ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

***«*САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»**

Институт компьютерных наук и технологий

**Отчет о прохождении учебной практики**

|  |
| --- |
| **Петров Виталий Дмитриевич** |

*(Ф.И.О. обучающегося)*

|  |
| --- |
| **2 курс, гр. 23531/4** |

*(номер курса обучения и учебной группы)*

|  |
| --- |
| **09.03.01 «Информатика и вычислительная техника »** |

*(Направление подготовки (код и наименование)*

|  |
| --- |
| **Место прохождения практики**: кафедра КСПТ ИКНТ ФГАОУ ВО «СПбПУ» |

*(указывается наименование профильной организации или наименование структурного подразделения*

|  |
| --- |
| **Сроки практики:** с 24 июня по 20 июля 2019 г. |

|  |
| --- |
| **Руководитель практики от ФГАОУ ВО «СПбПУ»:** |

|  |
| --- |
| Петров М.А., ассистент каф. КСПТ ИКНТ |

*(Ф.И.О., уч.степень, должность)*

|  |
| --- |
| **Оценка (зачет) : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |

|  |
| --- |
| Руководитель практики  от ФГАОУ ВО «СПбПУ»:: /Петров М.А./ |

|  |
| --- |
| Обучающийся: /Петров В.Д./ |

|  |
| --- |
| Дата: |

**Содержание**

1.Изучение 3

2. Индивидуальное задание 3

3. Введение 4

4. Основная часть 4

4.1. Описание классов программы 4

4.2. Примеры работы приложения 7

5. Заключение 9

6. Список использованных источников 9

7. Приложение 10

**Изучение**

В качестве учебной практики была выбрана разработка игры на языке Java, поэтому для изучения основ по этой теме нужно найти и изучить необходимую литературу, а также собрать все необходимые ресурсы. После этого следует применить полученные знания на практике.

**Индивидуальное задание**

В качестве индивидуального задания была выбрана разработка игры на языке Java. Данное приложение должно реализовывать следующие пункты:

* Поле размером 16x11 клеток, реализующее внутри себя переходы между различными частями карты, а также использующее физическую модель для игрока и его противников.
* Отрисовку поля, игрока и его противников на данной части карты.
* Переход между основной картой и подземельями, в конце которых находится какая-либо награда.
* Нанесение урона противникам, а также получение урона от них.
* Различное звуковое сопровождение для основной карты и для подземелий.
* Конечную цель, к которой стремится игрок и которая завершает игру.

**Введение**

На сегодняшний день в игровой индустрии среди крупных компаний практически исчезли проекты, использующие 2D-графику. Однако среди проектов больших компаний все чаще и чаще добиваются успеха игры от небольших групп энтузиастов, а также игры от одного человека, использующие преимущественно 2D или 2.5D-графику. Такие проекты чаще всего появляются на мобильных платформах, в особенности на платформе Android. Помимо возможности создавать свои игры на мобильные платформы, практика в создании 2D игр позволяет наиболее просто познакомиться с созданием игр в целом, в том числе и с использованием 3D-графики. На фоне всего выше изложенного навыки в использовании библиотек, облегчающих написание игр с 2D-графикой, имеют большое практическое применение.

Целью этой практики является изучение основ написания 2D-игр с использованием библиотеки LibGDX.

**Описание классов программы**

Далее представлено описание структуры приложения.

**AndroidLauncher.java** – класс для запуска приложения на платформе Android. Создает внутри себя объект класса MyGdxGame.

**DesktopLauncher.java** – класс для запуска приложения на персональных компьютерах. Создает внутри себя объект класса MyGdxGame.

**MyGdxGame.java** – главный класс приложения, реализует обязательные методы: create() – используется при запуске приложения, render() – используется для работы приложения.

**Controller.java** – класс, реализующий несколько видов управления героем:

с помощью клавиатуры при включении игры с компьютера, а также с помощью касаний при включении игры с мобильного устройства на платформе Android.

**WorldCreator.java** – класс, реализующий физическую модель для всех объектов на карте.

**State.java** – абстрактный класс, общий для всех экранов игры. Имеет абстрактные методы: handleInput() – для взаимодействия с игроком, update(float dt) – для обновления объектов экрана, render(SpriteBatch sb) – для отрисовки объектов экрана, dispose() – для удаления объектов экрана

**GameStateManager.java** – класс, реализующий очередь из экранов игры. Обновляет и рисует первый экран в очереди в классе MyGdxGame.java

**MenuState.java** – класс, реализующий начальный экран игры. После нажатия на экран или нажатия клавиши Enter экран удаляется, после чего в очередь GameStateManager записывается новый экземпляр класса PlayState.java

**PlayState.java** – класс, реализующий игровой экран. Проверяет нажатия игроком кнопок, после чего проигрывает музыку, обновляет все объекты на экране и отрисовывает их.

**GameOverState.java** – класс, реализующий экран конца игры. После нажатия на экран или нажатия клавиши Enter экран удаляется, после чего в очередь GameStateManager записывается новый экземпляр класса MenuState.java

**WinState.java** – класс, реализующий экран победы. Отрисовывает игрока, приз и поле, на котором был получен приз, после чего в очередь GameStateManager записывается новый экземпляр класса GameOverState.java

**Hud.java** – класс, реализующий отображение здоровья игрока и количество его денег на данный момент.

**DungeonManager.java** – класс, реализующий массив из объектов класса Level, содержащих ссылки на карты подземелья.

**LevelManager.java** – класс, реализующий массив из объектов класса Level, содержащих ссылки на части основной карты.

**Level.java** – класс, реализующий уровень игры. Содержит ссылку на созданный в программе Tiled .tmx файл, а также координаты своего местоположения на общей карте.

**MusicManager.java** – класс, реализующий загрузку музыки из памяти. После создания может быть использован для проигрывания загруженной музыки.

**TileType.java** – класс, содержащий методы и переменные для работы с одной клеткой поля, а также определяющий координаты следующего уровня.

**Animation.java** – класс, реализующий анимации для игры. После каждого обновления объектов из объекта анимации берется следующая текстура, которая и используется для отображения существа.

**Creature.java** – абстрактный класс, общий для всех существ. Содержит проигрывание анимации ходьбы, атаки, проверку атаки существ, а также координаты перехода между уровнями для игрока.

**CreatureAI.java**  - класс, реализующий искусственный интеллект противников. Каждые ½ секунды случайным образом выбирает направление движения, в котором будет двигаться существо.

**CreatureFactory.java** – класс, реализующий создание всех существ и игрока на поле, а также привязку текстур к их анимациям и удаление погибших существ с поля игры.

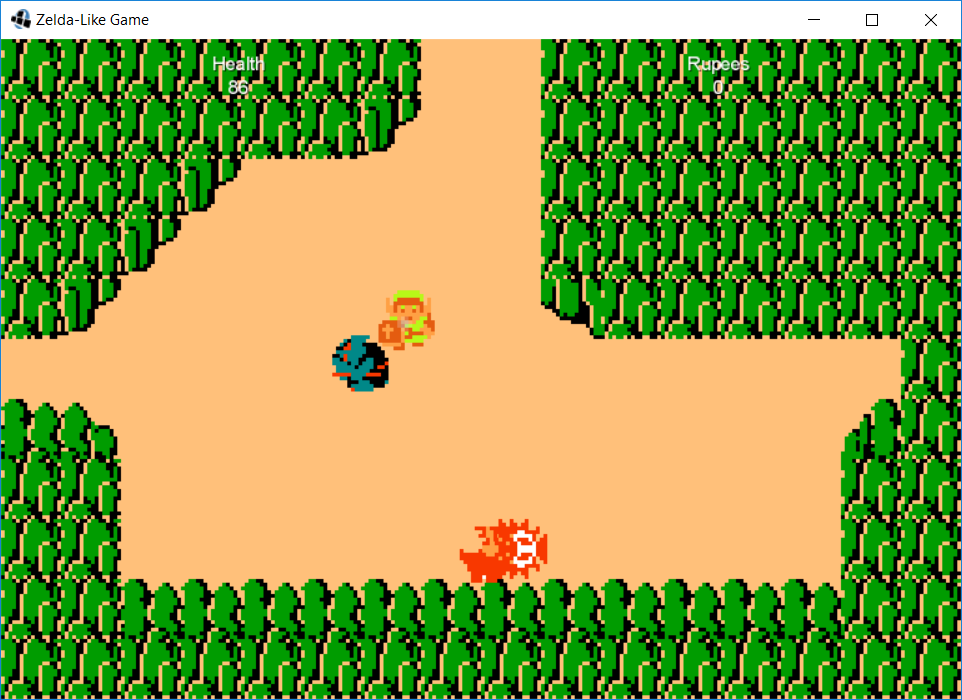
**Link.java** – класс, реализующий героя игры.

**Deku.java** – класс, реализующий один из видов противников.

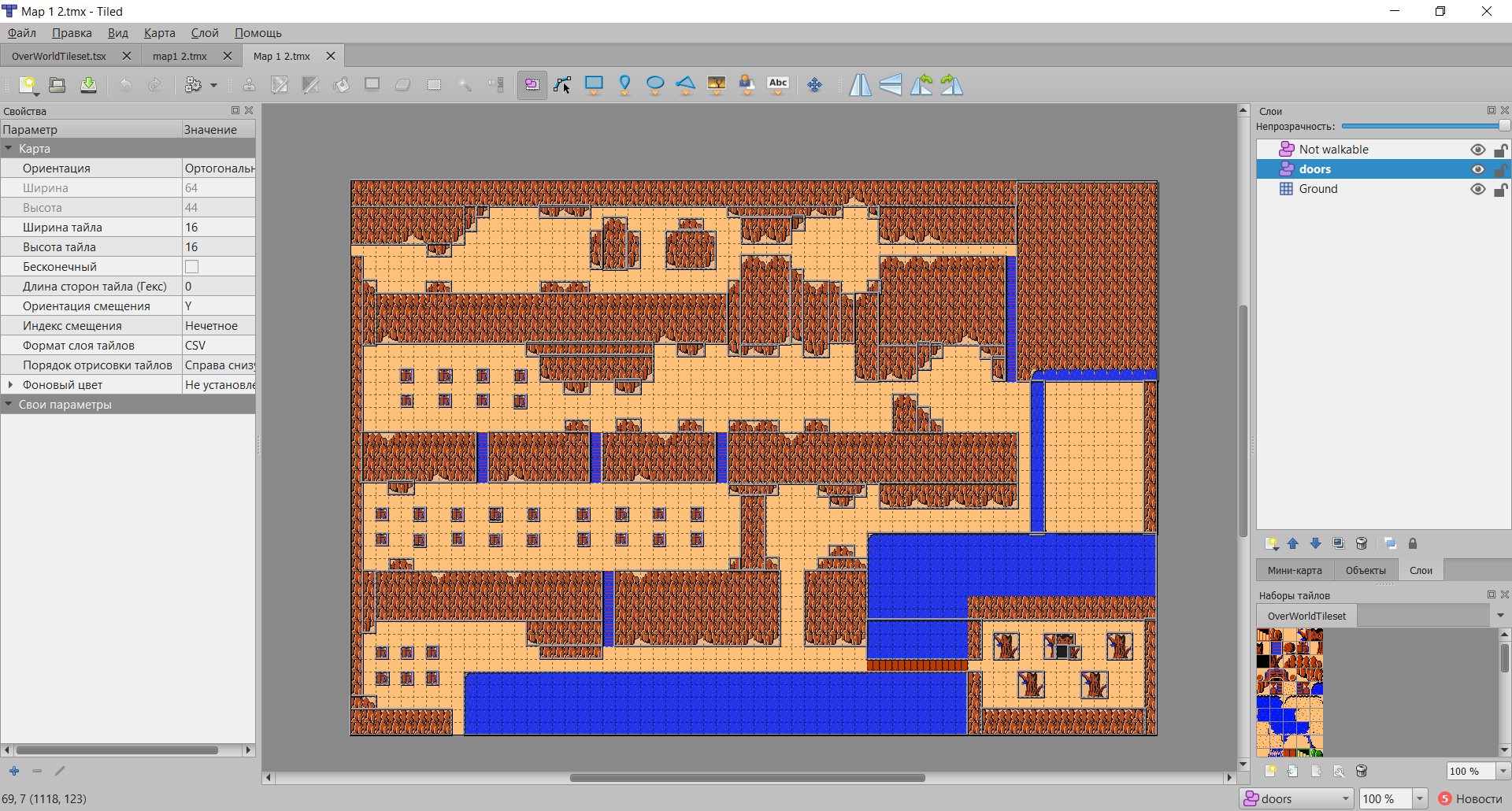
**Lynel.java** – класс, реализующий один из видов противников.

