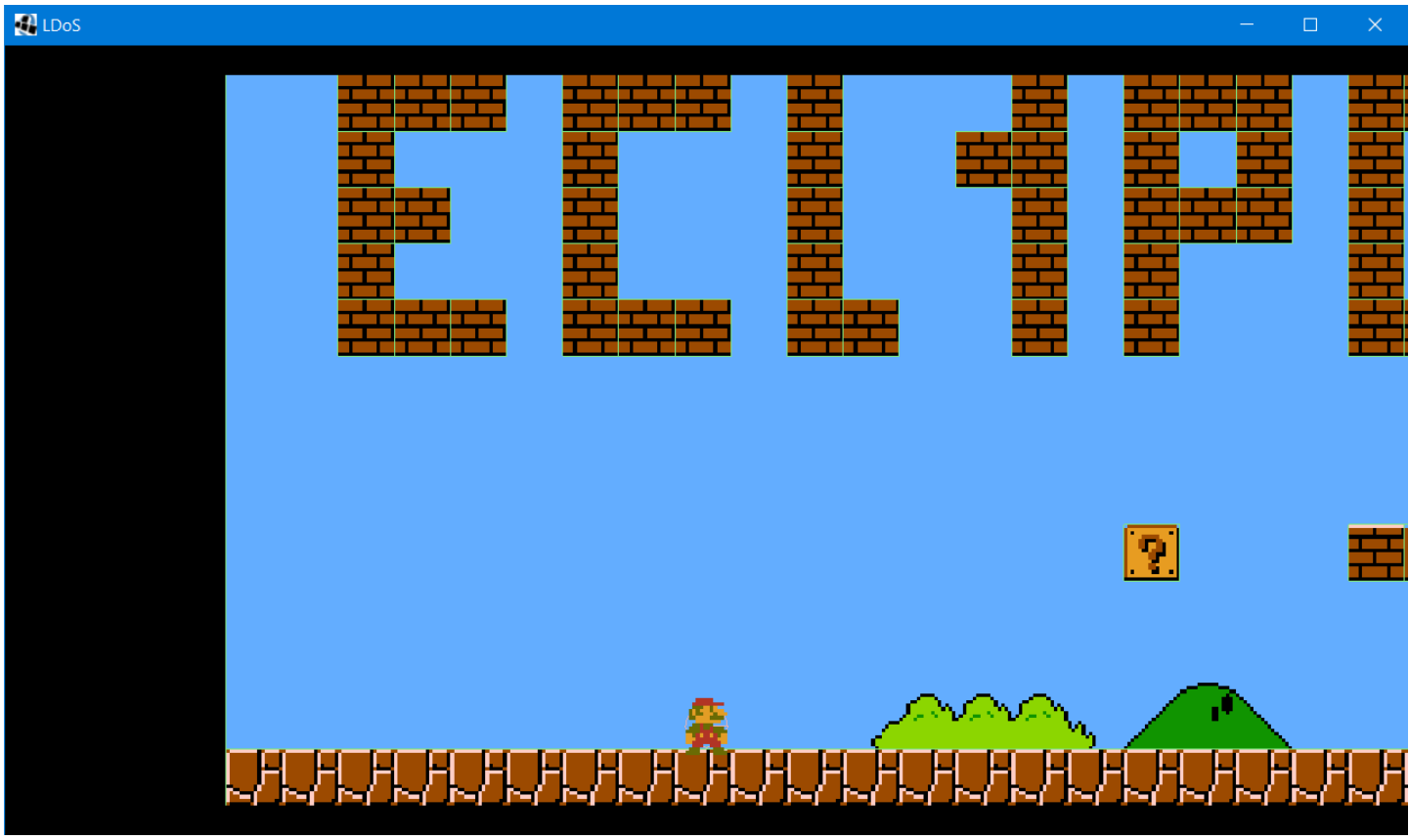

Рис. 2: Игровое окно

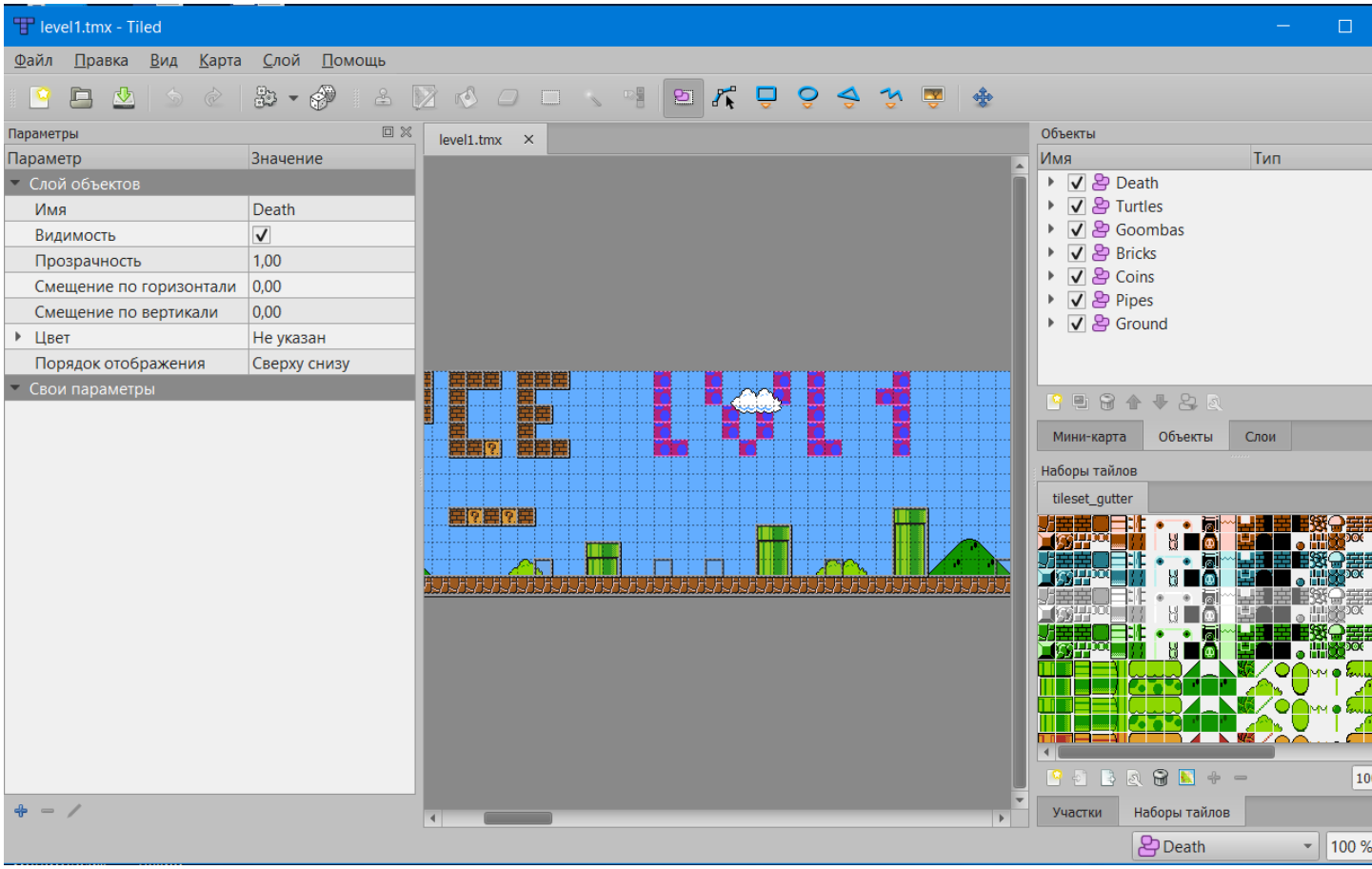

Рис. 3: Разработка карты

Рис. 4: Game Over Screen

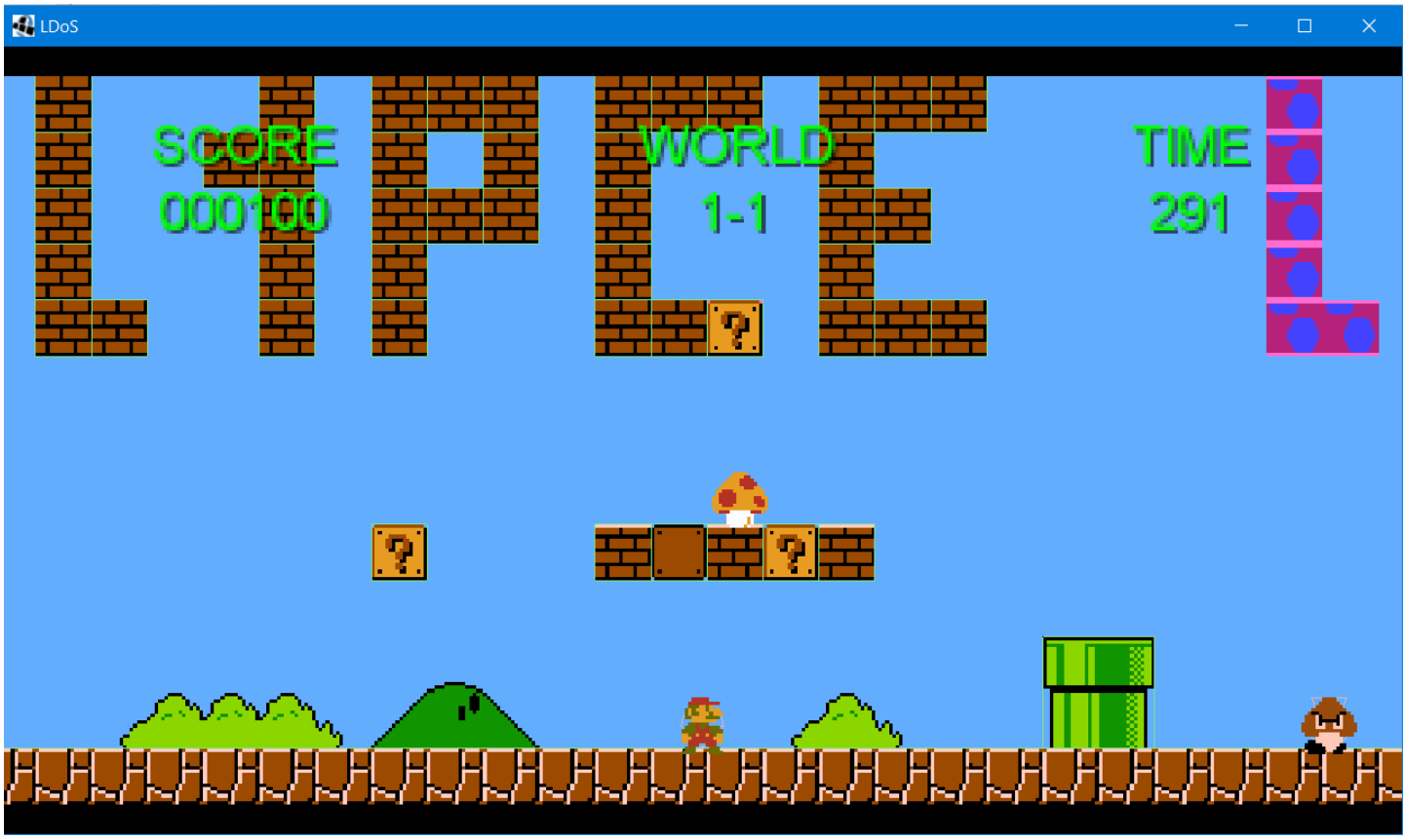

Рис. 5: Mushroom

Рис. 6: Противники

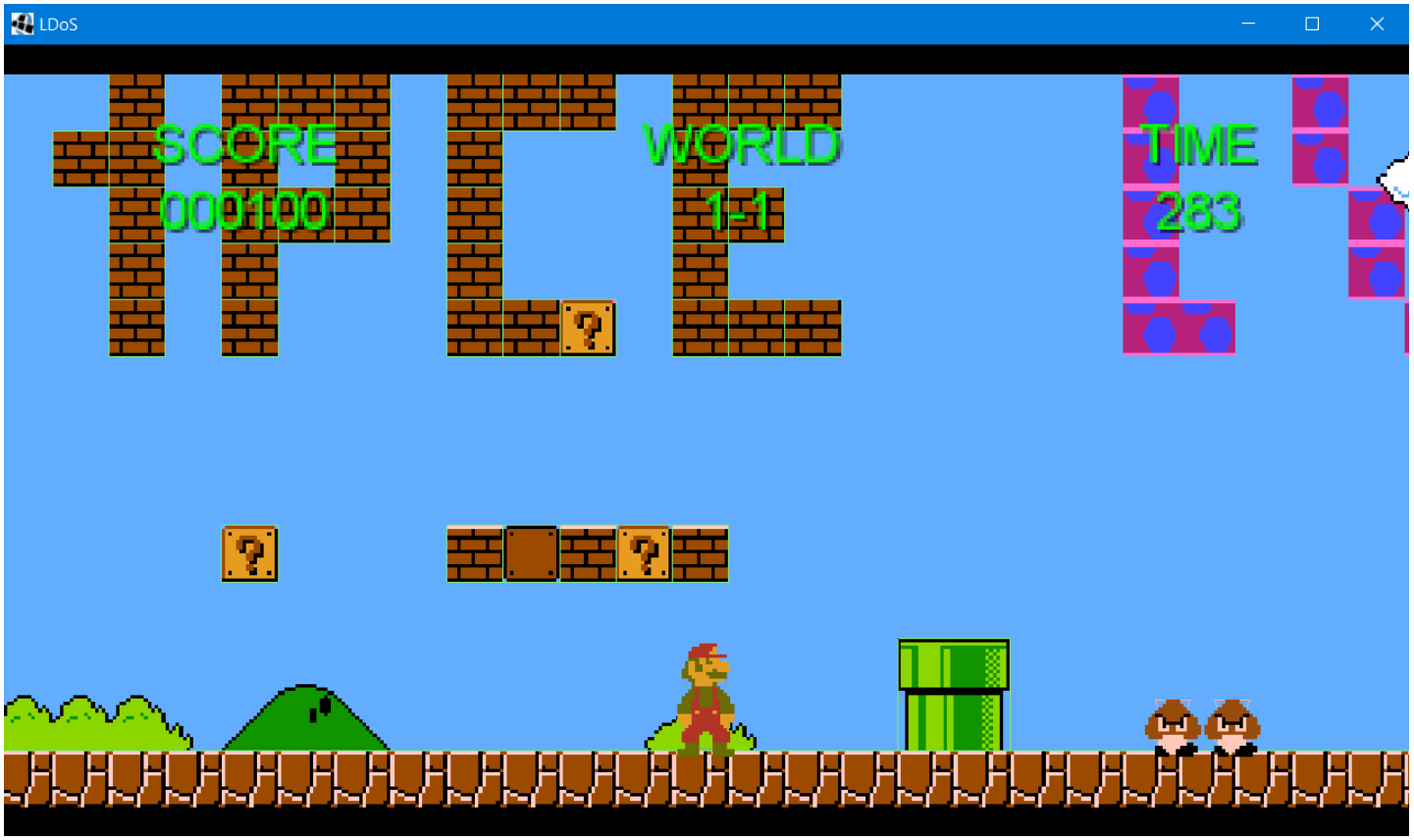

Рис. 7: Разбивание блока (до)

Рис. 8: Разбивание блока (после)

3.6 Перспективы развития приложения

Планируется реализовать следующую функциональность:

- Менеджер уровней
- Добавить сюжетную линию
- Добавить новых противников и оружие с различной механикой
- Добавить экран меню

3.7 Вывод

Были описаны используемые средства разработки. Кратко описан фреймворк LibGDX и обосновано его использование. Был поэтапно описан процесс разработки приложения.

4 Процесс обеспечения качества и тестирование игрового приложения Лабиринт

Для проверки корректности работы приложения использовалось ручное тестирование.

4.1 Тестирование

Для ручного тестирования в приложение были добавлены логи из библиотеки LibGDX. Любое событие отображалось в консоле разработки.
Тестирование проводилось по следующему сценарию.

- Запустить приложение
- Проверить верность исполнения команд героя (прыжок, движение влево и вправо)

- Разбить блоки
- Различным образом взаимодействовать с врагами: прыгать на них, сталкиваться сбоку
- Взаимодействовать с интерактивными объектами карты: брыгать на них сверху, пытаться разбить головой
- Проверить смерть персонажа от противников и по истечению времени
- Проверить верность отображения экрана Game Over
- Кликнуть по нему и проверить запускание игры заново

4.2 Вывод

Был описан процесс обеспечения качества и тестирования приложения.

5 Выводы

