Gymnázium, Praha 6, Arabská 14

Programování

ROČNÍKOVÁ PRÁCE

2019

Nguyen Tien, Grosman, Pelantová

Gymnázium, Praha 6, Arabská 14

Arabská 14, Praha 6, 160 00

ROČNÍKOVÁ PRÁCE

Předmět: Programování
Téma: RPG hra

Autoři: Nguyen Tien Dung, Lukáš Grosman, Michaela Pelantová
Třída: 3.E
Školní rok: 2018/2019
Třídní učitel: Mgr. Jana Urzová

Prohlašujeme, že jsme jedinými autory tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů udělujeme bezúplatně škole Gymnázium, Praha 6, Arabská 14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V Praze dne _____

_____

vlastnoruční podpis autorů

ANOTACE:

Tato ročníková práce se zabývá tvorbou a problematikou multiplayerové hry žánru Role Playing Game – RPG (v překladu: hra na hrdiny). V práci jsou uvedeny technologie/programy, které jsme při práci na projektu použili. Dále je v práci popsána tvorba příslušných textur, samotný průběh programování této hry a jak jsme řešili konkrétní problémy, které při tvorbě hry nastaly.

ANNOTATION:

This thesis is about the creation and matters of multiplayer Role Playing Game – RPG. In our thesis we included the technologies/programs we used when working on the project. The thesis contains description of creating applicable textures, the process of programming and how we solved particular problems which have appeared while working on the project.

OBSAH:

1. ÚVOD 6
1.1. Proč jsme si vybrali toto téma? 6
2. Box2D 9
3. Uživatelský vstup 10
4. Správce souborů 12
5. Obrazovky 13
5.1. LoadingScreen 13
5.2. MenuScreen 14
5.3. MainScreen 15
5.3.1. PlayerEntity 15
5.3.2. Projectile 15
6. Multiplayer 17
7. Grafická stránka projektu: 18
7.1. Tvorba textur 18
8. ZÁVĚR 19
9. ZDROJE: 20

1. ÚVOD

1.1. Proč jsme si vybrali toto téma?

Toto téma jsme si vybrali, protože sami rádi hrajeme různé (nejen RPG) hry a chtěli jsme si zkusit, jaké to je takovou RPG hru vlastnoručně naprogramovat.

1. Stáhnutí Libgdx a generování projektu

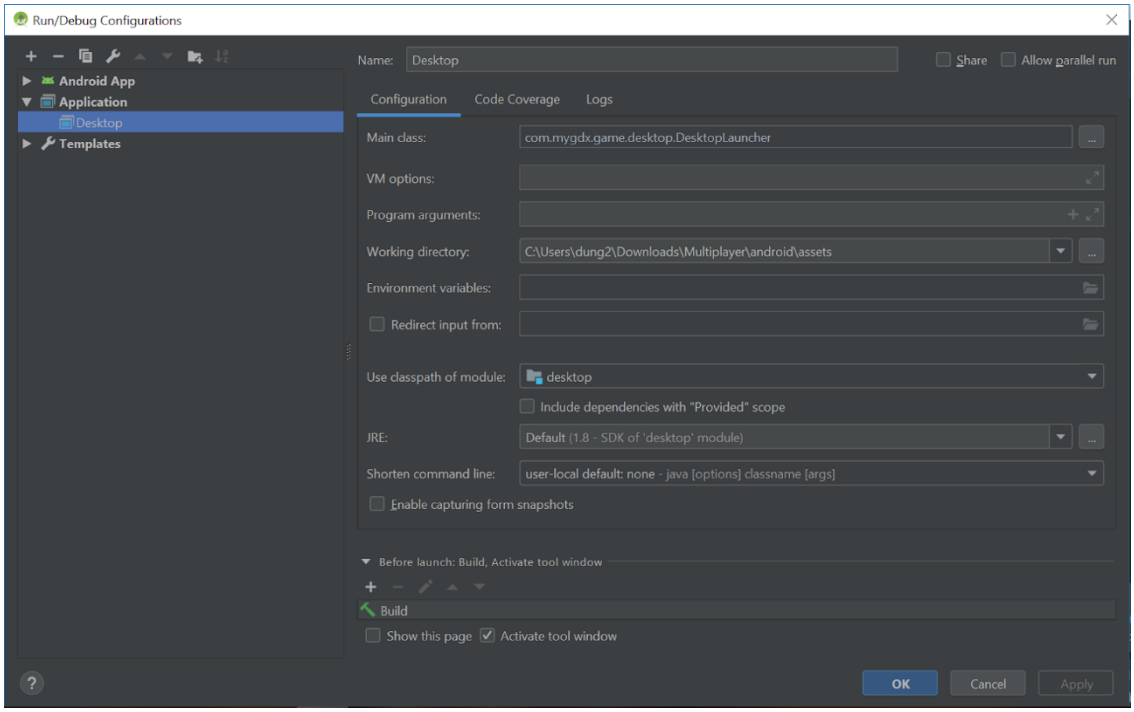
Po stáhnutí LibGDX z jejích oficiálních stránek[1] jsme dostali panel pro generování projektů (viz obr. 1).