**Moblin.java** – класс, реализующий один из видов противников.

**Примеры работы программы**







**Заключение**

Данный материал был освоен для того, чтобы в дальнейшем уметь создавать игры с 2D-графикой для мобильных платформ, а также чтобы изучить разработку игр с использованием библиотеки LibGDX .Так как эта библиотека, как и многие другие используется как для 2D-игр, так и для 3D-графики, то данный опыт будет полезен и поможет в будущем.

**Ссылка на репозиторий проекта на github**

https://github.com/ADsty/Zelda-like

**Список использованных источников**

1. http:// libgdx.badlogicgames.com
2. developer.alexanderklimov.ru/android/games/libgdx/
3. www.fandroid.info/libgdx-urok-3-sozdaem-prostuyu-igru-delaem-android-igry/

**Приложение**

1. **AndroidLauncher.java**

**package** com.mygdx.game;  
  
**import** android.os.Bundle;  
  
**import** com.badlogic.gdx.backends.android.AndroidApplication;  
**import** com.badlogic.gdx.backends.android.AndroidApplicationConfiguration;  
**import** com.mygdx.game.MyGdxGame;  
  
**public class** AndroidLauncher **extends** AndroidApplication {  
 @Override  
 **protected void** onCreate (Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 AndroidApplicationConfiguration config = **new** AndroidApplicationConfiguration();  
 initialize(**new** MyGdxGame(), config);  
 }  
}

1. **DesktopLauncher.java**

**package** com.mygdx.game.desktop;  
  
**import** com.badlogic.gdx.backends.lwjgl.LwjglApplication;  
**import** com.badlogic.gdx.backends.lwjgl.LwjglApplicationConfiguration;  
**import** com.mygdx.game.MyGdxGame;  
  
**public class** DesktopLauncher {  
 **public static void** main(String[] arg) {  
 System.*setProperty*(**"user.name"**, **"Public"**);  
 LwjglApplicationConfiguration config = **new** LwjglApplicationConfiguration();  
 config.**title** = **"Zelda-Like Game"**;  
 config.**width** = MyGdxGame.***WIDTH***;  
 config.**height** = MyGdxGame.***HEIGHT***;  
 **new** LwjglApplication(**new** MyGdxGame(), config);  
 }  
}

1. **MyGdxGame.java**

**package** com.mygdx.game;  
  
**import** com.badlogic.gdx.ApplicationAdapter;  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.graphics.GL20;  
**import** com.badlogic.gdx.graphics.g2d.SpriteBatch;  
**import** com.mygdx.game.levels.LevelManager;  
**import** com.mygdx.game.levels.TileType;  
**import** com.mygdx.game.states.GameStateManager;  
**import** com.mygdx.game.states.MenuState;  
  
**public class** MyGdxGame **extends** ApplicationAdapter {  
 **public static final int *WIDTH*** = TileType.***TILE\_SIZE*** \* LevelManager.***MAP\_WIDTH*** \* 3;  
 **public static final int *HEIGHT*** = TileType.***TILE\_SIZE*** \* LevelManager.***MAP\_HEIGHT*** \* 3;  
 **private** GameStateManager **gsm**;  
 **public static** SpriteBatch *batch*;  
  
 */\*\*  
 \* Start method of LibGDX app lifecycle which creates all main game classes  
 \*/* @Override  
 **public void** create() {  
 *batch* = **new** SpriteBatch();  
 **gsm** = **new** GameStateManager();  
 Gdx.*gl*.glClearColor(0, 0, 0, 0);  
 **gsm**.push(**new** MenuState(**gsm**));  
 }  
  
 */\*\*  
 \* Main method of LibGDX app which renders all textures at the screen  
 \*/* @Override  
 **public void** render() {  
 Gdx.*gl*.glClear(GL20.***GL\_COLOR\_BUFFER\_BIT***);  
 **gsm**.update(Gdx.*graphics*.getDeltaTime());  
 **gsm**.render(*batch*);  
 }  
}

1. **Controller.java**

**package** com.mygdx.game;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.Input;  
**import** com.badlogic.gdx.graphics.Texture;  
**import** com.badlogic.gdx.scenes.scene2d.InputEvent;  
**import** com.badlogic.gdx.scenes.scene2d.InputListener;  
**import** com.badlogic.gdx.scenes.scene2d.Stage;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Image;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Table;  
  
**public class** Controller {  
 **private** Stage **stage**;  
 **private boolean upPressed**, **downPressed**, **leftPressed**, **rightPressed**;  
  
 */\*\*  
 \* Creates controller of game screen which takes input from player and return  
 \* boolean values of touched buttons  
 \*  
 \** ***@param stage*** *is needed stage of PlayState object  
 \*/* **public** Controller(Stage stage) {  
 **this**.**stage** = stage;  
  
 **this**.**stage**.addListener(**new** InputListener() {  
  
 @Override  
 **public boolean** keyDown(InputEvent event, **int** keycode) {  
 **switch** (keycode) {  
 **case** Input.Keys.***UP***:  
 **upPressed** = **true**;  
 **break**;  
 **case** Input.Keys.***DOWN***:  
 **downPressed** = **true**;  
 **break**;  
 **case** Input.Keys.***LEFT***:  
 **leftPressed** = **true**;  
 **break**;  
 **case** Input.Keys.***RIGHT***:  
 **rightPressed** = **true**;  
 **break**;  
 }  
 **return true**;  
 }  
  
 @Override  
 **public boolean** keyUp(InputEvent event, **int** keycode) {  
 **switch** (keycode) {  
 **case** Input.Keys.***UP***:  
 **upPressed** = **false**;  
 **break**;  
 **case** Input.Keys.***DOWN***:  
 **downPressed** = **false**;  
 **break**;  
 **case** Input.Keys.***LEFT***:  
 **leftPressed** = **false**;  
 **break**;  
 **case** Input.Keys.***RIGHT***:  
 **rightPressed** = **false**;  
 **break**;  
 }  
 **return true**;  
 }  
 });  
  
 Gdx.*input*.setInputProcessor(stage);  
  
 Table table = **new** Table();  
 table.left().bottom();  
  
 Image upImg = **new** Image(**new** Texture(**"buttons/flatDark25.png"**));  
 upImg.setSize(15, 15);  
 upImg.addListener(**new** InputListener() {  
  
 @Override  
 **public boolean** touchDown(InputEvent event, **float** x, **float** y, **int** pointer, **int** button) {  
 **upPressed** = **true**;  
 **return true**;  
 }  
  
 @Override  
 **public void** touchUp(InputEvent event, **float** x, **float** y, **int** pointer, **int** button) {  
 **upPressed** = **false**;  
 }  
 });  
  
 Image downImg = **new** Image(**new** Texture(**"buttons/flatDark26.png"**));  
 downImg.setSize(15, 15);  
 downImg.addListener(**new** InputListener() {  
  
 @Override  
 **public boolean** touchDown(InputEvent event, **float** x, **float** y, **int** pointer, **int** button) {  
 **downPressed** = **true**;  
 **return true**;  
 }  
  
 @Override  
 **public void** touchUp(InputEvent event, **float** x, **float** y, **int** pointer, **int** button) {  
 **downPressed** = **false**;  
 }  
  
 });  
  
 Image rightImg = **new** Image(**new** Texture(**"buttons/flatDark24.png"**));  
 rightImg.setSize(15, 15);  
 rightImg.addListener(**new** InputListener() {  
  
 @Override  
 **public boolean** touchDown(InputEvent event, **float** x, **float** y, **int** pointer, **int** button) {  
 **rightPressed** = **true**;  
 **return true**;  
 }  
  
 @Override  
 **public void** touchUp(InputEvent event, **float** x, **float** y, **int** pointer, **int** button) {  
 **rightPressed** = **false**;  
 }  
  
 });  
  
 Image leftImg = **new** Image(**new** Texture(**"buttons/flatDark23.png"**));  
 leftImg.setSize(15, 15);  
 leftImg.addListener(**new** InputListener() {  
  
 @Override  
 **public boolean** touchDown(InputEvent event, **float** x, **float** y, **int** pointer, **int** button) {  
 **leftPressed** = **true**;  
 **return true**;  
 }  
  
 @Override  
 **public void** touchUp(InputEvent event, **float** x, **float** y, **int** pointer, **int** button) {  
 **leftPressed** = **false**;  
 }  
  
 });  
  
 table.add();  
 table.add(upImg).size(upImg.getWidth(), upImg.getHeight());  
 table.add();  
 table.row().pad(5, 5, 5, 5);  
 table.add(leftImg).size(leftImg.getWidth(), leftImg.getHeight());  
 table.add();  
 table.add(rightImg).size(rightImg.getWidth(), rightImg.getHeight());  
 table.row().padBottom(5);  
 table.add();  
 table.add(downImg).size(downImg.getWidth(), downImg.getHeight());  
 stage.addActor(table);  
 }  
  
 */\*\*  
 \* Method draws controller buttons (works only at Android platform)  
 \*/* **public void** draw() {  
 **stage**.draw();  
 }  
  
 **public boolean** isUpPressed() {  
 **return upPressed**;  
 }  
  
 **public boolean** isDownPressed() {  
 **return downPressed**;  
 }  
  
 **public boolean** isLeftPressed() {  
 **return leftPressed**;  
 }  
  
 **public boolean** isRightPressed() {  
 **return rightPressed**;  
 }  
  
 **public boolean** noKeyPressed() {  
 **return** !**upPressed** && !**downPressed** && !**rightPressed** && !**leftPressed**;  
 }  
  