Было спроектирована и реализована 2D-игра жанра платформер. В процессе разработки были тточены навики програмирования на языке Java в среде InteliJ IDEA. Была изучена сторонняя библиотека LibGDX и инструмент Tiled Map Editor. В дальнейшем планируется продолжение поддержки и улучшение приложения, а также исправление текущих недочётов.

6 Приложение

Исходный ход можно найти в репозитории[3] на ресурсе GitHub

6.1 Листинги

```
1  package com.mygdx.game.scenes;
2
3
4  import com.badlogic.gdx.Gdx;
5  import com.badlogic.gdx.graphics.Color;
6  import com.badlogic.gdx.graphics.OrthographicCamera;
7  import com.badlogic.gdx.graphics.g2d.BitmapFont;
8  import com.badlogic.gdx.graphics.g2d.SpriteBatch;
9  import com.badlogic.gdx.scenes.scene2d.Stage;
10 import com.badlogic.gdx.scenes.scene2d.ui.Label;
11 import com.badlogic.gdx.scenes.scene2d.ui.Table;
12 import com.badlogic.gdx.utils.Disposable;
13 import com.badlogic.gdx.utils.viewport.FitViewport;
14 import com.badlogic.gdx.utils.viewport.Viewport;
15 import com.mygdx.game.LDoS;
16
17
18 public class Hud implements Disposable{
19
20     //Scene2D.ui Stage and its own Viewport for HUD
21     public Stage stage;
22     private Viewport viewport;
23
24     //score/time Tracking Variables
25     private Integer worldTimer;
26     private boolean timeUp; // true when the world timer reaches 0
27     private float timeCount;
28     private static Integer score;
29
30     //Scene2D widgets
31     private Label countdownLabel;
32     private static Label scoreLabel;
33     private Label timeLabel;
34     private Label levelLabel;
35     private Label worldLabel;
36     private Label marioLabel;
37
```