Obr. 1

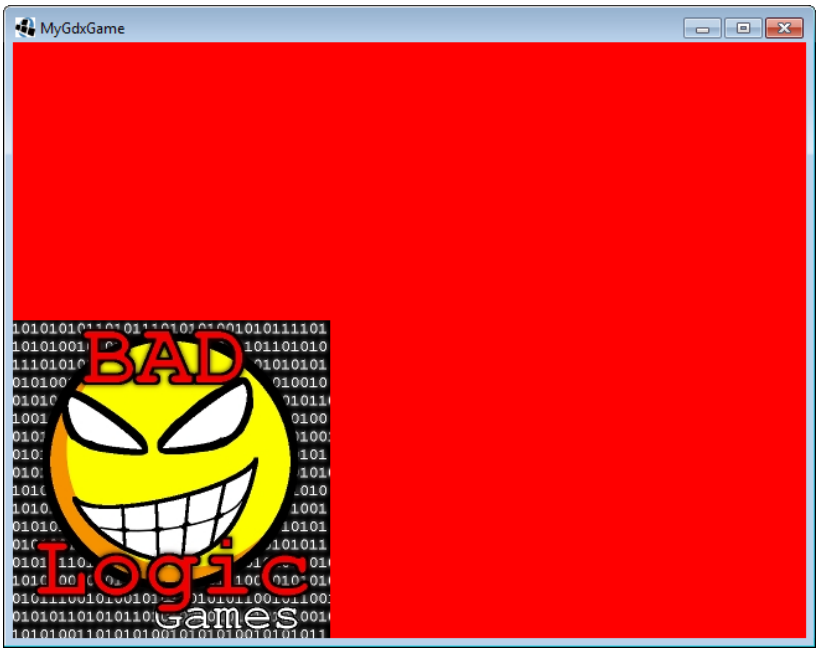
Po nastavení různých požadavků jsme nechali generovat projekt. Projekt jsme poté otevřeli v Android Studiu. LibGDX generuje jakýsi svůj *„Hello World“* program, ale abychom ho mohli zkompilovat, museli jsme nejprve udělat menší konfiguraci. V okně *„Run/Debug Configurations“* jsme přidali novou konfiguraci *„Application“* a nastavili (viz obr. 2).

Konečně jsme pak spustili program, který obsahoval jenom logo LibGDX a červené pozadí (viz obr. 3).

[1] *libGDX. Dostupné z URL:* <https://libgdx.badlogicgames.com/> *(navštíveno 7.3.2019)*


Run/Debug Configurations
Android App
Application
Desktop
Templates
Name: Desktop
Share
Allow parallel run
Configuration
Code Coverage
Logs
Main class: com.mygdx.game.desktop.DesktopLauncher
VM options:
Program arguments:
Working directory: C:\Users\dung2\Downloads\Multiplayer\android\assets
Environment variables:
Redirect input from:
Use classpath of module: desktop
Include dependencies with "Provided" scope
JRE: Default (1.8 - SDK of 'desktop' module)
Shorten command line: user-local default: none - java [options] classname [args]
Enable capturing form snapshots
Before launch: Build, Activate tool window
Build
Show this page
Activate tool window
OK
Cancel
Apply

Obr. 2


MyGdxGame
BAD
Logic
Games

Obr. 3

2. Box2D

Box2D nám velmi ulehčila práci s fyzikou hry. Tato knihovna nám pomohla vytvořit jakýsi imaginární model hry, díky ní můžeme definovat vlastnosti těl v našem světě a také nám ulehčila práci s hitboxy.