}

1. **WorldCreator.java**

**package** com.mygdx.game.world;  
  
**import** com.badlogic.gdx.maps.MapObject;  
**import** com.badlogic.gdx.maps.objects.RectangleMapObject;  
**import** com.badlogic.gdx.maps.tiled.TiledMap;  
**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.FixtureDef;  
**import** com.badlogic.gdx.physics.box2d.PolygonShape;  
**import** com.badlogic.gdx.physics.box2d.World;  
**import** com.mygdx.game.states.PlayState;  
  
**public class** WorldCreator {  
  
 */\*\*  
 \* Creates all physics object of box2d model  
 \*  
 \** ***@param state*** *is current object of PlayState  
 \*/* **public** WorldCreator(PlayState state) {  
 World world = state.getWorld();  
 TiledMap map = state.getMap();  
 BodyDef bodyDef = **new** BodyDef();  
 PolygonShape shape = **new** PolygonShape();  
 FixtureDef fixtureDef = **new** FixtureDef();  
 Body body;  
  
 **for** (MapObject object : map.getLayers().get(2).getObjects().getByType(RectangleMapObject.**class**)) {  
 Rectangle rect = ((RectangleMapObject) object).getRectangle();  
 bodyDef.**position**.set((rect.getX() + rect.getWidth() / 2), (rect.getY() + rect.getHeight() / 2));  
 shape.setAsBox(rect.getWidth() / 2, rect.getHeight() / 2);  
 bodyDef.**type** = BodyDef.BodyType.***StaticBody***;  
 body = world.createBody(bodyDef);  
 fixtureDef.**shape** = shape;  
 body.createFixture(fixtureDef);  
 }  
 }  
  
 */\*\*  
 \* Method checks if creature body is in other physics object of game field  
 \*  
 \** ***@param state*** *is current object of PlayState  
 \** ***@param rectangle*** *is creatures body  
 \** ***@return*** *true if creature body is in other physics object  
 \*/* **public boolean** checkPhysicsBoxes(PlayState state, Rectangle rectangle) {  
 TiledMap map = state.getMap();  
 **for** (MapObject object : map.getLayers().get(2).getObjects().getByType(RectangleMapObject.**class**)) {  
 Rectangle rect = ((RectangleMapObject) object).getRectangle();  
 **if** (rect.overlaps(rectangle)) {  
 **return true**;  
 }  
 }  
 **return false**;  
 }  
}

1. **State.java**

**package** com.mygdx.game.states;  
  
**import** com.badlogic.gdx.graphics.OrthographicCamera;  
**import** com.badlogic.gdx.graphics.g2d.SpriteBatch;  
  
**public abstract class** State {  
  
 OrthographicCamera **camera**;  
 GameStateManager **gsm**;  
  
 State(GameStateManager gsm) {  
 **this**.**gsm** = gsm;  
 **camera** = **new** OrthographicCamera();  
 }  
  
 **protected abstract void** handleInput();  
  
 **public abstract void** update(**float** dt);  
  
 **public abstract void** render(SpriteBatch sb);  
  
 **public abstract void** dispose();  
}

1. **GameStateManager.java**

**package** com.mygdx.game.states;  
  
**import** com.badlogic.gdx.graphics.g2d.SpriteBatch;  
  
**import** java.util.Stack;  
  
**public class** GameStateManager {  
  
 **private** Stack<State> **states**;  
  
 */\*\*  
 \* Just an ordinary stack which uses update render and dispose methods of states  
 \*/* **public** GameStateManager() {  
 **states** = **new** Stack<State>();  
 }  
  
 **public void** push(State state) {  
 **states**.push(state);  
 }  
  
 **public void** pop() {  
 **states**.pop().dispose();  
 }  
  
 **void** set(State state) {  
 **states**.pop().dispose();  
 **states**.push(state);  
 }  
  
 **public void** update(**float** dt) {  
 **states**.peek().update(dt);  
 }  
  
 **public void** render(SpriteBatch sb) {  
 **states**.peek().render(sb);  
 }  
}

1. **MenuState.java**

**package** com.mygdx.game.states;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.Input;  
**import** com.badlogic.gdx.graphics.Texture;  
**import** com.badlogic.gdx.graphics.g2d.SpriteBatch;  
**import** com.mygdx.game.MyGdxGame;  
  
**public class** MenuState **extends** State {  
  
 **private** Texture **background**;  
  
 */\*\*  
 \* It is start screen of game which checks for some taps on screen and then changes to game screen  
 \*/* **public** MenuState(GameStateManager gsm) {  
 **super**(gsm);  
 **background** = **new** Texture(**"Zelda-Menu.jpg"**);  
 }  
  
 @Override  
 **public void** handleInput() {  
 **if** (Gdx.*input*.justTouched()) {  
 **gsm**.set(**new** PlayState(**gsm**, 3, 3, 100, 100, 100, **false**, 0, 0));  
 }  
 **if** (Gdx.*input*.isKeyPressed(Input.Keys.***ENTER***)) {  
 **gsm**.set(**new** PlayState(**gsm**, 3, 3, 100, 100, 100, **false**, 0, 0));  
 }  
 }  
  
 @Override  
 **public void** update(**float** dt) {  
 handleInput();  
 }  
  
 @Override  
 **public void** render(SpriteBatch sb) {  
 sb.begin();  
 sb.draw(**background**, 0, 0, MyGdxGame.***WIDTH***, MyGdxGame.***HEIGHT***);  
 sb.end();  
 }  
  
 @Override  
 **public void** dispose() {  
 **background**.dispose();  
 System.***out***.println(**"Menu disposed"**);  
 }  
}

1. **PlayState.java**

**package** com.mygdx.game.states;  
  
**import** com.badlogic.gdx.Application;  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.Input;  
**import** com.badlogic.gdx.audio.Music;  
**import** com.badlogic.gdx.graphics.Texture;  
**import** com.badlogic.gdx.graphics.g2d.SpriteBatch;  
**import** com.badlogic.gdx.maps.tiled.TiledMap;  
**import** com.badlogic.gdx.math.Rectangle;  
**import** com.badlogic.gdx.math.Vector2;  
**import** com.badlogic.gdx.physics.box2d.World;  
**import** com.badlogic.gdx.scenes.scene2d.Stage;  
**import** com.badlogic.gdx.utils.viewport.FitViewport;  
**import** com.badlogic.gdx.utils.viewport.Viewport;  
**import** com.mygdx.game.Controller;  
**import** com.mygdx.game.MyGdxGame;  
**import** com.mygdx.game.creatures.Creature;  
**import** com.mygdx.game.creatures.CreatureFactory;  
**import** com.mygdx.game.creatures.Deku;  
**import** com.mygdx.game.creatures.Link;  
**import** com.mygdx.game.levels.LevelManager;  
**import** com.mygdx.game.levels.MusicManager;  
**import** com.mygdx.game.levels.TileType;  
**import** com.mygdx.game.world.WorldCreator;  
  
**import** java.util.Random;  
  
  
**public class** PlayState **extends** State {  
 **private** Controller **controller**;  
 **private** Stage **stage**;  
 **private** Link **link**;  
 **private** CreatureFactory **factory**;  
 **private** TileType **tileType**;  
 **private** LevelManager **manager**;  
 **private** World **world**;  
 **private** WorldCreator **worldCreator**;  
 **private** Hud **hud**;  
 **private** MusicManager **musicManager**;  
 **private int levelX**;  
 **private int levelY**;  
 **private int dungeonX**;  
 **private int dungeonY**;  
 **private boolean inDungeon**;  
 **private float linkX** = 300;  
 **private float linkY** = 200;  
 **private int linkHealth**;  
  
 */\*\*  
 \* Creates new game field with all needed objects  
 \*  
 \** ***@param gsm*** *is GameStateManager object which controls game  
 \** ***@param levelX*** *is current level X coordinate  
 \** ***@param levelY*** *is current level Y coordinate  
 \** ***@param linkHealth*** *is current link health  
 \** ***@param linkX*** *is current link position X coordinate  
 \** ***@param linkY*** *is current link position Y coordinate  
 \** ***@param inDungeon*** *will be true if the player is in the dungeon right now  
 \** ***@param dungeonX*** *current dungeon X coordinate  
 \** ***@param dungeonY*** *current dungeon Y coordinate  
 \*/* PlayState(GameStateManager gsm, **int** levelX, **int** levelY, **int** linkHealth, **float** linkX, **float** linkY, **boolean** inDungeon, **int** dungeonX, **int** dungeonY) {  
 **super**(gsm);  
 Music music;  
 **this**.**levelX** = levelX;  
 **this**.**levelY** = levelY;  
 **this**.**linkHealth** = linkHealth;  
 **this**.**dungeonX** = dungeonX;  
 **this**.**dungeonY** = dungeonY;  
 **this**.**inDungeon** = inDungeon;  
 **musicManager** = **new** MusicManager();  
 **stage** = createStage();  
 **controller** = **new** Controller(**stage**);  
 **factory** = **new** CreatureFactory();  
 **camera**.setToOrtho(**false**, 256, 176);  
 **tileType** = **new** TileType();  
 **manager** = **new** LevelManager(**camera**, levelX, levelY, **this**.**inDungeon**, dungeonX, dungeonY);  
 **world** = **new** World(**new** Vector2(0, 0), **true**);  
 **worldCreator** = **new** WorldCreator(**this**);  
 **link** = **factory**.createPlayer(**this**, linkX, linkY);  
 Random random = **new** Random();  
 **for** (**int** i = 1; i < random.nextInt(3) + 4; i++) {  
 **factory**.createEnemie(**this**);  
 }  
 **hud** = **new** Hud(MyGdxGame.*batch*, **this**);  
 **if** (**manager**.inDungeon()) {  
 music = **musicManager**.getManager().get(**"audio/music/Legend of Zelda - NES - Dungeon Theme.mp3"**, Music.**class**);  
 } **else** {  
 music = **musicManager**.getManager().get(**"audio/music/The Legend of Zelda - NES - Overworld.mp3"**, Music.**class**);  
 }  
 music.setLooping(**true**);  
 music.setVolume(0.1f);  
 music.play();  
 }  
  
 **public** WorldCreator getWorldCreator() {  
 **return worldCreator**;  
 }  
  
 */\*\*  
 \* Method creates new stage for controller  
 \*/* **private** Stage createStage() {  
 Viewport viewport = **new** FitViewport(MyGdxGame.***WIDTH*** - 100, MyGdxGame.***HEIGHT*** - 100, **camera**);  
 **return new** Stage(viewport, MyGdxGame.*batch*);  
 }  
  
 */\*\*  
 \* Method handles input that player gives  
 \*/* @Override  
 **protected void** handleInput() {  
 **try** {  
 **if** (**controller**.isLeftPressed()) {  
 **link**.goLeft(Gdx.*graphics*.getDeltaTime());  
 } **else if** (**controller**.isRightPressed()) {  
 **link**.goRight(Gdx.*graphics*.getDeltaTime());  
 } **else if** (**controller**.isUpPressed()) {  
 **link**.goUp(Gdx.*graphics*.getDeltaTime());  
 } **else if** (**controller**.isDownPressed()) {  
 **link**.goDown(Gdx.*graphics*.getDeltaTime());  
 }  
 **if** (**controller**.noKeyPressed()) {  
 **link**.setStayPosition();  
 }  
 **if** ((Gdx.*input*.isTouched() && **controller**.noKeyPressed()) || Gdx.*input*.isKeyPressed(Input.Keys.***Z***)) {  
 **link**.attack();  
 **for** (Creature creature : **factory**.getEnemies()) {  
 **link**.makeDamage(**link**.getDamage(), creature, Gdx.*graphics*.getDeltaTime());  
 }  
 }  
 } **catch** (NullPointerException e) {  
 System.***out***.println(e.getMessage());  
 }  
 }  
  