[3] https://github.com/Ecl1pce/LDoS

```
38      public Hud(SpriteBatch sb){
39          //define our tracking variables
40          worldTimer = 300;
41          timeCount = 0;
42          score = 0;
43
44
45          //setup the HUD viewport using a new camera seperate from our gamecam
46          //define our stage using that viewport and our games spritebatch
47          viewport = new FitViewport(LDoS.V_WIDTH, LDoS.V_HEIGHT, new OrthographicCamera());
48          stage = new Stage(viewport, sb);
49
50          //define a table used to organize our hud's labels
51          Table table = new Table();
52          //Top-Align table
53          table.top();
54          //make the table fill the entire stage
55          table.setFillParent(true);
56
57          //define labels using the String, and a Label style consisting of a font and color
58          countdownLabel = new Label(String.format("%03d", worldTimer), new Label.LabelStyle(new
        ↪  BitmapFont(), Color.GREEN));
59          scoreLabel =new Label(String.format("%06d", score), new Label.LabelStyle(new
        ↪ BitmapFont(), Color.GREEN));
60          timeLabel = new Label("TIME", new Label.LabelStyle(new BitmapFont(), Color.GREEN));
61          levelLabel = new Label("1-1", new Label.LabelStyle(new BitmapFont(), Color.GREEN));
62          worldLabel = new Label("WORLD", new Label.LabelStyle(new BitmapFont(), Color.GREEN));
63          marioLabel = new Label("SCORE", new Label.LabelStyle(new BitmapFont(), Color.GREEN));
64
65          //add labels to table, padding the top, and giving them all equal width with expandX
66          table.add(marioLabel).expandX().padTop(10);
67          table.add(worldLabel).expandX().padTop(10);
68          table.add(timeLabel).expandX().padTop(10);
69          //add second row to table
70          table.row();
71          table.add(scoreLabel).expandX();
72          table.add(levelLabel).expandX();
73          table.add(countdownLabel).expandX();
74
75          //add table to stage
76          stage.addActor(table);
77
78      }
79
80      public void update(float dt){
81          timeCount += dt;
82          if(timeCount >= 1){
83              if (worldTimer > 0) {
84                  worldTimer--;
85              } else {
86                  timeUp = true;
87              }
88              countdownLabel.setText(String.format("%03d", worldTimer));
89              timeCount = 0;
90          }
91      }
92
93      public static void addScore(int value){
94          score += value;
95          scoreLabel.setText(String.format("%06d", score));
96      }
97
98      @Override
99      public void dispose() { stage.dispose(); }
100
101     public boolean isTimeUp() { return timeUp; }
102 }
```

```
1  package com.mygdx.game.screens;
2
3  import com.badlogic.gdx.Game;
4  import com.badlogic.gdx.Gdx;
5  import com.badlogic.gdx.Screen;
6  import com.badlogic.gdx.graphics.Color;
7  import com.badlogic.gdx.graphics.GL20;
8  import com.badlogic.gdx.graphics.OrthographicCamera;
```

```
9  import com.badlogic.gdx.graphics.g2d.BitmapFont;
10 import com.badlogic.gdx.scenes.scene2d.Event;
11 import com.badlogic.gdx.scenes.scene2d.EventListener;
12 import com.badlogic.gdx.scenes.scene2d.InputEvent;
13 import com.badlogic.gdx.scenes.scene2d.InputListener;
14 import com.badlogic.gdx.scenes.scene2d.Stage;
15 import com.badlogic.gdx.scenes.scene2d.ui.Label;
16 import com.badlogic.gdx.scenes.scene2d.ui.Table;
17 import com.badlogic.gdx.scenes.scene2d.ui.TextButton;
18 import com.badlogic.gdx.utils.viewport.FitViewport;
19 import com.badlogic.gdx.utils.viewport.Viewport;
20 import com.mygdx.game.LDoS;
21
22
23 public class GameOverScreen implements Screen {
24     private Viewport viewport;
25     private Stage stage;
26
27     private Game game;
28
29     public GameOverScreen(Game game){
30         this.game = game;
31         viewport = new FitViewport(LDoS.V_WIDTH, LDoS.V_HEIGHT, new OrthographicCamera());
32         stage = new Stage(viewport, ((LDoS) game).batch);
33
34         Label.LabelStyle font = new Label.LabelStyle(new BitmapFont(), Color.GREEN);
35
36         Table table = new Table();
37         table.center();
38         table.setFillParent(true);
39
40         Label gameOverLabel = new Label("GAME OVER", font);
41         Label playAgainLabel = new Label("Click to try again", font);
42
43         table.add(gameOverLabel).expandX();
44         table.row();
45         table.add(playAgainLabel).expandX().padTop(10f);
46
47         stage.addActor(table);
48     }
49
50     @Override
51     public void show() {
52
53     }
54
55     @Override
56     public void render(float delta) {
57         if(Gdx.input.justTouched()) {
58             game.setScreen(new PlayScreen((LDoS) game));
59             dispose();
60         }
61         Gdx.gl.glClearColor(0, 0, 0, 1);
62         Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
63         stage.draw();
64     }
65
66     @Override
67     public void resize(int width, int height) {
68
69     }
70
71     @Override
72     public void pause() {
73
74     }
75
76     @Override
77     public void resume() {
78
79     }
80
81     @Override
82     public void hide() {
83
84     }
```

```
85
86      @Override
87      public void dispose() {
88          stage.dispose();
89      }
90  }
```

```
1  package com.mygdx.game.screens;
2
3  import com.badlogic.gdx.Gdx;
4  import com.badlogic.gdx.Input;
5  import com.badlogic.gdx.Screen;
6  import com.badlogic.gdx.audio.Music;
7  import com.badlogic.gdx.graphics.GL20;
8  import com.badlogic.gdx.graphics.OrthographicCamera;
9  import com.badlogic.gdx.graphics.g2d.TextureAtlas;
10 import com.badlogic.gdx.maps.tiled.TiledMap;
11 import com.badlogic.gdx.maps.tiled.TmxMapLoader;
12 import com.badlogic.gdx.maps.tiled.renderers.OrthogonalTiledMapRenderer;
13 import com.badlogic.gdx.math.Vector2;
14 import com.badlogic.gdx.physics.box2d.Box2DDebugRenderer;
15 import com.badlogic.gdx.physics.box2d.World;
16 import com.badlogic.gdx.utils.Array;
17 import com.badlogic.gdx.utils.viewport.FitViewport;
18 import com.badlogic.gdx.utils.viewport.Viewport;
19 import com.mygdx.game.LDoS;
20 import com.mygdx.game.scenes.Hud;
21 import com.mygdx.game.sprites.enemies.Enemy;
22 import com.mygdx.game.sprites.items.Item;
23 import com.mygdx.game.sprites.items.ItemDef;
24 import com.mygdx.game.sprites.items.Mushroom;
25 import com.mygdx.game.sprites.Hero;
26 import com.mygdx.game.tools.B2WorldCreator;
27 import com.mygdx.game.tools.WorldContactListener;
28
29 import java.util.PriorityQueue;
30 import java.util.concurrent.LinkedBlockingQueue;
31
32
33 public class PlayScreen implements Screen{
34     //Reference to our Game, used to set Screens
35     private LDoS game;
36     private TextureAtlas atlas;
37     public static boolean alreadyDestroyed = false;
38
39     //basic playscreen variables
40     private OrthographicCamera gamecam;
41     private Viewport gamePort;
42     private Hud hud;
43
44     //Tiled map variables
45     private TmxMapLoader maploader;
46     private TiledMap map;
47     private OrthogonalTiledMapRenderer renderer;
48
49     //Box2d variables
50     private World world;
51     private Box2DDebugRenderer b2dr;
52     private B2WorldCreator creator;
53
54     //sprites
55     private Hero player;
56
57     private Music music;
58
59     private Array<Item> items;
60     private LinkedBlockingQueue<ItemDef> itemsToSpawn;
61
62
63     public PlayScreen(LDoS game){
64         atlas = new TextureAtlas("Mario_and_Enemies.pack");
65
66         this.game = game;
67         //create cam used to follow mario through cam world
68         gamecam = new OrthographicCamera();
69
```

```
70          //create a FitViewport to maintain virtual aspect ratio despite screen size
71          gamePort = new FitViewport(LDoS.V_WIDTH / LDoS.PPM, LDoS.V_HEIGHT / LDoS.PPM, gamecam)
    ↪ ;
72
73          //create our game HUD for scores/timers/level info
74          hud = new Hud(game.batch);
75
76          //Load our map and setup our map renderer
77          maploader = new TmxMapLoader();
78          map = maploader.load("level1.tmx");
79          renderer = new OrthogonalTiledMapRenderer(map, 1  / LDoS.PPM);
80
81          //initially set our gamcam to be centered correctly at the start of map
82          gamecam.position.set(gamePort.getWorldWidth() / 2, gamePort.getWorldHeight() / 2, 0);
83
84          //create our Box2D world, setting no gravity in X, -10 gravity in Y, and allow bodies
    ↪ to sleep
85          world = new World(new Vector2(0, -10), true);
86          //allows for debug lines of our box2d world.
87          b2dr = new Box2DDebugRenderer();
88
89          creator = new B2WorldCreator(this);
90
91          //create mario in our game world
92          player = new Hero(this);
93
94          world.setContactListener(new WorldContactListener());
95
96          music = LDoS.manager.get("audio/music/mario_music.ogg", Music.class);
97          music.setLooping(true);
98          music.setVolume(0.2f);
99          //music.play();
100
101         items = new Array<Item>();
102         itemsToSpawn = new LinkedBlockingQueue<ItemDef>();
103     }
104
105     public void spawnItem(ItemDef idef){
106         itemsToSpawn.add(idef);
107     }
108
109
110     public void handleSpawningItems(){
111         if(!itemsToSpawn.isEmpty()){
112             ItemDef idef = itemsToSpawn.poll();
113             if(idef.type == Mushroom.class){
114                 items.add(new Mushroom(this, idef.position.x, idef.position.y));
115             }
116         }
117     }
118
119
120     public TextureAtlas getAtlas(){
121         return atlas;
122     }
123
124     @Override
125     public void show() {
126
127
128     }
129
130     public void handleInput(float dt){
131         //control our player using immediate impulses
132         if(player.currentState != Hero.State.DEAD) {
133             if (Gdx.input.isKeyJustPressed(Input.Keys.UP))
134                 player.jump();
135             if (Gdx.input.isKeyPressed(Input.Keys.RIGHT) && player.b2body.getLinearVelocity().
    ↪ x <= 2)
136                 player.b2body.applyLinearImpulse(new Vector2(0.1f, 0), player.b2body.
    ↪ getWorldCenter(), true);
137             if (Gdx.input.isKeyPressed(Input.Keys.LEFT) && player.b2body.getLinearVelocity().x
    ↪  >= -2)
138                 player.b2body.applyLinearImpulse(new Vector2(-0.1f, 0), player.b2body.
    ↪ getWorldCenter(), true);
139             //if (Gdx.input.isKeyJustPressed(Input.Keys.SPACE))
```

```
140             //    player.fire();
141         }
142 
143     }
144 
145     public void update(float dt){
146         //handle user input first
147         handleInput(dt);
148         handleSpawningItems();
149 
150         //takes 1 step in the physics simulation(60 times per second)
151         world.step(1 / 60f, 6, 2);
152 
153         player.update(dt);
154         for(Enemy enemy : creator.getEnemies()) {
155             enemy.update(dt);
156             if(enemy.getX() < player.getX() + 224 / LDoS.PPM) {
157                 enemy.b2body.setActive(true);
158             }
159         }
160 
161         for(Item item : items)
162             item.update(dt);
163 
164         hud.update(dt);
165 
166         //attach our gamecam to our players.x coordinate
167         if(player.currentState != Hero.State.DEAD) {
168             gamecam.position.x = player.b2body.getPosition().x;
169         }
170 
171         //update our gamecam with correct coordinates after changes
172         gamecam.update();
173         //tell our renderer to draw only what our camera can see in our game world.
174         renderer.setView(gamecam);
175 
176     }
177 
178 
179     @Override
180     public void render(float delta) {
181         //separate our update logic from render
182         update(delta);
183 
184         //Clear the game screen with Black
185         Gdx.gl.glClearColor(0, 0, 0, 1);
186         Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
187 
188         //render our game map
189         renderer.render();
190 
191         //renderer our Box2DDebugLines
192         b2dr.render(world, gamecam.combined);
193 
194         game.batch.setProjectionMatrix(gamecam.combined);
195         game.batch.begin();
196         player.draw(game.batch);
197         for (Enemy enemy : creator.getEnemies())
198             enemy.draw(game.batch);
199         for (Item item : items)
200             item.draw(game.batch);
201         game.batch.end();
202 
203         //Set our batch to now draw what the Hud camera sees.