Box2D obsahuje třídu *World* (česky svět), do kterého jsme přidávali instance třídy *Body*[2] (česky tělo). Vlastnosti těl byli následující. Jedno z vlastností těl se definovalo přes třídu *BodyDef*, kde tělo jsme mohli určit buď jako dynamické, statické, nebo kinematické.

- ***Dynamcké tělo*** - působí na něj síly a je ovlivněno ostatními těly. Dynamické tělo bylo pro nás ideální jako tělo našeho hráče.
- ***Statické tělo*** - opak dynamického těla - nepůsobí na něj síly, není ovlivněno ostatními těly a nehýbe se. Je tedy ideální na modely budov a celkového terénu.
- ***Kinetické tělo*** - něco mezi statickým a dynamickým tělem. Stejně jako dynamické tělo se může hýbat, ale podobně jako statické na něj nepůsobí síly. V našem projektu jsme pro něj nenašli využití.

Pomocí třídy *FixtureDef* jsme tělu nadefinovali tření, odpor vůči vzduchu nebo hustotu. Abychom nemuseli definovat vlastnost pro každé tělo zvlášť, vytvořili jsme si třídu *BodyFactory*[3], kde jsme vytvořili metody, u kterých jsme jen určovali parametry. Na třídu *BodyFactory* jsme použili designový pattern singleton, jelikož jsme nechtělivíce instancí té třídy a také tím jsme ušetřili paměť.

Pro náš modelový svět jsme si vytvořili třídu *B2dModel*. Tady jsme přidávali naše těla do našeho modelového světa a řešili uživatelský vstup.

[2] *Objects/Bodies*. Dostupné z URL: <https://github.com/libgdx/libgdx/wiki/box2d#objectsbodies/> (navštíveno 8.5.2019)
[3] *Full LibGDX Game Tutorial – Box2D Body Factory* Dostupné z URL: <https://www.gamedevelopment.blog/full-libgdx-game-tutorial-box2d-body-factory/> (navštíveno 25.4.2019)

3. Uživatelský vstup

Uživatelský vstup se dal řešit dvěma způsoby:

- Horší způsob: podmínky se dají do *render()* metody, která by řešila vstup z klávesnice (viz obr. 4)
- Lepší způsob (který jsme použili): vytvoření třídy *KeyBoardController*[4], která pak implementuje interface *InputProcessor* (viz obr. 5)

```
@Override
public void render() {
    Gdx.gl.glClearColor(1, 1, 1, 1);
    Gdx.gl.glClear(GL10.GL_COLOR_BUFFER_BIT);

    if(Gdx.input.isKeyPressed(Input.Keys.LEFT)){
        if(Gdx.input.isKeyPressed(Input.Keys.CONTROL_LEFT))
            sprite.translateX(-1f);
        else
            sprite.translateX(-10.0f);
    }
    if(Gdx.input.isKeyPressed(Input.Keys.RIGHT)){
        if(Gdx.input.isKeyPressed(Input.Keys.CONTROL_LEFT))
            sprite.translateX(1f);
        else
            sprite.translateX(10.0f);
    }
    batch.begin();
    sprite.draw(batch);
    batch.end();
}
```

Obr. 4

```
public class MyInputProcessor implements InputProcessor {
   public boolean keyDown (int keycode) {
      return false;
   }

   public boolean keyUp (int keycode) {
      return false;
   }

   public boolean keyTyped (char character) {
      return false;
   }

   public boolean touchDown (int x, int y, int pointer, int button) {
      return false;
   }

   public boolean touchUp (int x, int y, int pointer, int button) {
      return false;
   }

   public boolean touchDragged (int x, int y, int pointer) {
      return false;
   }

   public boolean mouseMoved (int x, int y) {
      return false;
   }
```

Obr. 5

[4] *Event driven keyboard and mouse handling.* Dostupné z URL: <https://www.gamefromscratch.com/post/2013/10/15/LibGDX-Tutorial-4-Handling-the-mouse-and-keyboard.aspx/> (navštíveno 4.5.2019)