 */\*\*  
 \* Method updates all objects on the game field  
 \*  
 \** ***@param dt*** *is delta time between frames  
 \*/* @Override  
 **public void** update(**float** dt) {  
 **if** (isWon()) {  
 **gsm**.set(**new** WinState(**gsm**));  
 }  
 handleInput();  
 **world**.step(1 / 60f, 6, 2);  
 **for** (Creature creature : **factory**.getEnemies()) {  
 **if** (creature.isDead()) {  
 creature.playDeathSound();  
 **world**.destroyBody(creature.getBody());  
 **factory**.delete(creature);  
 }  
 ((Deku) creature).update(dt);  
 creature.makeDamage(creature.getDamage(), **link**, dt);  
 }  
 **link**.update();  
 **linkHealth** = **link**.getHealth();  
 **hud**.update();  
 }  
  
 */\*\*  
 \* Method renders all needed objects on game field  
 \*  
 \** ***@param sb*** *is main class SpriteBatch object  
 \*/* @Override  
 **public void** render(SpriteBatch sb) {  
 **if** (**link**.isDead()) {  
 **gsm**.set(**new** GameOverState(**gsm**));  
 }  
 **if** (**tileType**.checkLevelEdge(**link**, **manager**.getTiledMap())) {  
 **int**[] array = **tileType**.getEdgeDirection(**link**);  
 **manager**.changeLevel(array[0], array[1]);  
 **link**.setNewPosition();  
 **if** (**inDungeon**) {  
 **dungeonX** += array[0];  
 **dungeonY** += array[1];  
 } **else** {  
 **levelX** += array[0];  
 **levelY** += array[1];  
 }  
 System.***out***.println(array[0]);  
 System.***out***.println(array[1]);  
 **linkX** = **link**.getCreatureRectangle().**x**;  
 **linkY** = **link**.getCreatureRectangle().**y**;  
 **gsm**.set(**new** PlayState(**gsm**, **levelX**, **levelY**, **linkHealth**, **linkX**, **linkY**, **inDungeon**, **dungeonX**, **dungeonY**));  
 } **else if** (**tileType**.isDoor(getMap(), **link**)) {  
 **if** (**manager**.inDungeon()) {  
 **manager**.exitFromDungeon();  
 **link**.setNewPosition();  
 **inDungeon** = **false**;  
 } **else** {  
 **manager**.goInDungeon();  
 **link**.setNewPosition();  
 **inDungeon** = **true**;  
 }  
 **gsm**.set(**new** PlayState(**gsm**, **levelX**, **levelY**, **linkHealth**, **linkX**, **linkY**, **inDungeon**, **dungeonX**, **dungeonY**));  
 System.***out***.println(**inDungeon**);  
 System.***out***.println(**levelX**);  
 System.***out***.println(**levelY**);  
 }  
 **manager**.drawCreatedLevel();  
 sb.setProjectionMatrix(**camera**.**combined**);  
 sb.begin();  
 **if** (**dungeonY** == 1 && **dungeonX** == 0) {  
 sb.draw(**new** Texture(**"items/chestClose.png"**), (**float**) 256 / 2, (**float**) 176 / 2 - 6);  
 }  
 sb.draw(**link**.getLink(), **link**.getPositionX() - (**float**) TileType.***TILE\_SIZE*** / 2, **link**.getPositionY() - (**float**) TileType.***TILE\_SIZE*** / 2);  
 **for** (Creature creature : **factory**.getEnemies()) {  
 **if** (!creature.isDead()) {  
 sb.draw(creature.getCreature(), creature.getPositionX() - (**float**) TileType.***TILE\_SIZE*** / 2, creature.getPositionY() - (**float**) TileType.***TILE\_SIZE*** / 2);  
 }  
 }  
 sb.end();  
 **if** (Gdx.*app*.getType() == Application.ApplicationType.***Android***) {  
 **controller**.draw();  
 }  
 **hud**.**stage**.draw();  
 }  
  
 **public** TiledMap getMap() {  
 **return manager**.getTiledMap();  
 }  
  
 **public** World getWorld() {  
 **return world**;  
 }  
  
 Link getLink() {  
 **return link**;  
 }  
  
 **public** LevelManager getManager() {  
 **return manager**;  
 }  
  
 **public int** getLinkHealth() {  
 **return linkHealth**;  
 }  
  
 **public** MusicManager getMusicManager() {  
 **return musicManager**;  
 }  
  
 */\*\*  
 \* Methods checks if the player is near of endpoint  
 \*  
 \** ***@return*** *true if the player is near of endpoint  
 \*/* **private boolean** isWon() {  
 Rectangle rectangle = **new** Rectangle();  
 rectangle.**x** = (**float**) 256 / 2;  
 rectangle.**y** = (**float**) 176 / 2 - 6;  
 rectangle.**width** = TileType.***TILE\_SIZE***;  
 rectangle.**height** = TileType.***TILE\_SIZE***;  
 **return dungeonY** == 1 && **dungeonX** == 0 && rectangle.overlaps(**link**.getCreatureRectangle());  
 }  
  
 */\*\*  
 \* Method disposes all objects of game field  
 \*/* @Override  
 **public void** dispose() {  
 **link**.dispose();  
 **manager**.dispose();  
 **factory**.dispose();  
 **hud**.dispose();  
 **musicManager**.dispose();  
 **stage**.dispose();  
 System.***out***.println(**"Play state disposed"**);  
 }  
}

1. **Hud.java**

**package** com.mygdx.game.states;  
  
**import** com.badlogic.gdx.utils.Disposable;  
**import** com.badlogic.gdx.graphics.Color;  
**import** com.badlogic.gdx.graphics.OrthographicCamera;  
**import** com.badlogic.gdx.graphics.g2d.BitmapFont;  
**import** com.badlogic.gdx.graphics.g2d.SpriteBatch;  
**import** com.badlogic.gdx.scenes.scene2d.Stage;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Label;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Table;  
**import** com.badlogic.gdx.utils.viewport.FitViewport;  
**import** com.badlogic.gdx.utils.viewport.Viewport;  
**import** com.mygdx.game.MyGdxGame;  
  
**public class** Hud **implements** Disposable {  
 Stage **stage**;  
 **private** Label **health**;  
 **private** Label **rupees**;  
 **private** PlayState **state**;  
  
 */\*\*  
 \* It is hud for game screen which displays player's health and money  
 \*/* Hud(SpriteBatch sb, PlayState state) {  
 **this**.**state** = state;  
 Viewport viewport = **new** FitViewport(MyGdxGame.***WIDTH***, MyGdxGame.***HEIGHT***, **new** OrthographicCamera());  
 **stage** = **new** Stage(viewport, sb);  
 Table table = **new** Table();  
 table.top();  
 table.setFillParent(**true**);  
 Label healthLabel = **new** Label(**"Health"**, **new** Label.LabelStyle(**new** BitmapFont(), Color.***WHITE***));  
 Label rupeesLabel = **new** Label(**"Rupees"**, **new** Label.LabelStyle(**new** BitmapFont(), Color.***WHITE***));  
 **health** = **new** Label(String.*format*(**"%02d"**, state.getLink().getHealth()), **new** Label.LabelStyle(**new** BitmapFont(), Color.***WHITE***));  
 **rupees** = **new** Label(String.*format*(**"%03d"**, state.getLink().getRupees()), **new** Label.LabelStyle(**new** BitmapFont(), Color.***WHITE***));  
 table.add(healthLabel).expandX().padTop(10);  
 table.add(rupeesLabel).expandX().padTop(10);  
 table.row();  
 table.add(**health**).expandX();  
 table.add(**rupees**).expandX();  
 **stage**.addActor(table);  
  
 }  
  
 */\*\*  
 \* Method updates hud values of health and money of player  
 \*/* **void** update() {  
 **health**.setText(**state**.getLink().getHealth());  
 **rupees**.setText(**state**.getLink().getRupees());  
 }  
  
 */\*\*  
 \* Method disposes stage used by hud  
 \*/* @Override  
 **public void** dispose() {  
 **stage**.dispose();  
 }  
  
}

1. **WinState.java**

**package** com.mygdx.game.states;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.audio.Sound;  
**import** com.badlogic.gdx.graphics.Texture;  
**import** com.badlogic.gdx.graphics.g2d.SpriteBatch;  
**import** com.badlogic.gdx.maps.tiled.TiledMap;  
**import** com.badlogic.gdx.maps.tiled.TmxMapLoader;  
**import** com.badlogic.gdx.maps.tiled.renderers.OrthogonalTiledMapRenderer;  
**import** com.mygdx.game.levels.MusicManager;  
**import** com.mygdx.game.levels.TileType;  
  
**public class** WinState **extends** State {  
 **private** OrthogonalTiledMapRenderer **tiledMapRenderer**;  
 **private float time**;  
  
 */\*\*  
 \* It is the finish point of game which renders win screen and play some music  
 \* after which it will be changed to black screen  
 \*/* WinState(GameStateManager gsm) {  
 **super**(gsm);  
 **camera**.setToOrtho(**false**, 256, 176);  
 TiledMap tiledMap = **new** TmxMapLoader().load(**"maps/dungeon2 1.tmx"**);  
 **tiledMapRenderer** = **new** OrthogonalTiledMapRenderer(tiledMap);  
 MusicManager musicManager = **new** MusicManager();  
 Sound music = musicManager.getManager().get(**"audio/sounds/Sound Effect (20).wav"**);  
 music.play(0.1f);  
 }  
  
 @Override  
 **protected void** handleInput() {  
 **if** (Gdx.*input*.justTouched()) {  
 **gsm**.set(**new** GameOverState(**gsm**));  
 }  
 **time** += Gdx.*graphics*.getDeltaTime();  
 **if** (**time** >= 2f) {  
 **gsm**.set(**new** GameOverState(**gsm**));  
 }  
 }  
  
 @Override  
 **public void** update(**float** dt) {  
 handleInput();  
 }  
  
 @Override  
 **public void** render(SpriteBatch sb) {  
 **tiledMapRenderer**.setView(**camera**);  
 **tiledMapRenderer**.render();  
 Texture chestOpen = **new** Texture(**"items/chestOpen.png"**);  
 Texture linkWin = **new** Texture(**"link images/linkGetItem.png"**);  
 Texture winItem = **new** Texture(**"items/rupee.png"**);  
 sb.setProjectionMatrix(**camera**.**combined**);  
 sb.begin();  
 sb.draw(chestOpen, (**float**) 256 / 2, (**float**) 176 / 2 - 6);  
 sb.draw(linkWin, (**float**) 256 / 2, (**float**) 176 / 2 - 6);  
 sb.draw(winItem, (**float**) 256 / 2 + 4, (**float**) 176 / 2 + TileType.***TILE\_SIZE*** - 6);  
 sb.end();  
 }  
  
 @Override  
 **public void** dispose() {  
 **tiledMapRenderer**.dispose();  
 }  
}