204         game.batch.setProjectionMatrix(hud.stage.getCamera().combined);
205         hud.stage.draw();
206 
207         if(gameOver()){
208             game.setScreen(new GameOverScreen(game));
209             dispose();
210         }
211 
212     }
213 
214     public boolean gameOver(){
215         if(player.currentState == Hero.State.DEAD && player.getStateTimer() > 3){
```

```
216             return true;
217         }
218         return false;
219     }
220
221     @Override
222     public void resize(int width, int height) {
223         //updated our game viewport
224         gamePort.update(width,height);
225
226     }
227
228     public TiledMap getMap(){
229         return map;
230     }
231     public World getWorld(){
232         return world;
233     }
234
235     @Override
236     public void pause() {
237
238     }
239
240     @Override
241     public void resume() {
242
243     }
244
245     @Override
246     public void hide() {
247
248     }
249
250     @Override
251     public void dispose() {
252         //dispose of all our opened resources
253         map.dispose();
254         renderer.dispose();
255         world.dispose();
256         b2dr.dispose();
257         hud.dispose();
258     }
259
260     public Hud getHud(){ return hud; }
261 }
```

```
1  package com.mygdx.game.sprites.enemies;
2
3  import com.badlogic.gdx.graphics.g2d.Sprite;
4  import com.badlogic.gdx.math.Vector2;
5  import com.badlogic.gdx.physics.box2d.Body;
6  import com.badlogic.gdx.physics.box2d.World;
7  import com.mygdx.game.screens.PlayScreen;
8  import com.mygdx.game.sprites.Hero;
9
10
11 public abstract class Enemy extends Sprite {
12     protected World world;
13     protected PlayScreen screen;
14     public Body b2body;
15     public Vector2 velocity;
16
17     public Enemy(PlayScreen screen, float x, float y){
18         this.world = screen.getWorld();
19         this.screen = screen;
20         setPosition(x, y);
21         defineEnemy();
22         velocity = new Vector2(-1, -2);
23         b2body.setActive(false);
24     }
25
26     protected abstract void defineEnemy();
27     public abstract void update(float dt);
28     public abstract void hitOnHead(Hero mario);
29     public abstract void hitByEnemy(Enemy enemy);
```

```
30
31      public void reverseVelocity(boolean x, boolean y){
32          if(x)
33              velocity.x = -velocity.x;
34          if(y)
35              velocity.y = -velocity.y;
36      }
37  }
```

```
1   package com.mygdx.game.sprites.enemies;
2
3   import com.badlogic.gdx.audio.Sound;
4   import com.badlogic.gdx.graphics.g2d.Animation;
5   import com.badlogic.gdx.graphics.g2d.Batch;
6   import com.badlogic.gdx.graphics.g2d.TextureRegion;
7   import com.badlogic.gdx.math.Vector2;
8   import com.badlogic.gdx.physics.box2d.BodyDef;
9   import com.badlogic.gdx.physics.box2d.CircleShape;
10  import com.badlogic.gdx.physics.box2d.FixtureDef;
11  import com.badlogic.gdx.physics.box2d.PolygonShape;
12  import com.badlogic.gdx.utils.Array;
13  import com.mygdx.game.LDoS;
14  import com.mygdx.game.screens.PlayScreen;
15  import com.mygdx.game.sprites.Hero;
16  //import com.mygdx.game.sprites.enemies.Enemy;
17
18
19  public class Goomba extends Enemy
20  {
21      private float stateTime;
22      private Animation walkAnimation;
23      private Array<TextureRegion> frames;
24      private boolean setToDestroy;
25      private boolean destroyed;
26      float angle;
27
28
29      public Goomba(PlayScreen screen, float x, float y) {
30          super(screen, x, y);
31          frames = new Array<TextureRegion>();
32          for(int i = 0; i < 2; i++)
33              frames.add(new TextureRegion(screen.getAtlas().findRegion("goomba"), i * 16, 0,
    ↪ 16, 16));
34          walkAnimation = new Animation(0.4f, frames);
35          stateTime = 0;
36          setBounds(getX(), getY(), 16 / LDoS.PPM, 16 / LDoS.PPM);
37          setToDestroy = false;
38          destroyed = false;
39          angle = 0;
40      }
41
42      public void update(float dt){
43          stateTime += dt;
44          if(setToDestroy && !destroyed){
45              world.destroyBody(b2body);
46              destroyed = true;
47              setRegion(new TextureRegion(screen.getAtlas().findRegion("goomba"), 32, 0, 16, 16)
    ↪ );
48              stateTime = 0;
49          }
50          else if(!destroyed) {
51              b2body.setLinearVelocity(velocity);
52              setPosition(b2body.getPosition().x - getWidth() / 2, b2body.getPosition().y -
    ↪ getHeight() / 2);
53              setRegion(walkAnimation.getKeyFrame(stateTime, true));
54          }
55      }
56
57      @Override
58      protected void defineEnemy() {
59          BodyDef bdef = new BodyDef();
60          bdef.position.set(getX(), getY());
61          bdef.type = BodyDef.BodyType.DynamicBody;
62          b2body = world.createBody(bdef);
63
64          FixtureDef fdef = new FixtureDef();
```

```
65          CircleShape shape = new CircleShape();
66          shape.setRadius(6 / LDoS.PPM);
67          fdef.filter.categoryBits = LDoS.ENEMY_BIT;
68          fdef.filter.maskBits = LDoS.GROUND_BIT |
69                  LDoS.COIN_BIT |
70                  LDoS.BRICK_BIT |
71                  LDoS.ENEMY_BIT |
72                  LDoS.OBJECT_BIT |
73                  LDoS.MARIO_BIT;
74
75          fdef.shape = shape;
76          b2body.createFixture(fdef).setUserData(this);
77
78          //Create the Head here:
79          PolygonShape head = new PolygonShape();
80          Vector2[] vertice = new Vector2[4];
81          vertice[0] = new Vector2(-5, 8).scl(1 / LDoS.PPM);
82          vertice[1] = new Vector2(5, 8).scl(1 / LDoS.PPM);
83          vertice[2] = new Vector2(-3, 3).scl(1 / LDoS.PPM);
84          vertice[3] = new Vector2(3, 3).scl(1 / LDoS.PPM);
85          head.set(vertice);
86
87          fdef.shape = head;
88          fdef.restitution = 0.5f;
89          fdef.filter.categoryBits = LDoS.ENEMY_HEAD_BIT;
90          b2body.createFixture(fdef).setUserData(this);
91
92      }
93
94      public void draw(Batch batch){
95          if(!destroyed || stateTime < 1)
96              super.draw(batch);
97      }
98
99
100
101     @Override
102     public void hitOnHead(Hero mario) {
103         setToDestroy = true;
104         LDoS.manager.get("audio/sounds/stomp.wav", Sound.class).play();
105     }
106
107     @Override
108     public void hitByEnemy(Enemy enemy) {
109         if(enemy instanceof Turtle && ((Turtle) enemy).currentState == Turtle.State.
    ↪ MOVING_SHELL)
110             setToDestroy = true;
111         else
112             reverseVelocity(true, false);
113     }
114 }
```

```
1  package com.mygdx.game.sprites.enemies;
2
3  import com.badlogic.gdx.graphics.g2d.Animation;
4  import com.badlogic.gdx.graphics.g2d.TextureRegion;
5  import com.badlogic.gdx.math.Vector2;
6  import com.badlogic.gdx.physics.box2d.BodyDef;
7  import com.badlogic.gdx.physics.box2d.CircleShape;
8  import com.badlogic.gdx.physics.box2d.FixtureDef;
9  import com.badlogic.gdx.physics.box2d.PolygonShape;
10 import com.badlogic.gdx.utils.Array;
11 import com.mygdx.game.LDoS;
12 import com.mygdx.game.screens.PlayScreen;
13 import com.mygdx.game.sprites.Hero;
14
15
16 public class Turtle extends Enemy {
17     public static final int KICK_LEFT = -2;
18     public static final int KICK_RIGHT = 2;
19     public enum State {WALKING, MOVING_SHELL, STANDING_SHELL}
20     public State currentState;
21     public State previousState;
22     private float stateTime;
23     private Animation walkAnimation;
24     private Array<TextureRegion> frames;
```

```
25      private TextureRegion shell;
26      private boolean setToDestroy;
27      private boolean destroyed;
28
29
30      public Turtle(PlayScreen screen, float x, float y) {
31          super(screen, x, y);
32          frames = new Array<TextureRegion>();
33          frames.add(new TextureRegion(screen.getAtlas().findRegion("turtle"), 0, 0, 16, 24));
34          frames.add(new TextureRegion(screen.getAtlas().findRegion("turtle"), 16, 0, 16, 24));
35          shell = new TextureRegion(screen.getAtlas().findRegion("turtle"), 64, 0, 16, 24);
36          walkAnimation = new Animation(0.2f, frames);
37          currentState = previousState = State.WALKING;
38
39          setBounds(getX(), getY(), 16 / LDoS.PPM, 24 / LDoS.PPM);
40
41      }
42
43      @Override
44      protected void defineEnemy() {
45          BodyDef bdef = new BodyDef();
46          bdef.position.set(getX(), getY());
47          bdef.type = BodyDef.BodyType.DynamicBody;
48          b2body = world.createBody(bdef);
49
50          FixtureDef fdef = new FixtureDef();
51          CircleShape shape = new CircleShape();
52          shape.setRadius(6 / LDoS.PPM);
53          fdef.filter.categoryBits = LDoS.ENEMY_BIT;
54          fdef.filter.maskBits = LDoS.GROUND_BIT |
55                  LDoS.COIN_BIT |
56                  LDoS.BRICK_BIT |
57                  LDoS.ENEMY_BIT |
58                  LDoS.OBJECT_BIT |
59                  LDoS.MARIO_BIT;
60
61          fdef.shape = shape;
62          b2body.createFixture(fdef).setUserData(this);
63
64          //Create the Head here:
65          PolygonShape head = new PolygonShape();
66          Vector2[] vertice = new Vector2[4];
67          vertice[0] = new Vector2(-5, 8).scl(1 / LDoS.PPM);
68          vertice[1] = new Vector2(5, 8).scl(1 / LDoS.PPM);
69          vertice[2] = new Vector2(-3, 3).scl(1 / LDoS.PPM);
70          vertice[3] = new Vector2(3, 3).scl(1 / LDoS.PPM);
71          head.set(vertice);
72
73          fdef.shape = head;
74          fdef.restitution = 1.8f;
75          fdef.filter.categoryBits = LDoS.ENEMY_HEAD_BIT;
76          b2body.createFixture(fdef).setUserData(this);
77      }
78
79      public TextureRegion getFrame(float dt){
80          TextureRegion region;
81
82          switch (currentState){
83              case MOVING_SHELL:
84              case STANDING_SHELL:
85                  region = shell;
86                  break;
87              case WALKING:
88              default:
89                  region = walkAnimation.getKeyFrame(stateTime, true);
90                  break;
91          }
92
93          if(velocity.x > 0 && region.isFlipX() == false){
94              region.flip(true, false);
95          }
96          if(velocity.x < 0 && region.isFlipX() == true){
97              region.flip(true, false);
98          }
99          stateTime = currentState == previousState ? stateTime + dt : 0;
100         //update previous state
```

```
        previousState = currentState;
        //return our final adjusted frame
        return region;
    }

    @Override
    public void update(float dt) {
        setRegion(getFrame(dt));
        if(currentState == State.STANDING_SHELL && stateTime > 5){
            currentState = State.WALKING;
            velocity.x = 1;
            System.out.println("WAKE UP SHELL");
        }

        setPosition(b2body.getPosition().x - getWidth() / 2, b2body.getPosition().y - 8 /LDoS.
    ↪ PPM);
        b2body.setLinearVelocity(velocity);
    }

    @Override
    public void hitOnHead(Hero mario) {
        if(currentState == State.STANDING_SHELL) {
            if(mario.b2body.getPosition().x > b2body.getPosition().x)
                velocity.x = -2;
            else
                velocity.x = 2;
            currentState = State.MOVING_SHELL;
            System.out.println("Set to moving shell");
        }
        else {
            currentState = State.STANDING_SHELL;
            velocity.x = 0;
        }
    }

    @Override
    public void hitByEnemy(Enemy enemy) {
        reverseVelocity(true, false);
    }

    public void kick(int direction){
        velocity.x = direction;
        currentState = State.MOVING_SHELL;
    }
}
```

```
package com.mygdx.game.sprites.items;

import com.badlogic.gdx.graphics.g2d.Batch;
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.World;
import com.mygdx.game.LDoS;
import com.mygdx.game.screens.PlayScreen;
import com.mygdx.game.sprites.Hero;

public abstract class Item extends Sprite {
    protected PlayScreen screen;
    protected World world;
    protected Vector2 velocity;
    protected boolean toDestroy;
    protected boolean destroyed;
    protected Body body;

    public Item(PlayScreen screen, float x, float y){
        this.screen = screen;
        this.world = screen.getWorld();
        toDestroy = false;
        destroyed = false;

        setPosition(x, y);
        setBounds(getX(), getY(), 16 / LDoS.PPM, 16 / LDoS.PPM);
        defineItem();
    }