Instanci třídy jmeném *controller*[5] jsme pak vytvořili ve třídě *B2dModel*, která uživatelský vstup řešila v metodě *logicStep()* (viz obr. 6).

```
public void logicStep(float delta){
    if(controller.isMouseIDown && pointIntersectsBody(player,controller.mouseLocation)){
        System.out.println("Player was clicked");
    }
    if(controller.isMouseIDown){
        sword.setTransform(sword.getPosition(), angle: -90*BodyFactory.DEGTORAD);
        swordIsUp = false;
    }
    else if (!swordIsUp){
        sword.setTransform(sword.getPosition(), angle: 0);
    }

    if(controller.left){
       player.applyForceToCenter( forceX: -10, forceY: 0, wake: true);

    }else if(controller.right){
        player.applyForceToCenter( forceX: 10, forceY: 0, wake: true);
    }else if(controller.up){
        player.applyForceToCenter( forceX: 0, forceY: 10, wake: true);

    }else if(controller.down){
        player.applyForceToCenter( forceX: 0, forceY: -10, wake: true);
    }else{
        player.setLinearVelocity( vX: 0, vY: 0);
    }

    if(Gdx.input.justTouched()){
        System.out.println(Gdx.input.getX() + " " + Gdx.input.getY());
        listOfProjectiles.add(new ProjectileEntity( x: player.getPosition().x +2, y: player.getPosition().y+2));
    }
```

Obr. 6

[5] *Full LibGDX Game Tutorial – Input Controller*. Dostupné z URL: <https://www.gamedevelopment.blog/full-libgdx-game-tutorial-input-controller/> (navštíveno 25.4.2019)

4. Správce souborů

Knihovna LibGdx nám umožňuje použít třídu *AssetManager*[6], která přednačítá a donačítá naše soubory. Je to náš správce souborů, který se stará o paměť.

Vytvořili jsme třídu *B2dAssetManager* (viz obr. 7). Třídu jsme museli přejmenovat, protože třída *AssetManager* je již exitující třídou knihovny LibGdx. V této třídě jsme vytvořili metody, které načítají různé soubory, jako jsou textury postav, zbraní, popřípadě nějaké zvuky a hudbu. Důležité je, že všechny tyto soubory musí být ve složce *„Assets“* uvnitř složky *„Android“,* jinak se občas stane, že něco nefunguje (např. načítání mapy).

Zvažovali jsme použití TexturePackeru, který sjednotí více textur do jednoho souboru, typu *atlas*, a pak v tom souboru načte jistý region, ale pro jednoduchost jsme se rozhodli tento krok vynechat.

```
public class B2dAssetManager {

    public final AssetManager manager = new AssetManager();

    public final String playerImage = "images/player.png";
    public final String enemyImage = "images/enemy.png";
    public final String boingSound = "sounds/boing.wav";
    public final String pingSound = "sounds/ping.wav";
    public final String sword = "images/GoldenSpear.png";
    public final String skin = "skin/glassy-ui.json";
    public final String playingSong = "music/Vitas_7th_Element.mp3";
    public final String background = "skin/EternalBackground.png";

    public void queueAddMusic() { manager.load(playingSong, Music.class); }
    public void queueAddImages(){
        manager.load(playerImage, Texture.class);
        manager.load(enemyImage, Texture.class);
        manager.load(sword, Texture.class);

    }

    public void queueAddSkin(){
        SkinLoader.SkinParameter params = new SkinLoader.SkinParameter( textureAtlasPath: "skin/glassy-ui.atlas");
        manager.load(skin, Skin.class, params);
        manager.load(background,Texture.class);
    }

    public void queueAddBackground() { manager.load(background,Texture.class); }
```

Obr. 7

[6] *Full LibGDX Game Tutorial – Rendering and Asset manager*. Dostupné z URL: <https://www.gamedevelopment.blog/full-libgdx-game-tutorial-rendering-asset-manager/> (navštíveno 1.5.2019)

5. Obrazovky

Abychom mohli přepínat mezi obrazovkami, udělali jsme z naší třídy *MyGdxGame* potomka třídy *Game*, a pro třídy *LoadingScreen*, *MainScreen*, *MenuScreen* a *PreferencesScreen* jsme naimplementovali interface *Screen*. Poté jsme vytvořili metodu *changeScreen()*, která mezi nimi přepíná (viz obr. 8).