1. **GameOverState.java**

**package** com.mygdx.game.states;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.graphics.Texture;  
**import** com.badlogic.gdx.graphics.g2d.SpriteBatch;  
**import** com.mygdx.game.MyGdxGame;  
  
**public class** GameOverState **extends** State {  
  
 **private** Texture **gameover**;  
  
 */\*\*  
 \* It is black screen which creates after the game ends and which waites for some taps  
 \* to create new game  
 \*/* GameOverState(GameStateManager gsm) {  
 **super**(gsm);  
 **camera**.setToOrtho(**false**, MyGdxGame.***WIDTH***, MyGdxGame.***HEIGHT***);  
 **gameover** = **new** Texture(**"gameover.jpg"**);  
 }  
  
 @Override  
 **protected void** handleInput() {  
 **if** (Gdx.*input*.justTouched()) {  
 **gsm**.set(**new** MenuState(**gsm**));  
 }  
 }  
  
 @Override  
 **public void** update(**float** dt) {  
 handleInput();  
 }  
  
 @Override  
 **public void** render(SpriteBatch sb) {  
 sb.setProjectionMatrix(**camera**.**combined**);  
 sb.begin();  
 sb.draw(**gameover**, **camera**.**position**.**x** - **gameover**.getWidth() / 2, **camera**.**position**.**y** - **gameover**.getHeight() / 2);  
 sb.end();  
 }  
  
 @Override  
 **public void** dispose() {  
 **gameover**.dispose();  
 System.***out***.println(**"GameOver Disposed"**);  
 }  
}

1. **Level.java**

**package** com.mygdx.game.levels;  
  
**public class** Level {  
 **private** String **mapURL**;  
 **private int levelPositionX**;  
 **private int levelPositionY**;  
  
 */\*\*  
 \* Creates new level  
 \*  
 \** ***@param mapURL*** *is path to level map  
 \** ***@param levelPositionX*** *is X coordinate of level  
 \** ***@param levelPositionY*** *is Y coordinate of level  
 \*/* Level(String mapURL, **int** levelPositionX, **int** levelPositionY) {  
 **this**.**mapURL** = mapURL;  
 **this**.**levelPositionX** = levelPositionX;  
 **this**.**levelPositionY** = levelPositionY;  
 }  
  
  
 **public int** getLevelPositionX() {  
 **return levelPositionX**;  
 }  
  
 **public int** getLevelPositionY() {  
 **return levelPositionY**;  
 }  
  
 String getMapURL() {  
 **return mapURL**;  
 }  
}

1. **TileType.java**

**package** com.mygdx.game.levels;  
  
**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.TiledMapTileLayer;  
**import** com.badlogic.gdx.math.Rectangle;  
**import** com.mygdx.game.creatures.Link;  
  
**public class** TileType {  
 **public static final int *TILE\_SIZE*** = 16;  
  
 **public** TileType() {  
  
 }  
  
 */\*\*  
 \* Method check for overlaps between player and door marks of current map  
 \*  
 \** ***@param tiledMap*** *is current map of game field  
 \** ***@param link*** *is main hero of game  
 \** ***@return*** *\*/* **public boolean** isDoor(TiledMap tiledMap, Link link) {  
 Rectangle rectangle = **new** Rectangle();  
 rectangle.**x** = link.getCreatureRectangle().**x**;  
 rectangle.**y** = link.getCreatureRectangle().**y**;  
 **for** (MapObject object : tiledMap.getLayers().get(1).getObjects().getByType(RectangleMapObject.**class**)) {  
 Rectangle rect = ((RectangleMapObject) object).getRectangle();  
 **if** (rect.overlaps(rectangle)) {  
 **return true**;  
 }  
 }  
 **return false**;  
 }  
  
 */\*\*  
 \* Method finds a tile from current player's tile  
 \*  
 \** ***@param dx*** *is difference of X coordinate between tiles  
 \** ***@param dy*** *is difference of Y coordinate between tiles  
 \** ***@param link*** *is the main hero  
 \** ***@param tiledMap*** *is current map  
 \** ***@return*** *id of needed cell  
 \*/* **private int** getTileId(**int** dx, **int** dy, Link link, TiledMap tiledMap) {  
 TiledMapTileLayer tileid = (TiledMapTileLayer) tiledMap.getLayers().get(0);  
 **int** x = (**int**) link.getPositionX() / TileType.***TILE\_SIZE***;  
 **int** y = (**int**) link.getPositionY() / TileType.***TILE\_SIZE***;  
 **return** tileid.getCell(x + dx, y + dy).getTile().getId();  
 }  
  
 */\*\*  
 \* Method checks if the player go away from the map  
 \*  
 \** ***@param link*** *is the main hero  
 \** ***@param tiledMap*** *is current map  
 \** ***@return*** *true if the player go away from the map  
 \*/* **public boolean** checkLevelEdge(Link link, TiledMap tiledMap) {  
 **try** {  
 **return** getTileId(0, 0, link, tiledMap) <= 0;  
 } **catch** (NullPointerException e) {  
 **return true**;  
 }  
 }  
  
 */\*\*  
 \* Method finds direction of next level  
 \*  
 \** ***@param link*** *is the main hero  
 \** ***@return*** *X and Y coordinates of next level  
 \*/* **public int**[] getEdgeDirection(Link link) {  
 Rectangle rectangle = link.getCreatureRectangle();  
 **int**[] array = **new int**[2];  
 **if** (rectangle.**x** < 0 && rectangle.**y** >= 0 && rectangle.**y** < ***TILE\_SIZE*** \* LevelManager.***MAP\_HEIGHT***) {  
 array[0] = -1;  
 } **else if** (rectangle.**x** >= 0 && rectangle.**y** > ***TILE\_SIZE*** \* LevelManager.***MAP\_HEIGHT*** &&  
 rectangle.**x** < ***TILE\_SIZE*** \* LevelManager.***MAP\_WIDTH***) {  
 array[1] = -1;  
 } **else if** (rectangle.**x** >= 0 && rectangle.**y** < 0 && rectangle.**x** < ***TILE\_SIZE*** \* LevelManager.***MAP\_WIDTH***) {  
 array[1] = 1;  
 } **else if** (rectangle.**x** > ***TILE\_SIZE*** \* LevelManager.***MAP\_WIDTH*** && rectangle.**y** >= 0  
 && rectangle.**y** < ***TILE\_SIZE*** \* LevelManager.***MAP\_HEIGHT***) {  
 array[0] = 1;  
 }  
 **return** array;  
 }  
  
}

1. **LevelManager.java**

**package** com.mygdx.game.levels;  
  
**import** com.badlogic.gdx.graphics.OrthographicCamera;  
**import** com.badlogic.gdx.maps.tiled.TiledMap;  
**import** com.badlogic.gdx.maps.tiled.TmxMapLoader;  
**import** com.badlogic.gdx.maps.tiled.renderers.OrthogonalTiledMapRenderer;  
**import** com.badlogic.gdx.utils.Array;  
  
**public class** LevelManager {  
 **private** Array<Level> **manager**;  
 **private** OrthographicCamera **camera**;  
 **private** TiledMap **tiledMap**;  
 **private int currentLevel**;  
 **private int currentDungeon**;  
 **private int pastDungeon** = 0;  
 **private** DungeonManager **dungeons**;  
 **public static final int *MAP\_WIDTH*** = 16;  
 **public static final int *MAP\_HEIGHT*** = 11;  
 **private** OrthogonalTiledMapRenderer **tiledMapRenderer**;  
  
 */\*\*  
 \* Creates manager which contains all levels and dungeons of game  
 \*  
 \** ***@param camera*** *is game camera  
 \** ***@param levelX*** *is current level X coordinate  
 \** ***@param levelY*** *is current level Y coordinate  
 \** ***@param inDungeon*** *true if player is in dungeon right now  
 \** ***@param dungeonX*** *is current dungeon X coordinate  
 \** ***@param dungeonY*** *is current dungeon Y coordinate  
 \*/* **public** LevelManager(OrthographicCamera camera, **int** levelX, **int** levelY, **boolean** inDungeon, **int** dungeonX, **int** dungeonY) {  
 **currentDungeon** = dungeonY \* 2 + dungeonX;  
 **currentLevel** = levelY \* 4 + levelX;  
 **manager** = **new** Array<Level>();  
 **this**.**camera** = camera;  
 **dungeons** = **new** DungeonManager();  
 **dungeons**.setDungeons();  
 startGame();  
 **dungeons**.setInDungeon(inDungeon);  
 **if** (**dungeons**.inDungeon()) {  
 **tiledMap** = **new** TmxMapLoader().load(**dungeons**.getCurrentDungeonLevel(**currentDungeon**).getMapURL());  
 } **else** {  
 **tiledMap** = **new** TmxMapLoader().load(**manager**.get(**currentLevel**).getMapURL());  
 }  
 **tiledMapRenderer** = **new** OrthogonalTiledMapRenderer(**tiledMap**);  
 }  
  
 */\*\*  
 \* Loads all level maps  
 \*/* **private void** startGame() {  
 Level testLevel = **new** Level(**"maps/map1 1.tmx"**, 0, 0);  
 Level testLevel1 = **new** Level(**"maps/map1 2.tmx"**, 1, 0);  
 Level testLevel2 = **new** Level(**"maps/map1 3.tmx"**, 2, 0);  
 Level testLevel3 = **new** Level(**"maps/map1 4.tmx"**, 3, 0);  
 Level testLevel4 = **new** Level(**"maps/map2 1.tmx"**, 0, 1);  
 Level testLevel5 = **new** Level(**"maps/map2 2.tmx"**, 1, 1);  
 Level testLevel6 = **new** Level(**"maps/map2 3.tmx"**, 2, 1);  
 Level testLevel7 = **new** Level(**"maps/map2 4.tmx"**, 3, 1);  
 Level testLevel8 = **new** Level(**"maps/map3 1.tmx"**, 0, 2);  
 Level testLevel9 = **new** Level(**"maps/map3 2.tmx"**, 1, 2);  
 Level testLevel10 = **new** Level(**"maps/map3 3.tmx"**, 2, 2);  
 Level testLevel11 = **new** Level(**"maps/map3 4.tmx"**, 3, 2);  
 Level testLevel12 = **new** Level(**"maps/map4 1.tmx"**, 0, 3);  
 Level testLevel13 = **new** Level(**"maps/map4 2.tmx"**, 1, 3);  
 Level testLevel14 = **new** Level(**"maps/map4 3.tmx"**, 2, 3);  
 Level testLevel15 = **new** Level(**"maps/map4 4.tmx"**, 3, 3);  
 **manager**.add(testLevel);  
 **manager**.add(testLevel1);  
 **manager**.add(testLevel2);  
 **manager**.add(testLevel3);  
 **manager**.add(testLevel4);  
 **manager**.add(testLevel5);  
 **manager**.add(testLevel6);  
 **manager**.add(testLevel7);  
 **manager**.add(testLevel8);  
 **manager**.add(testLevel9);  
 **manager**.add(testLevel10);  
 **manager**.add(testLevel11);  
 **manager**.add(testLevel12);  
 **manager**.add(testLevel13);  
 **manager**.add(testLevel14);  
 **manager**.add(testLevel15);  
 }  
  