```

```
    public abstract void defineItem();
    public abstract void use(Hero mario);

    public void update(float dt){
        if(toDestroy && !destroyed){
            world.destroyBody(body);
            destroyed = true;
        }
    }

    public void draw(Batch batch){
        if(!destroyed)
            super.draw(batch);
    }

    public void destroy(){
        toDestroy = true;
    }
    public void reverseVelocity(boolean x, boolean y){
        if(x)
            velocity.x = -velocity.x;
        if(y)
            velocity.y = -velocity.y;
    }
}
```

```
package com.mygdx.game.sprites.items;

import com.badlogic.gdx.math.Vector2;


public class ItemDef {
    public Vector2 position;
    public Class<?> type;

    public ItemDef(Vector2 position, Class<?> type){
        this.position = position;
        this.type = type;
    }
}
```

```
package com.mygdx.game.sprites.items;

import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.physics.box2d.BodyDef;
import com.badlogic.gdx.physics.box2d.CircleShape;
import com.badlogic.gdx.physics.box2d.FixtureDef;
import com.mygdx.game.LDoS;
import com.mygdx.game.screens.PlayScreen;
import com.mygdx.game.sprites.Hero;


public class Mushroom extends Item {
    public Mushroom(PlayScreen screen, float x, float y) {
        super(screen, x, y);
        setRegion(screen.getAtlas().findRegion("mushroom"), 0, 0, 16, 16);
        velocity = new Vector2(0.7f, 0);
    }

    @Override
    public void defineItem() {
        BodyDef bdef = new BodyDef();
        bdef.position.set(getX(), getY());
        bdef.type = BodyDef.BodyType.DynamicBody;
        body = world.createBody(bdef);

        FixtureDef fdef = new FixtureDef();
        CircleShape shape = new CircleShape();
        shape.setRadius(6 / LDoS.PPM);
        fdef.filter.categoryBits = LDoS.ITEM_BIT;
        fdef.filter.maskBits = LDoS.MARIO_BIT |
                LDoS.OBJECT_BIT |
                LDoS.GROUND_BIT |
                LDoS.COIN_BIT |
                LDoS.BRICK_BIT;
```

```
35
36          fdef.shape = shape;
37          body.createFixture(fdef).setUserData(this);
38      }
39
40      @Override
41      public void use(Hero mario) {
42          destroy();
43          mario.grow();
44      }
45
46      @Override
47      public void update(float dt) {
48          super.update(dt);
49          setPosition(body.getPosition().x - getWidth() / 2, body.getPosition().y - getHeight()
    ↪ / 2);
50          velocity.y = body.getLinearVelocity().y;
51          body.setLinearVelocity(velocity);
52      }
53  }
```

```
1   package com.mygdx.game.sprites.other;
2
3   import com.badlogic.gdx.graphics.g2d.Animation;
4   import com.badlogic.gdx.graphics.g2d.Sprite;
5   import com.badlogic.gdx.graphics.g2d.TextureRegion;
6   import com.badlogic.gdx.math.Vector2;
7   import com.badlogic.gdx.physics.box2d.Body;
8   import com.badlogic.gdx.physics.box2d.BodyDef;
9   import com.badlogic.gdx.physics.box2d.CircleShape;
10  import com.badlogic.gdx.physics.box2d.FixtureDef;
11  import com.badlogic.gdx.physics.box2d.PolygonShape;
12  import com.badlogic.gdx.physics.box2d.World;
13  import com.badlogic.gdx.utils.Array;
14  import com.mygdx.game.LDoS;
15  import com.mygdx.game.screens.PlayScreen;
16
17
18  public class FireBall extends Sprite {
19
20      PlayScreen screen;
21      World world;
22      Array<TextureRegion> frames;
23      Animation fireAnimation;
24      float stateTime;
25      boolean destroyed;
26      boolean setToDestroy;
27      boolean fireRight;
28
29      Body b2body;
30      public FireBall(PlayScreen screen, float x, float y, boolean fireRight){
31          this.fireRight = fireRight;
32          this.screen = screen;
33          this.world = screen.getWorld();
34          frames = new Array<TextureRegion>();
35          for(int i = 0; i < 4; i++){
36              frames.add(new TextureRegion(screen.getAtlas().findRegion("fireball"), i * 8, 0,
    ↪ 8, 8));
37          }
38          fireAnimation = new Animation(0.2f, frames);
39          setRegion(fireAnimation.getKeyFrame(0));
40          setBounds(x, y, 6 / LDoS.PPM, 6 / LDoS.PPM);
41          defineFireBall();
42      }
43
44      public void defineFireBall(){
45          BodyDef bdef = new BodyDef();
46          bdef.position.set(fireRight ? getX() + 12 /LDoS.PPM : getX() - 12 /LDoS.PPM, getY());
47          bdef.type = BodyDef.BodyType.DynamicBody;
48          if(!world.isLocked())
49              b2body = world.createBody(bdef);
50
51          FixtureDef fdef = new FixtureDef();
52          CircleShape shape = new CircleShape();
53          shape.setRadius(3 / LDoS.PPM);
54          fdef.filter.categoryBits = LDoS.FIREBALL_BIT;
```

```
        fdef.filter.maskBits = LDoS.GROUND_BIT |
                LDoS.COIN_BIT |
                LDoS.BRICK_BIT |
                LDoS.ENEMY_BIT |
                LDoS.OBJECT_BIT;

        fdef.shape = shape;
        fdef.restitution = 1;
        fdef.friction = 0;
        b2body.createFixture(fdef).setUserData(this);
        b2body.setLinearVelocity(new Vector2(fireRight ? 2 : -2, 2.5f));
    }

    public void update(float delta){
        stateTime += delta;
        setRegion(fireAnimation.getKeyFrame(stateTime, true));
        setPosition(b2body.getPosition().x - getWidth() / 2, b2body.getPosition().y -
    ↪ getHeight() / 2);
        if((stateTime > 3 || setToDestroy) && !destroyed) {
            world.destroyBody(b2body);
            destroyed = true;
        }
        if(b2body.getLinearVelocity().y > 2f)
            b2body.setLinearVelocity(b2body.getLinearVelocity().x, 2f);
        if((fireRight && b2body.getLinearVelocity().x < 0) || (!fireRight && b2body.
    ↪ getLinearVelocity().x > 0))
            setToDestroy();
    }

    public void setToDestroy(){
        setToDestroy = true;
    }

    public boolean isDestroyed(){
        return destroyed;
    }


}
```

```
package com.mygdx.game.sprites.tileObjects;

import com.badlogic.gdx.audio.Sound;
import com.badlogic.gdx.maps.MapObject;
import com.badlogic.gdx.math.Rectangle;
import com.mygdx.game.LDoS;
import com.mygdx.game.scenes.Hud;
import com.mygdx.game.screens.PlayScreen;
import com.mygdx.game.sprites.Hero;

public class Brick extends InteractiveTileObject {
    public Brick(PlayScreen screen, MapObject object){
        super(screen, object);
        fixture.setUserData(this);
        setCategoryFilter(LDoS.BRICK_BIT);
    }

    @Override
    public void onHeadHit(Hero mario) {
        if(mario.isBig()) {
            setCategoryFilter(LDoS.DESTROYED_BIT);
            getCell().setTile(null);
            Hud.addScore(200);
            LDoS.manager.get("audio/sounds/breakblock.wav", Sound.class).play();
        }
        LDoS.manager.get("audio/sounds/bump.wav", Sound.class).play();
    }

}
```

```
package com.mygdx.game.sprites.tileObjects;

import com.badlogic.gdx.audio.Sound;
import com.badlogic.gdx.audio.Music;
import com.badlogic.gdx.maps.MapObject;
```

```
import com.badlogic.gdx.maps.tiled.TiledMapTileSet;
import com.badlogic.gdx.math.Rectangle;
import com.badlogic.gdx.math.Vector2;
import com.mygdx.game.LDoS;
import com.mygdx.game.scenes.Hud;
import com.mygdx.game.screens.PlayScreen;
import com.mygdx.game.sprites.items.ItemDef;
import com.mygdx.game.sprites.items.Mushroom;
import com.mygdx.game.sprites.Hero;


public class Coin extends InteractiveTileObject {
    private static TiledMapTileSet tileSet;
    private final int BLANK_COIN = 28;

    public Coin(PlayScreen screen, MapObject object){
        super(screen, object);
        tileSet = map.getTileSets().getTileSet("tileset_gutter");
        fixture.setUserData(this);
        setCategoryFilter(LDoS.COIN_BIT);
    }

    @Override
    public void onHeadHit(Hero mario) {
        if(getCell().getTile().getId() == BLANK_COIN)
            LDoS.manager.get("audio/sounds/bump.wav", Sound.class).play();
        else {
            if(object.getProperties().containsKey("mushroom")) {
                screen.spawnItem(new ItemDef(new Vector2(body.getPosition().x, body.
    ↪ getPosition().y + 16 / LDoS.PPM),
                        Mushroom.class));
                LDoS.manager.get("audio/sounds/50_cent-in_da_club.mp3", Sound.class).play(0.2f
    ↪ );
            }
            else
                LDoS.manager.get("audio/sounds/coin.wav", Sound.class).play();
            getCell().setTile(tileSet.getTile(BLANK_COIN));
            Hud.addScore(100);
        }
    }
}
```

```
package com.mygdx.game.sprites.tileObjects;

import com.badlogic.gdx.audio.Sound;
import com.badlogic.gdx.maps.MapObject;
import com.badlogic.gdx.math.Rectangle;
import com.mygdx.game.LDoS;
import com.mygdx.game.scenes.Hud;
import com.mygdx.game.screens.PlayScreen;
import com.mygdx.game.sprites.Hero;

public class DeadZone extends InteractiveTileObject {
    public DeadZone(PlayScreen screen, MapObject object){
        super(screen, object);
        fixture.setUserData(this);
        setCategoryFilter(LDoS.DEATH_BIT);
    }
    @Override
    public void onHeadHit(Hero mario) {

    }
}
```

```
package com.mygdx.game.sprites.tileObjects;

import com.badlogic.gdx.maps.MapObject;
import com.badlogic.gdx.maps.objects.RectangleMapObject;
import com.badlogic.gdx.maps.tiled.TiledMap;
import com.badlogic.gdx.maps.tiled.TiledMapTile;
import com.badlogic.gdx.maps.tiled.TiledMapTileLayer;
import com.badlogic.gdx.math.Rectangle;
import com.badlogic.gdx.physics.box2d.Body;
import com.badlogic.gdx.physics.box2d.BodyDef;
import com.badlogic.gdx.physics.box2d.Filter;
```

```
12 import com.badlogic.gdx.physics.box2d.Fixture;
13 import com.badlogic.gdx.physics.box2d.FixtureDef;
14 import com.badlogic.gdx.physics.box2d.PolygonShape;
15 import com.badlogic.gdx.physics.box2d.World;
16 import com.badlogic.gdx.utils.Array;
17 import com.mygdx.game.LDoS;
18 import com.mygdx.game.scenes.Hud;