 */\*\*  
 \* Method draw current map  
 \*/* **public void** drawCreatedLevel() {  
 **tiledMapRenderer**.setView(**camera**);  
 **tiledMapRenderer**.render();  
 }  
  
 **public void** goInDungeon() {  
 **dungeons**.goInDungeon();  
 }  
  
 **public void** exitFromDungeon() {  
 **dungeons**.exitFromDungeon();  
 }  
  
 **public boolean** inDungeon() {  
 **return dungeons**.inDungeon();  
 }  
  
 */\*\*  
 \* Method changes current level  
 \*  
 \** ***@param dx*** *is X difference coordinate of next level  
 \** ***@param dy*** *is Y difference coordinate of next level  
 \*/* **public void** changeLevel(**int** dx, **int** dy) {  
 **if** (inDungeon()) {  
 **pastDungeon** = **currentDungeon**;  
 **currentDungeon** += dy \* 2 + dx;  
 } **else** {  
 **currentLevel** += dy \* 4 + dx;  
 }  
 **if** (dungeonExit()) {  
 **currentLevel** = 1;  
 }  
 }  
  
 **public** TiledMap getTiledMap() {  
 **return tiledMap**;  
 }  
  
 **public void** dispose() {  
 **tiledMap**.dispose();  
 }  
  
 */\*\*  
 \* Method checks if player just go in dungeon  
 \*/* **public boolean** dungeonStart() {  
 **return currentDungeon** == 0 && **pastDungeon** == 0 && inDungeon();  
 }  
  
 */\*\*  
 \* Method checks if player just find an exit from dungeon  
 \*/* **public boolean** dungeonExit() {  
 **if** (**currentDungeon** == 0 && !inDungeon() && **currentLevel** == 1) {  
 **pastDungeon** = 0;  
 **return true**;  
 }  
 **return false**;  
 }  
}

1. **MusicManager.java**

**package** com.mygdx.game.levels;  
  
**import** com.badlogic.gdx.assets.AssetManager;  
**import** com.badlogic.gdx.audio.Music;  
**import** com.badlogic.gdx.audio.Sound;  
  
**public class** MusicManager {  
 **private** AssetManager **manager**;  
  
 */\*\*  
 \* Creates all music of game  
 \*/* **public** MusicManager() {  
 **manager** = **new** AssetManager();  
 **manager**.load(**"audio/music/Legend of Zelda - NES - Dungeon Theme.mp3"**, Music.**class**);  
 **manager**.load(**"audio/music/The Legend of Zelda - NES - Overworld.mp3"**, Music.**class**);  
 **manager**.load(**"audio/sounds/Sound Effect (3).wav"**, Sound.**class**);  
 **manager**.load(**"audio/sounds/Sound Effect (20).wav"**, Sound.**class**);  
 **manager**.load(**"audio/sounds/Sound Effect (21).wav"**, Sound.**class**);  
 **manager**.finishLoading();  
 }  
  
 **public** AssetManager getManager() {  
 **return manager**;  
 }  
  
 **public void** dispose() {  
 **manager**.dispose();  
 }  
}

1. **DungeonManager.java**

**package** com.mygdx.game.levels;  
  
**import** com.badlogic.gdx.utils.Array;  
  
**class** DungeonManager {  
 **private** Array<Level> **manager**;  
 **private boolean inDungeon** = **false**;  
  
 */\*\*  
 \* Creates manager which contains all dungeons  
 \*/* DungeonManager() {  
 **manager** = **new** Array<Level>();  
 }  
  
 */\*\*  
 \* Method loads all dungeon maps  
 \*/* **void** setDungeons() {  
 Level testLevel = **new** Level(**"maps/dungeon1 1.tmx"**, 0, 0);  
 Level testLevel1 = **new** Level(**"maps/dungeon1 2.tmx"**, 1, 0);  
 Level testLevel2 = **new** Level(**"maps/dungeon2 1.tmx"**, 0, 1);  
 Level testLevel3 = **new** Level(**"maps/dungeon2 2.tmx"**, 1, 1);  
 **manager**.add(testLevel);  
 **manager**.add(testLevel1);  
 **manager**.add(testLevel2);  
 **manager**.add(testLevel3);  
 }  
  
 **boolean** inDungeon() {  
 **return inDungeon**;  
 }  
  
 **void** goInDungeon() {  
 **inDungeon** = **true**;  
 }  
  
 **void** exitFromDungeon() {  
 **inDungeon** = **false**;  
 }  
  
 Level getCurrentDungeonLevel(**int** id) {  
 **return manager**.get(id);  
 }  
  
 **void** setInDungeon(**boolean** inDungeon) {  
 **this**.**inDungeon** = inDungeon;  
 }  
  
}

1. **Animation.java**

**package** com.mygdx.game.creatures;  
  
**import** com.badlogic.gdx.graphics.Texture;  
**import** com.badlogic.gdx.utils.Array;  
  
**class** Animation {  
  
 **private** Array<Texture> **frames**;  
 **private float maxFrameTime**;  
 **private float currentFrameTime**;  
 **private int frameCount**;  
 **private int frame**;  
  
 */\*\*  
 \* Creates all animations in game  
 \*  
 \** ***@param texture1*** *first texture of animation  
 \** ***@param texture2*** *second texture of animation  
 \** ***@param frameCount*** *counter for textures  
 \** ***@param cycleTime*** *time of aniamtion  
 \*/* Animation(Texture texture1, Texture texture2, **int** frameCount, **float** cycleTime) {  
 **frames** = **new** Array<Texture>();  
 **frames**.add(texture1);  
 **frames**.add(texture2);  
 **this**.**frameCount** = frameCount;  
 **maxFrameTime** = cycleTime / frameCount;  
 **frame** = 0;  
 }  
  
 */\*\*  
 \* Method updates current texture of animation  
 \*  
 \** ***@param dt*** *is delta time  
 \*/* **void** update(**float** dt) {  
 **currentFrameTime** += dt;  
 **if** (**currentFrameTime** > **maxFrameTime**) {  
 **frame**++;  
 **currentFrameTime** = 0;  
 }  
 **if** (**frame** >= **frameCount**) {  
 **frame** = 0;  
 }  
 }  
  
 */\*\*  
 \* Method makes texture for player when he doesn't move  
 \*  
 \** ***@param direction*** *is direction of player's movement  
 \** ***@return*** *stay texture  
 \*/* **static** Texture stayTexture(**int** direction) {  
 **if** (direction == 1) {  
 **return new** Texture(**"link images/link7.png"**);  
 } **else if** (direction == 2) {  
 **return new** Texture(**"link images/link4.png"**);  
 } **else if** (direction == 4) {  
 **return new** Texture(**"link images/link3.png"**);  
 } **else** {  
 **return new** Texture(**"link images/link1.png"**);  
 }  
 }  
  
 Texture getTexture() {  
 **return frames**.get(**frame**);  
 }  
  
 **void** dispose() {  
 **frames**.get(0).dispose();  
 **frames**.get(1).dispose();  
 }  
  
}

1. **Creature.java**

**package** com.mygdx.game.creatures;  
  
**import** com.badlogic.gdx.audio.Sound;  
**import** com.badlogic.gdx.graphics.Texture;  
**import** com.badlogic.gdx.math.Rectangle;  
**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.FixtureDef;  
**import** com.badlogic.gdx.physics.box2d.PolygonShape;  
**import** com.badlogic.gdx.physics.box2d.World;  
**import** com.mygdx.game.levels.TileType;  
**import** com.mygdx.game.states.PlayState;  
  
  
**public abstract class** Creature {  
 **private** Texture **creature**;  
 **private** Rectangle **creatureRectangle**;  
 **private** PlayState **state**;  
 **private int health**;  
 **private int damage**;  
 **private** Animation **goLeft**;  
 **private** Animation **goRight**;  
 **private** Animation **goUp**;  
 **private** Animation **goDown**;  
 **private** Animation **attackUp**;  
 **private** Animation **attackDown**;  
 **private** Animation **attackLeft**;  
 **private** Animation **attackRight**;  
 **private** Sound **deathSound**;  
 **private** World **world**;  
 **private** Body **body**;  
 **private int direction** = 3;  
 **private boolean countActive** = **true**;  
 **private float lastDamage** = 0;  
  
 */\*\*  
 \* Creates all creatures of game  
 \*  
 \** ***@param x*** *is X coordinate of creature  
 \** ***@param y*** *is Y coordinate of creature  
 \** ***@param texture*** *is start texture of creature  
 \** ***@param health*** *is start health of creature  
 \** ***@param damage*** *is damage of creature  
 \** ***@param goLeft*** *is animation of left side move  
 \** ***@param goRight*** *is animation of right side move  
 \** ***@param goUp*** *is animation of up side move  
 \** ***@param goDown*** *is animation of down side move  
 \** ***@param attackLeft*** *is animation of left side attack  
 \** ***@param attackRight*** *is animation of right side attack  
 \** ***@param attackUp*** *is animation of up side attack  
 \** ***@param attackDown*** *is animation of down side attack  
 \** ***@param state*** *is current PlayState object  
 \*/* Creature(**int** x, **int** y, String texture, **int** health, **int** damage, Animation goLeft,  
 Animation goRight, Animation goUp, Animation goDown, Animation attackLeft,  
 Animation attackRight, Animation attackUp, Animation attackDown, PlayState state) {  
 **creature** = **new** Texture(texture);  
 **creatureRectangle** = **new** Rectangle();  
 **this**.**state** = state;  
 **creatureRectangle**.**x** = x;  
 **creatureRectangle**.**y** = y;  
 **creatureRectangle**.**width** = **creature**.getWidth();  
 **creatureRectangle**.**height** = **creature**.getHeight();  
 **this**.**health** = health;  
 **this**.**damage** = damage;  
 **this**.**goLeft** = goLeft;  
 **this**.**goRight** = goRight;  
 **this**.**goDown** = goDown;  
 **this**.**goUp** = goUp;  
 **this**.**attackLeft** = attackLeft;  
 **this**.**attackRight** = attackRight;  
 **this**.**attackUp** = attackUp;  
 **this**.**attackDown** = attackDown;  
 **this**.**world** = state.getWorld();  
 **this**.**deathSound** = state.getMusicManager().getManager().get(**"audio/sounds/Sound Effect (3).wav"**, Sound.**class**);  
 createBody();  
 **this**.**body**.setActive(**true**);  
 }  
  
 */\*\*  
 \* Method creates body for creature  
 \*/* **private void** createBody() {  
 BodyDef bodyDef = **new** BodyDef();  
 PolygonShape shape = **new** PolygonShape();  
 FixtureDef fixtureDef = **new** FixtureDef();  
 bodyDef.**type** = BodyDef.BodyType.***DynamicBody***;  
 bodyDef.**position**.set(**creatureRectangle**.**x**, **creatureRectangle**.**y**);  
 **body** = **world**.createBody(bodyDef);  
 shape.setAsBox((**float**) TileType.***TILE\_SIZE*** / 2 - 4, (**float**) TileType.***TILE\_SIZE*** / 2 - 2);  
 fixtureDef.**shape** = shape;  
 **body**.createFixture(fixtureDef);  
 }  
  
 **public int** getDamage() {  
 **return damage**;  
 }  
  
 **public int** getHealth() {  
 **return health**;  
 }  
  
 **void** setHP(**int** health) {  
 **this**.**health** = health;  
 }  
  
 **public** Texture getCreature() {  
 **return creature**;  
 }  
  
 **public int** getDirection() {  
 **return direction**;  
 }  
  
 **public** Rectangle getCreatureRectangle() {  
 **return creatureRectangle**;  
 }  
  
 */\*\*  
 \* Method updates current texture for creature to next left side movement animation texture  
 \*  
 \** ***@param value*** *is delta time  
 \*/* **public void** goLeft(**float** value) {  
 **body**.applyLinearImpulse(**new** Vector2(-100f, 0), **body**.getWorldCenter(), **true**);  
 **goLeft**.update(value);  
 **creature** = **goLeft**.getTexture();  
 **direction** = 4;  
 }  
  
 */\*\*  
 \* Method updates current texture for creature to next right side movement animation texture  
 \*  
 \** ***@param value*** *is delta time  
 \*/* **public void** goRight(**float** value) {  
 **body**.applyLinearImpulse(**new** Vector2(100f, 0), **body**.getWorldCenter(), **true**);  
 **goRight**.update(value);  
 **creature** = **goRight**.getTexture();  
 **direction** = 2;  
 }  
  
 */\*\*  
 \* Method updates current texture for creature to next up side movement animation texture  
 \*  
 \** ***@param value*** *is delta time  
 \*/* **public void** goUp(**float** value) {  
 **body**.applyLinearImpulse(**new** Vector2(0, 100f), **body**.getWorldCenter(), **true**);  
 **goUp**.update(value);  
 **creature** = **goUp**.getTexture();  
 **direction** = 1;  
 }  
  
 */\*\*  
 \* Method updates current texture for creature to next down side movement animation texture  
 \*  
 \** ***@param value*** *is delta time  
 \*/* **public void** goDown(**float** value) {  
 **body**.applyLinearImpulse(**new** Vector2(0, -100f), **body**.getWorldCenter(), **true**);  
 **goDown**.update(value);  
 **creature** = **goDown**.getTexture();  
 **direction** = 3;  
 }  
  
 */\*\*  
 \* Method updates all creature's positions  
 \*/* **public void** update() {  
 **creatureRectangle**.**x** = **body**.getPosition().**x**;  
 **creatureRectangle**.**y** = **body**.getPosition().**y**;  
 **creatureRectangle**.**width** = **creature**.getWidth();  
 **creatureRectangle**.**height** = **creature**.getHeight();  
 Vector2 vector = **new** Vector2();  
 vector.**x** = 0f;  
 vector.**y** = 0f;  
 **body**.setLinearVelocity(vector);  
 }  
  
 **public void** setStayPosition() {  
 **creature** = Animation.*stayTexture*(**direction**);  
 }  
  
 */\*\*  
 \* Method changes current texture to attack texture  
 \*/* **public void** attack() {  
 **if** (**direction** == 3) {  
 **creature** = **attackDown**.getTexture();  
 } **else if** (**direction** == 1) {  
 **creature** = **attackUp**.getTexture();  
 } **else if** (**direction** == 4) {  
 **creature** = **attackLeft**.getTexture();  
 } **else if** (**direction** == 2) {  
 **creature** = **attackRight**.getTexture();  
 }  
 }  
  
 */\*\*  
 \* Method tries to make damage to other creature and if it does then creates a delay for next hit  
 \*  
 \** ***@param damage*** *is creatures damage  
 \** ***@param creature*** *is damaged creature  
 \** ***@param dt*** *is delta time  
 \*/* **public void** makeDamage(**int** damage, Creature creature, **float** dt) {  
 **if** (**countActive**) {  
 **lastDamage** += dt;  
 }  
 **if** (**lastDamage** >= 0.25) {  
 **countActive** = **false**;  
 }  
 **if** (checkBoxes(creature) && **lastDamage** >= 0.25) {  
 creature.setHealth(damage);  
 **lastDamage** = 0;  
 **countActive** = **true**;  
 }  
 }  
  
 **public float** getPositionX() {  
 **return body**.getWorldCenter().**x**;  
 }  
  
 **public float** getPositionY() {  
 **return body**.getWorldCenter().**y**;  
 }  
  
 **public** Body getBody() {  
 **return body**;  
 }  
  
 **private void** setHealth(**int** damage) {  
 **health** -= damage;  
 }  
  
 */\*\*  
 \* Method checks if this creature's texture is in other creature's texture  
 \*  
 \** ***@param creature*** *is other creature  
 \** ***@return*** *\*/* **private boolean** checkBoxes(Creature creature) {  
 **return creatureRectangle**.**x** < creature.**creatureRectangle**.**x** + creature.**creatureRectangle**.**width** && **creatureRectangle**.**x** + **creatureRectangle**.**width** > creature.**creatureRectangle**.**x** && **creatureRectangle**.**y** < creature.**creatureRectangle**.**y** + creature.**creatureRectangle**.**height** && **creatureRectangle**.**y** + **creatureRectangle**.**height** > creature.**creatureRectangle**.**y**;  
 }  
  
 **boolean** checkBoxes(Rectangle rectangle) {  
 **return creatureRectangle**.**x** < rectangle.**x** + rectangle.**width** && **creatureRectangle**.**x** + **creatureRectangle**.**width** > rectangle.**x** && **creatureRectangle**.**y** < rectangle.**y** + rectangle.**height** && **creatureRectangle**.**y** + **creatureRectangle**.**height** > rectangle.**y**;  
 }  
  
 **public boolean** isDead() {  
 **return health** <= 0;  
 }  
  
 */\*\*  
 \* Method sets position for creature when is comes to new level  
 \*/* **public void** setNewPosition() {  
 **if** (**state**.getManager().dungeonStart()) {  
 **creatureRectangle**.**x** = 20;  
 **creatureRectangle**.**y** = 85;  
 } **else if** (**state**.getManager().dungeonExit()) {  
 **creatureRectangle**.**x** = 132;  
 **creatureRectangle**.**y** = 80;  
 } **else if** (**direction** == 1) {  
 **creatureRectangle**.**y** = TileType.***TILE\_SIZE***;  
 System.***out***.println(**creatureRectangle**.**x**);  
 System.***out***.println(**creatureRectangle**.**y**);  
 } **else if** (**direction** == 2) {  
 **creatureRectangle**.**x** = TileType.***TILE\_SIZE***;  
 System.***out***.println(**creatureRectangle**.**x**);  
 System.***out***.println(**creatureRectangle**.**y**);  
 } **else if** (**direction** == 3) {  
 **creatureRectangle**.**y** = 176 - TileType.***TILE\_SIZE***;  
 System.***out***.println(**creatureRectangle**.**x**);  
 System.***out***.println(**creatureRectangle**.**y**);  
 } **else if** (**direction** == 4) {  
 **creatureRectangle**.**x** = 256 - TileType.***TILE\_SIZE***;  
 System.***out***.println(**creatureRectangle**.**x**);  
 System.***out***.println(**creatureRectangle**.**y**);  
 }  
 }  
  
 **public void** playDeathSound() {  
 **deathSound**.play(0.1f);  
 }  
  
 **public void** dispose() {  
 **creature**.dispose();  
 **goLeft**.dispose();  
 **goRight**.dispose();  
 **goUp**.dispose();  
 **goDown**.dispose();  
 **attackLeft**.dispose();  
 **attackRight**.dispose();  
 **attackUp**.dispose();  
 **attackDown**.dispose();  
 }  
}

1. **CreatureAI.java**

**package** com.mygdx.game.creatures;  
  
**import** java.util.Random;  
  
**class** CreatureAI {  
 **private** Creature **creature**;  
 **private int direction** = 3;  
 **private float lastPoint** = 0;  
  
 */\*\*  
 \* Creates simplest ai  
 \*  
 \** ***@param creature*** *is creatures for ai  
 \*/* CreatureAI(Creature creature) {  
 **this**.**creature** = creature;  
 }  
  
 */\*\*  
 \* Method randomizes creatures movement direction  
 \*  
 \** ***@param dt*** *is delta time  
 \*/* **void** update(**float** dt) {  
 **lastPoint** += dt;  
 **if** (**lastPoint** >= 0.5f) {  
 **lastPoint** = 0;  
 changeDirection();  
 }  
 **if** (**direction** == 1) {  
 **creature**.goUp(dt);  
 }  
 **if** (**direction** == 2) {  
 **creature**.goRight(dt);  
 }  
 **if** (**direction** == 3) {  
 **creature**.goDown(dt);  
 }  
 **if** (**direction** == 4) {  
 **creature**.goLeft(dt);  
 }  
 }  
  
 **private void** changeDirection() {  
 Random random = **new** Random();  
 **direction** = random.nextInt(4) + 1;  
 }  
}

1. **CreatureFactory.java**

**package** com.mygdx.game.creatures;  
  
**import** com.badlogic.gdx.graphics.Texture;  
**import** com.badlogic.gdx.math.Rectangle;  
**import** com.badlogic.gdx.utils.Array;  
**import** com.mygdx.game.levels.TileType;  
**import** com.mygdx.game.states.PlayState;  
  
**import** java.util.Random;  
  
**public class** CreatureFactory {  
 **private** Array<Creature> **enemies**;  
  
 */\*\*  
 \* Creates factory to player and all his enemies  
 \*/* **public** CreatureFactory() {  
 **enemies** = **new** Array<Creature>();  
 }  
  
 **public void** createEnemie(PlayState state) {  
 Random random = **new** Random();  
 **int** id = random.nextInt(3) + 1;  
 **if** (id == 1) {  
 createDeku(state);  
 } **else if** (id == 2) {  
 createMoblin(state);  
 } **else if** (id == 3) {  
 createLynel(state);  
 } **else** {  
 createDeku(state);  
 }  
 }  
  
 **private void** createDeku(PlayState state) {  
 Random random = **new** Random();  
 Rectangle rectangle = **new** Rectangle();  
 rectangle.**x** = random.nextInt(230) + 10;  
 rectangle.**y** = random.nextInt(150) + 10;  
 **while** (state.getWorldCreator().checkPhysicsBoxes(state, rectangle)) {  
 rectangle.**x** = random.nextInt(230) + 10;  
 rectangle.**y** = random.nextInt(150) + 10;  
 }  
 rectangle.**width** = TileType.***TILE\_SIZE***;  
 rectangle.**height** = TileType.***TILE\_SIZE***;  
 Animation goDown = **new** Animation(**new** Texture(**"enemie images/enemie1.png"**), **new** Texture(**"enemie images/enemie2.png"**), 2, 0.5f);  
 Animation goUp = **new** Animation(**new** Texture(**"enemie images/enemie5.png"**), **new** Texture(**"enemie images/enemie6.png"**), 2, 0.5f);  
 Animation goLeft = **new** Animation(**new** Texture(**"enemie images/enemie3.png"**), **new** Texture(**"enemie images/enemie4.png"**), 2, 0.5f);  
 Animation goRight = **new** Animation(**new** Texture(**"enemie images/enemie7.png"**), **new** Texture(**"enemie images/enemie8.png"**), 2, 0.5f);  
 Animation attackDown = **new** Animation(**new** Texture(**"enemie images/enemie1.png"**), **new** Texture(**"enemie images/enemie2.png"**), 2, 0.5f);  
 Animation attackUp = **new** Animation(**new** Texture(**"enemie images/enemie1.png"**), **new** Texture(**"enemie images/enemie2.png"**), 2, 0.5f);  
 Animation attackLeft = **new** Animation(**new** Texture(**"enemie images/enemie1.png"**), **new** Texture(**"enemie images/enemie2.png"**), 2, 0.5f);  
 Animation attackRight = **new** Animation(**new** Texture(**"enemie images/enemie1.png"**), **new** Texture(**"enemie images/enemie2.png"**), 2, 0.5f);  
 Deku enemy = **new** Deku((**int**) rectangle.**x**, (**int**) rectangle.**y**, **"enemie images/enemie1.png"**, 1, 1, goLeft, goRight, goUp, goDown, attackLeft, attackRight, attackUp, attackDown, state);  
 **enemies**.add(enemy);  
 }  
  