19 import com.mygdx.game.screens.PlayScreen;
20 import com.mygdx.game.sprites.Hero;
21
22
23 public abstract class InteractiveTileObject {
24     protected World world;
25     protected TiledMap map;
26     protected Rectangle bounds;
27     protected Body body;
28     protected PlayScreen screen;
29     protected MapObject object;
30
31     protected Fixture fixture;
32
33     public InteractiveTileObject(PlayScreen screen, MapObject object){
34         this.object = object;
35         this.screen = screen;
36         this.world = screen.getWorld();
37         this.map = screen.getMap();
38         this.bounds = ((RectangleMapObject) object).getRectangle();
39
40         BodyDef bdef = new BodyDef();
41         FixtureDef fdef = new FixtureDef();
42         PolygonShape shape = new PolygonShape();
43
44         bdef.type = BodyDef.BodyType.StaticBody;
45         bdef.position.set((bounds.getX() + bounds.getWidth() / 2) / LDoS.PPM, (bounds.getY() +
    ↪  bounds.getHeight() / 2) / LDoS.PPM);
46
47         body = world.createBody(bdef);
48
49         shape.setAsBox(bounds.getWidth() / 2 / LDoS.PPM, bounds.getHeight() / 2 / LDoS.PPM);
50         fdef.shape = shape;
51         fixture = body.createFixture(fdef);
52
53     }
54
55     public abstract void onHeadHit(Hero mario);
56     public void setCategoryFilter(short filterBit){
57         Filter filter = new Filter();
58         filter.categoryBits = filterBit;
59         fixture.setFilterData(filter);
60     }
61
62     public TiledMapTileLayer.Cell getCell(){
63         TiledMapTileLayer layer = (TiledMapTileLayer) map.getLayers().get(1);
64         return layer.getCell((int)(body.getPosition().x * LDoS.PPM / 16),
65                 (int)(body.getPosition().y * LDoS.PPM / 16));
66     }
67
68 }
```

```
1 package com.mygdx.game.sprites;
2
3 import com.badlogic.gdx.audio.Music;
4 import com.badlogic.gdx.audio.Sound;
5 import com.badlogic.gdx.graphics.g2d.Animation;
6 import com.badlogic.gdx.graphics.g2d.Batch;
7 import com.badlogic.gdx.graphics.g2d.Sprite;
8 import com.badlogic.gdx.graphics.g2d.TextureRegion;
9 import com.badlogic.gdx.math.Vector2;
10 import com.badlogic.gdx.physics.box2d.Body;
11 import com.badlogic.gdx.physics.box2d.BodyDef;
12 import com.badlogic.gdx.physics.box2d.CircleShape;
13 import com.badlogic.gdx.physics.box2d.EdgeShape;
14 import com.badlogic.gdx.physics.box2d.Filter;
15 import com.badlogic.gdx.physics.box2d.Fixture;
16 import com.badlogic.gdx.physics.box2d.FixtureDef;
17 import com.badlogic.gdx.physics.box2d.World;
```

```
18 import com.badlogic.gdx.utils.Array;
19 import com.mygdx.game.LDoS;
20 import com.mygdx.game.screens.PlayScreen;
21 import com.mygdx.game.sprites.other.FireBall;
22 import com.mygdx.game.sprites.enemies.*;
23
24
25 public class Hero extends Sprite {
26     public enum State { FALLING, JUMPING, STANDING, RUNNING, GROWING, DEAD };
27     public State currentState;
28     public State previousState;
29
30     public World world;
31     public Body b2body;
32
33     private TextureRegion marioStand;
34     private Animation marioRun;
35     private TextureRegion marioJump;
36     private TextureRegion marioDead;
37     private TextureRegion bigMarioStand;
38     private TextureRegion bigMarioJump;
39     private Animation bigMarioRun;
40     private Animation growMario;
41
42     private float stateTimer;
43     private boolean runningRight;
44     private boolean marioIsBig;
45     private boolean runGrowAnimation;
46     private boolean timeToDefineBigMario;
47     private boolean timeToRedefineMario;
48     private boolean marioIsDead;
49     private PlayScreen screen;
50
51     private Array<FireBall> fireballs;
52
53     public Hero(PlayScreen screen){
54         //initialize default values
55         this.screen = screen;
56         this.world = screen.getWorld();
57         currentState = State.STANDING;
58         previousState = State.STANDING;
59         stateTimer = 0;
60         runningRight = true;
61
62         Array<TextureRegion> frames = new Array<TextureRegion>();
63
64         //get run animation frames and add them to marioRun Animation
65         for(int i = 1; i < 4; i++)
66             frames.add(new TextureRegion(screen.getAtlas().findRegion("little_mario"), i * 16,
    ↪  0, 16, 16));
67         marioRun = new Animation(0.1f, frames);
68
69         frames.clear();
70
71         for(int i = 1; i < 4; i++)
72             frames.add(new TextureRegion(screen.getAtlas().findRegion("big_mario"), i * 16, 0,
    ↪  16, 32));
73         bigMarioRun = new Animation(0.1f, frames);
74
75         frames.clear();
76
77         //get set animation frames from growing mario
78         frames.add(new TextureRegion(screen.getAtlas().findRegion("big_mario"), 240, 0, 16,
    ↪ 32));
79         frames.add(new TextureRegion(screen.getAtlas().findRegion("big_mario"), 0, 0, 16, 32))
    ↪ ;
80         frames.add(new TextureRegion(screen.getAtlas().findRegion("big_mario"), 240, 0, 16,
    ↪ 32));
81         frames.add(new TextureRegion(screen.getAtlas().findRegion("big_mario"), 0, 0, 16, 32))
    ↪ ;
82         growMario = new Animation(0.2f, frames);
83
84
85         //get jump animation frames and add them to marioJump Animation
86         marioJump = new TextureRegion(screen.getAtlas().findRegion("little_mario"), 80, 0, 16,
    ↪  16);
```

```
87          bigMarioJump = new TextureRegion(screen.getAtlas().findRegion("big_mario"), 80, 0, 16,
        ↪  32);
88
89          //create texture region for mario standing
90          marioStand = new TextureRegion(screen.getAtlas().findRegion("little_mario"), 0, 0, 16,
        ↪  16);
91          bigMarioStand = new TextureRegion(screen.getAtlas().findRegion("big_mario"), 0, 0, 16,
        ↪  32);
92
93          //create dead mario texture region
94          marioDead = new TextureRegion(screen.getAtlas().findRegion("little_mario"), 96, 0, 16,
        ↪  16);
95
96          //define mario in Box2d
97          defineMario();
98
99          //set initial values for marios location, width and height. And initial frame as
        ↪ marioStand.
100         setBounds(0, 0, 16 / LDoS.PPM, 16 / LDoS.PPM);
101         setRegion(marioStand);
102
103         fireballs = new Array<FireBall>();
104
105     }
106
107     public void update(float dt){
108
109         // time is up : too late mario dies T_T
110         // the !isDead() method is used to prevent multiple invocation
111         // of "die␣music" and jumping
112         // there is probably better ways to do that but it works for now.
113         if (screen.getHud().isTimeUp() && !isDead()) {
114             die();
115         }
116
117         //update our sprite to correspond with the position of our Box2D body
118         if(marioIsBig)
119             setPosition(b2body.getPosition().x - getWidth() / 2, b2body.getPosition().y -
        ↪ getHeight() / 2 - 6 / LDoS.PPM);
120         else
121             setPosition(b2body.getPosition().x - getWidth() / 2, b2body.getPosition().y -
        ↪ getHeight() / 2);
122         //update sprite with the correct frame depending on marios current action
123         setRegion(getFrame(dt));
124         if(timeToDefineBigMario)
125             defineBigMario();
126         if(timeToRedefineMario)
127             redefineMario();
128
129         for(FireBall  ball : fireballs) {
130             ball.update(dt);
131             if(ball.isDestroyed())
132                 fireballs.removeValue(ball, true);
133         }
134
135     }
136
137     public TextureRegion getFrame(float dt){
138         //get marios current state. ie. jumping, running, standing...
139         currentState = getState();
140
141         TextureRegion region;
142
143         //depending on the state, get corresponding animation keyFrame.
144         switch(currentState){
145             case DEAD:
146                 region = marioDead;
147                 break;
148             case GROWING:
149                 region = growMario.getKeyFrame(stateTimer);
150                 if(growMario.isAnimationFinished(stateTimer)) {
151                     runGrowAnimation = false;
152                 }
153                 break;
154             case JUMPING:
155                 region = marioIsBig ? bigMarioJump : marioJump;
```

```
156                 break;
157             case RUNNING:
158                 region = marioIsBig ? bigMarioRun.getKeyFrame(stateTimer, true) : marioRun.
    ↪ getKeyFrame(stateTimer, true);
159                 break;
160             case FALLING:
161             case STANDING:
162             default:
163                 region = marioIsBig ? bigMarioStand : marioStand;
164                 break;
165         }
166 
167         //if mario is running left and the texture isnt facing left... flip it.
168         if((b2body.getLinearVelocity().x < 0 || !runningRight) && !region.isFlipX()){
169             region.flip(true, false);
170             runningRight = false;
171         }
172 
173         //if mario is running right and the texture isnt facing right... flip it.
174         else if((b2body.getLinearVelocity().x > 0 || runningRight) && region.isFlipX()){
175             region.flip(true, false);
176             runningRight = true;
177         }
178 
179         //if the current state is the same as the previous state increase the state timer.
180         //otherwise the state has changed and we need to reset timer.
181         stateTimer = currentState == previousState ? stateTimer + dt : 0;
182         //update previous state
183         previousState = currentState;
184         //return our final adjusted frame
185         return region;
186 
187     }
188 
189     public State getState(){
190         //Test to Box2D for velocity on the X and Y-Axis
191         //if mario is going positive in Y-Axis he is jumping... or if he just jumped and is