 **private void** createMoblin(PlayState state) {  
 Random random = **new** Random();  
 Rectangle rectangle = **new** Rectangle();  
 rectangle.**x** = random.nextInt(230) + 10;  
 rectangle.**y** = random.nextInt(150) + 10;  
 **while** (state.getWorldCreator().checkPhysicsBoxes(state, rectangle)) {  
 rectangle.**x** = random.nextInt(230) + 10;  
 rectangle.**y** = random.nextInt(150) + 10;  
 }  
 rectangle.**width** = TileType.***TILE\_SIZE***;  
 rectangle.**height** = TileType.***TILE\_SIZE***;  
 Animation goDown = **new** Animation(**new** Texture(**"enemie images/Moblin1.png"**), **new** Texture(**"enemie images/Moblin2.png"**), 2, 0.5f);  
 Animation goUp = **new** Animation(**new** Texture(**"enemie images/Moblin5.png"**), **new** Texture(**"enemie images/Moblin6.png"**), 2, 0.5f);  
 Animation goLeft = **new** Animation(**new** Texture(**"enemie images/Moblin3.png"**), **new** Texture(**"enemie images/Moblin4.png"**), 2, 0.5f);  
 Animation goRight = **new** Animation(**new** Texture(**"enemie images/Moblin7.png"**), **new** Texture(**"enemie images/Moblin8.png"**), 2, 0.5f);  
 Animation attackDown = **new** Animation(**new** Texture(**"enemie images/Moblin1.png"**), **new** Texture(**"enemie images/Moblin2.png"**), 2, 0.5f);  
 Animation attackUp = **new** Animation(**new** Texture(**"enemie images/Moblin1.png"**), **new** Texture(**"enemie images/Moblin2.png"**), 2, 0.5f);  
 Animation attackLeft = **new** Animation(**new** Texture(**"enemie images/Moblin1.png"**), **new** Texture(**"enemie images/Moblin2.png"**), 2, 0.5f);  
 Animation attackRight = **new** Animation(**new** Texture(**"enemie images/Moblin1.png"**), **new** Texture(**"enemie images/Moblin2.png"**), 2, 0.5f);  
 Deku enemy = **new** Deku((**int**) rectangle.**x**, (**int**) rectangle.**y**, **"enemie images/Moblin1.png"**, 1, 5, goLeft, goRight, goUp, goDown, attackLeft, attackRight, attackUp, attackDown, state);  
 **enemies**.add(enemy);  
 }  
  
 **private void** createLynel(PlayState state) {  
 Random random = **new** Random();  
 Rectangle rectangle = **new** Rectangle();  
 rectangle.**x** = random.nextInt(230) + 10;  
 rectangle.**y** = random.nextInt(150) + 10;  
 **while** (state.getWorldCreator().checkPhysicsBoxes(state, rectangle)) {  
 rectangle.**x** = random.nextInt(230) + 10;  
 rectangle.**y** = random.nextInt(150) + 10;  
 }  
 rectangle.**width** = TileType.***TILE\_SIZE***;  
 rectangle.**height** = TileType.***TILE\_SIZE***;  
 Animation goDown = **new** Animation(**new** Texture(**"enemie images/Lynel1.png"**), **new** Texture(**"enemie images/Lynel2.png"**), 2, 0.5f);  
 Animation goUp = **new** Animation(**new** Texture(**"enemie images/Lynel5.png"**), **new** Texture(**"enemie images/Lynel6.png"**), 2, 0.5f);  
 Animation goLeft = **new** Animation(**new** Texture(**"enemie images/Lynel3.png"**), **new** Texture(**"enemie images/Lynel4.png"**), 2, 0.5f);  
 Animation goRight = **new** Animation(**new** Texture(**"enemie images/Lynel7.png"**), **new** Texture(**"enemie images/Lynel8.png"**), 2, 0.5f);  
 Animation attackDown = **new** Animation(**new** Texture(**"enemie images/Lynel1.png"**), **new** Texture(**"enemie images/Lynel2.png"**), 2, 0.5f);  
 Animation attackUp = **new** Animation(**new** Texture(**"enemie images/Lynel1.png"**), **new** Texture(**"enemie images/Lynel2.png"**), 2, 0.5f);  
 Animation attackLeft = **new** Animation(**new** Texture(**"enemie images/Lynel1.png"**), **new** Texture(**"enemie images/Lynel2.png"**), 2, 0.5f);  
 Animation attackRight = **new** Animation(**new** Texture(**"enemie images/Lynel1.png"**), **new** Texture(**"enemie images/Lynel2.png"**), 2, 0.5f);  
 Deku enemy = **new** Deku((**int**) rectangle.**x**, (**int**) rectangle.**y**, **"enemie images/Lynel1.png"**, 1, 4, goLeft, goRight, goUp, goDown, attackLeft, attackRight, attackUp, attackDown, state);  
 **enemies**.add(enemy);  
 }  
  
 **public** Link createPlayer(PlayState state, **float** LinkX, **float** LinkY) {  
 Animation goLeft = **new** Animation(**new** Texture(**"link images/link2.png"**), **new** Texture(**"link images/link3.png"**), 2, 0.3f);  
 Animation goRight = **new** Animation(**new** Texture(**"link images/link4.png"**), **new** Texture(**"link images/link5.png"**), 2, 0.3f);  
 Animation goUp = **new** Animation(**new** Texture(**"link images/link6.png"**), **new** Texture(**"link images/link7.png"**), 2, 0.3f);  
 Animation goDown = **new** Animation(**new** Texture(**"link images/link8.png"**), **new** Texture(**"link images/link9.png"**), 2, 0.3f);  
 Animation attackLeft = **new** Animation(**new** Texture(**"link images/link10.png"**), **new** Texture(**"link images/link3.png"**), 2, 0.3f);  
 Animation attackRight = **new** Animation(**new** Texture(**"link images/link11.png"**), **new** Texture(**"link images/link4.png"**), 2, 0.3f);  
 Animation attackUp = **new** Animation(**new** Texture(**"link images/link12.png"**), **new** Texture(**"link images/link6.png"**), 2, 0.3f);  
 Animation attackDown = **new** Animation(**new** Texture(**"link images/link13.png"**), **new** Texture(**"link images/link8.png"**), 2, 0.3f);  
 **return new** Link((**int**) LinkX, (**int**) LinkY, **"link images/link1.png"**, 200, 2, goLeft, goRight, goUp, goDown, attackLeft, attackRight, attackUp, attackDown, state);  
 }  
  
 **public** Array<Creature> getEnemies() {  
 **return enemies**;  
 }  
  
 **public void** delete(Creature creature) {  
 **enemies**.removeValue(creature, **true**);  
 }  
  
 **public void** dispose() {  
 **for** (Creature enemie : **enemies**) {  
 enemie.dispose();  
 }  
 }  
}

1. **Deku.java**

**package** com.mygdx.game.creatures;  
  
**import** com.mygdx.game.states.PlayState;  
  
**public class** Deku **extends** Creature {  
  
 **private** CreatureAI **ai**;  
  
 Deku(**int** x, **int** y, String texture, **int** health, **int** damage, Animation goLeft, Animation goRight, Animation goUp, Animation goDown, Animation attackLeft, Animation attackRight, Animation attackUp, Animation attackDown, PlayState state) {  
 **super**(x, y, texture, health, damage, goLeft, goRight, goUp, goDown, attackLeft, attackRight, attackUp, attackDown, state);  
 **this**.**ai** = **new** CreatureAI(**this**);  
 }  
  
 **public void** update(**float** dt) {  
 update();  
 **if** (!isDead()) {  
 **ai**.update(dt);  
 }  
 }  
}

1. **Link.java**

**package** com.mygdx.game.creatures;  
  
**import** com.badlogic.gdx.graphics.Texture;  
**import** com.mygdx.game.states.PlayState;  
  
  
**public class** Link **extends** Creature {  
  
 **private int rupees**;  
  
 Link(**int** x, **int** y, String texture, **int** health, **int** damage, Animation goLeft,  
 Animation goRight, Animation goUp, Animation goDown, Animation attackLeft,  
 Animation attackRight, Animation attackUp, Animation attackDown, PlayState state) {  
 **super**(x, y, texture, health, damage, goLeft, goRight, goUp, goDown, attackLeft,  
 attackRight, attackUp, attackDown, state);  
 **this**.**rupees** = 0;  
 setHP(state.getLinkHealth());  
 }  
  
  
 **public** Texture getLink() {  
 **return** getCreature();  
 }  
  
  
 **public int** getRupees() {  
 **return rupees**;  
 }  
  
}

1. **Lynel.java**

**package** com.mygdx.game.creatures;  
  
**import** com.mygdx.game.states.PlayState;  
  
  
**public class** Lynel **extends** Creature {  
 **private** CreatureAI **ai**;  
  
 Lynel(**int** x, **int** y, String texture, **int** health, **int** damage, Animation goLeft, Animation goRight, Animation goUp, Animation goDown, Animation attackLeft, Animation attackRight, Animation attackUp, Animation attackDown, PlayState state) {  
 **super**(x, y, texture, health, damage, goLeft, goRight, goUp, goDown, attackLeft, attackRight, attackUp, attackDown, state);  
 **this**.**ai** = **new** CreatureAI(**this**);  
 }  
  
 **public void** update(**float** dt) {  
 update();  
 **if** (!isDead()) {  
 **ai**.update(dt);  
 }  
 }  
}

1. **Moblin.java**

**package** com.mygdx.game.creatures;  
  
**import** com.mygdx.game.states.PlayState;  
  
**public class** Moblin **extends** Creature {  
 **private** CreatureAI **ai**;  
  
 Moblin(**int** x, **int** y, String texture, **int** health, **int** damage, Animation goLeft, Animation goRight, Animation goUp, Animation goDown, Animation attackLeft, Animation attackRight, Animation attackUp, Animation attackDown, PlayState state) {  
 **super**(x, y, texture, health, damage, goLeft, goRight, goUp, goDown, attackLeft, attackRight, attackUp, attackDown, state);  
 **this**.**ai** = **new** CreatureAI(**this**);  
 }  
  
 **public void** update(**float** dt) {  
 update();  
 **if** (!isDead()) {  
 **ai**.update(dt);  
 }  
 }  
}