    ↪ falling remain in jump state
192         if(marioIsDead)
193             return State.DEAD;
194         else if(runGrowAnimation)
195             return State.GROWING;
196         else if((b2body.getLinearVelocity().y > 0 && currentState == State.JUMPING) || (b2body
    ↪ .getLinearVelocity().y < 0 && previousState == State.JUMPING))
197             return State.JUMPING;
198             //if negative in Y-Axis mario is falling
199         else if(b2body.getLinearVelocity().y < 0)
200             return State.FALLING;
201             //if mario is positive or negative in the X axis he is running
202         else if(b2body.getLinearVelocity().x != 0)
203             return State.RUNNING;
204             //if none of these return then he must be standing
205         else
206             return State.STANDING;
207     }
208 
209     public void grow(){
210         if( !isBig() ) {
211             runGrowAnimation = true;
212             marioIsBig = true;
213             timeToDefineBigMario = true;
214             setBounds(getX(), getY(), getWidth(), getHeight() * 2);
215             LDoS.manager.get("audio/sounds/powerup.wav", Sound.class).play();
216         }
217     }
218 
219     public void die() {
220 
221         if (!isDead()) {
222 
223             LDoS.manager.get("audio/music/mario_music.ogg", Music.class).stop();
224             LDoS.manager.get("audio/sounds/mariodie.wav", Sound.class).play();
225             marioIsDead = true;
226             Filter filter = new Filter();
227             filter.maskBits = LDoS.NOTHING_BIT;
228 
```

```
229             for (Fixture fixture : b2body.getFixtureList()) {
230                 fixture.setFilterData(filter);
231             }
232 
233             b2body.applyLinearImpulse(new Vector2(0, 4f), b2body.getWorldCenter(), true);
234         }
235     }
236 
237     public boolean isDead(){
238         return marioIsDead;
239     }
240 
241     public float getStateTimer(){
242         return stateTimer;
243     }
244 
245     public boolean isBig(){
246         return marioIsBig;
247     }
248 
249     public void jump(){
250         if ( currentState != State.JUMPING ) {
251             b2body.applyLinearImpulse(new Vector2(0, 4f), b2body.getWorldCenter(), true);
252             currentState = State.JUMPING;
253         }
254     }
255 
256     public void hit(Enemy enemy){
257         if(enemy instanceof Turtle && ((Turtle) enemy).currentState == Turtle.State.
    ↪ STANDING_SHELL)
258             ((Turtle) enemy).kick(enemy.b2body.getPosition().x > b2body.getPosition().x ?
    ↪ Turtle.KICK_RIGHT : Turtle.KICK_LEFT);
259         else {
260             if (marioIsBig) {
261                 marioIsBig = false;
262                 timeToRedefineMario = true;
263                 setBounds(getX(), getY(), getWidth(), getHeight() / 2);
264                 LDoS.manager.get("audio/sounds/powerdown.wav", Sound.class).play();
265             } else {
266                 die();
267             }
268         }
269     }
270 
271     public void redefineMario(){
272         Vector2 position = b2body.getPosition();
273         world.destroyBody(b2body);
274 
275         BodyDef bdef = new BodyDef();
276         bdef.position.set(position);
277         bdef.type = BodyDef.BodyType.DynamicBody;
278         b2body = world.createBody(bdef);
279 
280         FixtureDef fdef = new FixtureDef();
281         CircleShape shape = new CircleShape();
282         shape.setRadius(6 / LDoS.PPM);
283         fdef.filter.categoryBits = LDoS.MARIO_BIT;
284         fdef.filter.maskBits = LDoS.GROUND_BIT |
285                 LDoS.COIN_BIT |
286                 LDoS.BRICK_BIT |
287                 LDoS.ENEMY_BIT |
288                 LDoS.OBJECT_BIT |
289                 LDoS.ENEMY_HEAD_BIT |
290                 LDoS.ITEM_BIT;
291 
292         fdef.shape = shape;
293         b2body.createFixture(fdef).setUserData(this);
294 
295         EdgeShape head = new EdgeShape();
296         head.set(new Vector2(-2 / LDoS.PPM, 6 / LDoS.PPM), new Vector2(2 / LDoS.PPM, 6 / LDoS.
    ↪ PPM));
297         fdef.filter.categoryBits = LDoS.MARIO_HEAD_BIT;
298         fdef.shape = head;
299         fdef.isSensor = true;
300 
301         b2body.createFixture(fdef).setUserData(this);
```

```
302
303         timeToRedefineMario = false;
304
305     }
306
307     public void defineBigMario(){
308         Vector2 currentPosition = b2body.getPosition();
309         world.destroyBody(b2body);
310
311         BodyDef bdef = new BodyDef();
312         bdef.position.set(currentPosition.add(0, 10 / LDoS.PPM));
313         bdef.type = BodyDef.BodyType.DynamicBody;
314         b2body = world.createBody(bdef);
315
316         FixtureDef fdef = new FixtureDef();
317         CircleShape shape = new CircleShape();
318         shape.setRadius(6 / LDoS.PPM);
319         fdef.filter.categoryBits = LDoS.MARIO_BIT;
320         fdef.filter.maskBits = LDoS.GROUND_BIT |
321                 LDoS.COIN_BIT |
322                 LDoS.BRICK_BIT |
323                 LDoS.ENEMY_BIT |
324                 LDoS.OBJECT_BIT |
325                 LDoS.ENEMY_HEAD_BIT |
326                 LDoS.ITEM_BIT;
327
328         fdef.shape = shape;
329         b2body.createFixture(fdef).setUserData(this);
330         shape.setPosition(new Vector2(0, -14 / LDoS.PPM));
331         b2body.createFixture(fdef).setUserData(this);
332
333         EdgeShape head = new EdgeShape();
334         head.set(new Vector2(-2 / LDoS.PPM, 6 / LDoS.PPM), new Vector2(2 / LDoS.PPM, 6 / LDoS.
    ↪ PPM));
335         fdef.filter.categoryBits = LDoS.MARIO_HEAD_BIT;
336         fdef.shape = head;
337         fdef.isSensor = true;
338
339         b2body.createFixture(fdef).setUserData(this);
340         timeToDefineBigMario = false;
341     }
342
343     public void defineMario(){
344         BodyDef bdef = new BodyDef();
345         bdef.position.set(32 / LDoS.PPM, 32 / LDoS.PPM);
346         bdef.type = BodyDef.BodyType.DynamicBody;
347         b2body = world.createBody(bdef);
348
349         FixtureDef fdef = new FixtureDef();
350         CircleShape shape = new CircleShape();
351         shape.setRadius(6 / LDoS.PPM);
352         fdef.filter.categoryBits = LDoS.MARIO_BIT;
353         fdef.filter.maskBits = LDoS.GROUND_BIT |
354                 LDoS.COIN_BIT |
355                 LDoS.BRICK_BIT |
356                 LDoS.ENEMY_BIT |
357                 LDoS.OBJECT_BIT |
358                 LDoS.ENEMY_HEAD_BIT |
359                 LDoS.ITEM_BIT;
360
361         fdef.shape = shape;
362         b2body.createFixture(fdef).setUserData(this);
363
364         EdgeShape head = new EdgeShape();
365         head.set(new Vector2(-2 / LDoS.PPM, 6 / LDoS.PPM), new Vector2(2 / LDoS.PPM, 6 / LDoS.
    ↪ PPM));
366         fdef.filter.categoryBits = LDoS.MARIO_HEAD_BIT;
367         fdef.shape = head;
368         fdef.isSensor = true;
369
370         b2body.createFixture(fdef).setUserData(this);
371     }
372
373     public void fire(){
374         fireballs.add(new FireBall(screen, b2body.getPosition().x, b2body.getPosition().y,
    ↪ runningRight ? true : false));
```

```
375     }
376
377     public void draw(Batch batch){
378         super.draw(batch);
379         for(FireBall ball : fireballs)
380             ball.draw(batch);
381     }
382 }
```

```
1  package com.mygdx.game.tools;
2
3          import com.badlogic.gdx.maps.MapObject;
4          import com.badlogic.gdx.maps.objects.RectangleMapObject;
5          import com.badlogic.gdx.maps.tiled.TiledMap;
6          import com.badlogic.gdx.math.Rectangle;
7          import com.badlogic.gdx.physics.box2d.Body;
8          import com.badlogic.gdx.physics.box2d.BodyDef;
9          import com.badlogic.gdx.physics.box2d.FixtureDef;
10         import com.badlogic.gdx.physics.box2d.PolygonShape;
11         import com.badlogic.gdx.physics.box2d.World;
12         import com.badlogic.gdx.utils.Array;
13         import com.mygdx.game.LDoS;
14         import com.mygdx.game.screens.PlayScreen;
15         import com.mygdx.game.sprites.enemies.*;
16         import com.mygdx.game.sprites.tileObjects.*;
17
18
19
20 public class B2WorldCreator {
21     private Array<Goomba> goombas;
22     private Array<Turtle> turtles;
23
24     public B2WorldCreator(PlayScreen screen){
25         World world = screen.getWorld();
26         TiledMap map = screen.getMap();
27         //create body and fixture variables
28         BodyDef bdef = new BodyDef();
29         PolygonShape shape = new PolygonShape();
30         FixtureDef fdef = new FixtureDef();
31         Body body;
32
33         //create ground bodies/fixtures
34         for(MapObject object : map.getLayers().get(2).getObjects().getByType(
    ↪ RectangleMapObject.class)){
35             Rectangle rect = ((RectangleMapObject) object).getRectangle();
36
37             bdef.type = BodyDef.BodyType.StaticBody;
38             bdef.position.set((rect.getX() + rect.getWidth() / 2) / LDoS.PPM, (rect.getY() +
    ↪ rect.getHeight() / 2) / LDoS.PPM);
39
40             body = world.createBody(bdef);
41
42             shape.setAsBox(rect.getWidth() / 2 / LDoS.PPM, rect.getHeight() / 2 / LDoS.PPM);
43             fdef.shape = shape;
44             body.createFixture(fdef);
45         }
46
47         //create pipe bodies/fixtures
48         for(MapObject object : map.getLayers().get(3).getObjects().getByType(
    ↪ RectangleMapObject.class)){
49             Rectangle rect = ((RectangleMapObject) object).getRectangle();
50
51             bdef.type = BodyDef.BodyType.StaticBody;
52             bdef.position.set((rect.getX() + rect.getWidth() / 2) / LDoS.PPM, (rect.getY() +
    ↪ rect.getHeight() / 2) / LDoS.PPM);
53
54             body = world.createBody(bdef);
55
56             shape.setAsBox(rect.getWidth() / 2 / LDoS.PPM, rect.getHeight() / 2 / LDoS.PPM);
57             fdef.shape = shape;
58             fdef.filter.categoryBits = LDoS.OBJECT_BIT;
59             body.createFixture(fdef);
60         }
61
62         //create brick bodies/fixtures
63         for(MapObject object : map.getLayers().get(5).getObjects().getByType(
```

```
    ↪ RectangleMapObject.class)){
64              new Brick(screen, object);
65          }
66
67          //create death zone
68
69          for(MapObject object : map.getLayers().get(8).getObjects().getByType(
    ↪ RectangleMapObject.class)){
70              Rectangle rect = ((RectangleMapObject) object).getRectangle();
71
72              bdef.type = BodyDef.BodyType.StaticBody;
73              bdef.position.set((rect.getX() + rect.getWidth() / 2) / LDoS.PPM, (rect.getY() +
    ↪ rect.getHeight() / 2) / LDoS.PPM);
74
75              body = world.createBody(bdef);
76
77              shape.setAsBox(rect.getWidth() / 2 / LDoS.PPM, rect.getHeight() / 2 / LDoS.PPM);
78              fdef.shape = shape;
79              fdef.filter.categoryBits = LDoS.DEATH_BIT;
80              body.createFixture(fdef);
81          }
82          /** for(MapObject object : map.getLayers().get(8).getObjects().getByType(
    ↪ RectangleMapObject.class)){
83              new DeadZone(screen, object);
84          } **/
85
86          //create coin bodies/fixtures
87          for(MapObject object : map.getLayers().get(4).getObjects().getByType(
    ↪ RectangleMapObject.class)){
88
89              new Coin(screen, object);
90          }
91
92          //create all goombas
93          goombas = new Array<Goomba>();
94          for(MapObject object : map.getLayers().get(6).getObjects().getByType(
    ↪ RectangleMapObject.class)){
95              Rectangle rect = ((RectangleMapObject) object).getRectangle();
96              goombas.add(new Goomba(screen, rect.getX() / LDoS.PPM, rect.getY() / LDoS.PPM));
97          }
98          turtles = new Array<Turtle>();
99          for(MapObject object : map.getLayers().get(7).getObjects().getByType(
    ↪ RectangleMapObject.class)){
100             Rectangle rect = ((RectangleMapObject) object).getRectangle();
101             turtles.add(new Turtle(screen, rect.getX() / LDoS.PPM, rect.getY() / LDoS.PPM));
102         }
103     }
104
105
106     public Array<Goomba> getGoombas() {
107         return goombas;
108     }
109     public Array<Enemy> getEnemies(){
110         Array<Enemy> enemies = new Array<Enemy>();
111         enemies.addAll(goombas);
112         enemies.addAll(turtles);
113         return enemies;
114     }
115 }
```

```
1  package com.mygdx.game.tools;
2
3  import com.badlogic.gdx.Gdx;
4  import com.badlogic.gdx.physics.box2d.Contact;
5  import com.badlogic.gdx.physics.box2d.ContactImpulse;
6  import com.badlogic.gdx.physics.box2d.ContactListener;
7  import com.badlogic.gdx.physics.box2d.Fixture;
8  import com.badlogic.gdx.physics.box2d.Manifold;
9  import com.mygdx.game.LDoS;
10 import com.mygdx.game.sprites.enemies.Enemy;
11 import com.mygdx.game.sprites.items.Item;
12 import com.mygdx.game.sprites.Hero;
13 import com.mygdx.game.sprites.other.FireBall;
14 import com.mygdx.game.sprites.tileObjects.InteractiveTileObject;
15
16
```

```
17 public class WorldContactListener implements ContactListener {
18     @Override
19     public void beginContact(Contact contact) {
20         Fixture fixA = contact.getFixtureA();
21         Fixture fixB = contact.getFixtureB();
22 
23         int cDef = fixA.getFilterData().categoryBits | fixB.getFilterData().categoryBits;
24 
25         switch (cDef){
26             case LDoS.MARIO_HEAD_BIT | LDoS.BRICK_BIT:
27             case LDoS.MARIO_HEAD_BIT | LDoS.COIN_BIT:
28                 if(fixA.getFilterData().categoryBits == LDoS.MARIO_HEAD_BIT)
29                     ((InteractiveTileObject) fixB.getUserData()).onHeadHit((Hero) fixA.
    ↪ getUserData());
30                 else
31                     ((InteractiveTileObject) fixA.getUserData()).onHeadHit((Hero) fixB.
    ↪ getUserData());
32                 break;
33             case LDoS.ENEMY_HEAD_BIT | LDoS.MARIO_BIT:
34                 if(fixA.getFilterData().categoryBits == LDoS.ENEMY_HEAD_BIT)
35                     ((Enemy)fixA.getUserData()).hitOnHead((Hero) fixB.getUserData());
36                 else
37                     ((Enemy)fixB.getUserData()).hitOnHead((Hero) fixA.getUserData());
38                 break;
39             case LDoS.ENEMY_BIT | LDoS.OBJECT_BIT:
40                 if(fixA.getFilterData().categoryBits == LDoS.ENEMY_BIT)
41                     ((Enemy)fixA.getUserData()).reverseVelocity(true, false);
42                 else
43                     ((Enemy)fixB.getUserData()).reverseVelocity(true, false);
44                 break;
45             case LDoS.MARIO_BIT | LDoS.ENEMY_BIT:
46                 if(fixA.getFilterData().categoryBits == LDoS.MARIO_BIT)
47                     ((Hero) fixA.getUserData()).hit((Enemy)fixB.getUserData());
48                 else
49                     ((Hero) fixB.getUserData()).hit((Enemy)fixA.getUserData());
50                 break;
51             case LDoS.ENEMY_BIT | LDoS.ENEMY_BIT:
52                 ((Enemy)fixA.getUserData()).hitByEnemy((Enemy)fixB.getUserData());
53                 ((Enemy)fixB.getUserData()).hitByEnemy((Enemy)fixA.getUserData());
54                 break;
55             case LDoS.ITEM_BIT | LDoS.OBJECT_BIT:
56                 if(fixA.getFilterData().categoryBits == LDoS.ITEM_BIT)
57                     ((Item)fixA.getUserData()).reverseVelocity(true, false);
58                 else
59                     ((Item)fixB.getUserData()).reverseVelocity(true, false);
60                 break;
61             case LDoS.ITEM_BIT | LDoS.MARIO_BIT:
62                 if(fixA.getFilterData().categoryBits == LDoS.ITEM_BIT)
63                     ((Item)fixA.getUserData()).use((Hero) fixB.getUserData());
64                 else
65                     ((Item)fixB.getUserData()).use((Hero) fixA.getUserData());
66                 break;
67                 /**if(fixA.getFilterData().categoryBits == LDoS.MARIO_BIT)
68                     ((Hero)fixA.getUserData()).die();
69                 else
70                     ((Hero)fixB.getUserData()).die();
71                 break;**/
72 
73             case LDoS.FIREBALL_BIT | LDoS.OBJECT_BIT:
74                 if(fixA.getFilterData().categoryBits == LDoS.FIREBALL_BIT)
75                     ((FireBall)fixA.getUserData()).setToDestroy();
76                 else
77                     ((FireBall)fixB.getUserData()).setToDestroy();
78                 break;
79             case LDoS.MARIO_BIT | LDoS.DEATH_BIT:
80                 if(fixA.getFilterData().categoryBits == LDoS.MARIO_BIT)
81                     ((Hero)fixA.getUserData()).die();
82                 else
83                     ((Hero)fixB.getUserData()).die();
84                  break;
85         }
86     }
87 
88     @Override
89     public void endContact(Contact contact) {
90     }
```

```
91
92      @Override
93      public void preSolve(Contact contact, Manifold oldManifold) {
94      }
95
96      @Override
97      public void postSolve(Contact contact, ContactImpulse impulse) {
98
99      }
100 }
```

```
1  package com.mygdx.game;
2
3  import com.badlogic.gdx.Game;
4  import com.badlogic.gdx.assets.AssetManager;
5  import com.badlogic.gdx.audio.Music;
6  import com.badlogic.gdx.audio.Sound;
7  import com.badlogic.gdx.graphics.g2d.SpriteBatch;
8  import com.badlogic.gdx.physics.box2d.Filter;
9  import com.mygdx.game.screens.PlayScreen;
10
11 public class LDoS extends Game {
12   //Virtual Screen size and Box2D Scale(Pixels Per Meter)
13   public static final int V_WIDTH = 400;
14   public static final int V_HEIGHT = 208;
15   public static final float PPM = 100;
16
17   //Box2D Collision Bits
18   public static final short NOTHING_BIT = 0;
19   public static final short GROUND_BIT = 1;
20   public static final short MARIO_BIT = 2;
21   public static final short BRICK_BIT = 4;
22   public static final short COIN_BIT = 8;
23   public static final short DESTROYED_BIT = 16;
24   public static final short OBJECT_BIT = 32;
25   public static final short ENEMY_BIT = 64;
26   public static final short ENEMY_HEAD_BIT = 128;
27   public static final short ITEM_BIT = 256;
28   public static final short MARIO_HEAD_BIT = 512;
29   public static final short FIREBALL_BIT = 1024;
30   public static final short DEATH_BIT = 2048;
31
32   public SpriteBatch batch;
33
34   /* WARNING Using AssetManager in a static way can cause issues, especially on Android.
35   Instead you may want to pass around Assetmanager to those the classes that need it.
36   We will use it in the static context to save time for now. */
37   public static AssetManager manager;
38
39   @Override
40   public void create () {
41     batch = new SpriteBatch();
42     manager = new AssetManager();
43     manager.load("audio/music/mario_music.ogg", Music.class);
44     manager.load("audio/sounds/coin.wav", Sound.class);
45     manager.load("audio/sounds/bump.wav", Sound.class);
46     manager.load("audio/sounds/breakblock.wav", Sound.class);
47     manager.load("audio/sounds/powerup_spawn.wav", Sound.class);
48     manager.load("audio/sounds/powerup.wav", Sound.class);
49     manager.load("audio/sounds/powerdown.wav", Sound.class);
50     manager.load("audio/sounds/stomp.wav", Sound.class);
51     manager.load("audio/sounds/mariodie.wav", Sound.class);
52     manager.load("audio/sounds/50_cent-in_da_club.mp3", Sound.class);
53     manager.finishLoading();
54
55     setScreen(new PlayScreen(this));
56   }
57
58
59   @Override
60   public void dispose() {
61     super.dispose();
62     manager.dispose();
63     batch.dispose();
64   }
65
```

```
66    @Override
67    public void render () {
68       super.render();
69    }
70 }
```