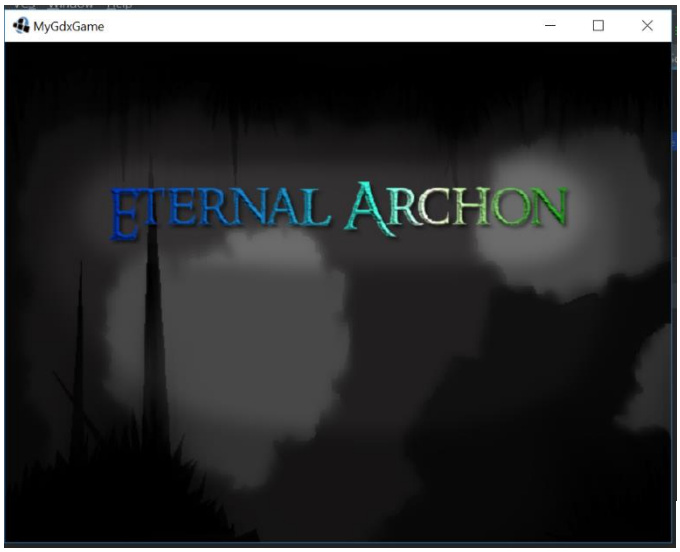
```
public void changeScreen(int screen){
    switch(screen){
        case MENU:
            if(menuScreen == null) menuScreen = new MenuScreen( main: this);
            this.setScreen(menuScreen);
            break;
        case PREFERENCES:
            if(preferencesScreen == null) preferencesScreen = new PreferencesScreen( main: this);
            this.setScreen(preferencesScreen);
            break;
        case APPLICATION:
            if(mainScreen == null) mainScreen = new MainScreen(this);
            this.setScreen(mainScreen);
            break;

    }
}
```

Obr. 8

5.1. LoadingScreen[7]

Načítací obrazovka (viz obr. 9) načítá soubory ke hře. Jelikož jsme si již vytvořili správce souborů, stačí zavolat jeho metody a naše soubory se načtou před začátkem hry. Pomocí třídy *Spritebatch* se pak vykreslí pozadí načítací obrazovky.



Obr. 9

[7] *Full LibGDX Game Tutorial – Loading Screen.* Dostupné z URL: <https://www.gamedevelopment.blog/full-libgdx-game-tutorial-loading-screen/> (navštíveno 1.5.2019)

5.2. MenuScreen

Poté, co se načtou soubory, se nám zobrazí menu s třemi tlačítky: *„Play“*, *„Preferenc“* a *„Exit“* (viz obr. 10). Abychom vytvořili tahle tři tlačítka, stačilo vytvořit instanci třídy *Stage* a pak instanci *Table*.

Pomocí metody *addActor()* jsme přidali instanci třídy *Table*, pak jsme vytvořili 3 instance třídy *TextButton* a přidali je do instance *Table*. Aby tlačítka něco dělala, přidali jsme každému tlačítku *eventListener*. Aby se pak vše zobrazilo, zavolali jsme v metodě *render()* metodu *stage.draw()* (viz obr. 11).

```
@Override
public void show() {
    Gdx.input.setInputProcessor(stage);
    Table table = new Table();
    table.setFillParent(true);
        table.setDebug(true);
        stage.addActor(table);


    TextButton newGame = new TextButton( text: "New Game", skin);
    TextButton preferences = new TextButton( text: "Preferences", skin);
    TextButton exit = new TextButton( text: "Exit", skin);

    table.add(newGame).fillX().uniformX();
    table.row().pad( top: 10, left: 0, bottom: 10, right: 0);
    table.add(preferences).fillX().uniformX();
    table.row();
    table.add(exit).fillX().uniformX();

    exit.addListener((ChangeListener) (event, actor) -> { Gdx.app.exit(); });

    newGame.addListener((ChangeListener) (event, actor) -> {
            parent.changeScreen(MyGdxGame.APPLICATION);
    });

    preferences.addListener((ChangeListener) (event, actor) -> {
            parent.changeScreen(MyGdxGame.PREFERENCES);
    });
```

Obr. 10

```
@Override
public void render(float delta) {
    Gdx.gl.glClearColor( red: 0f, green: 0f, blue: 0f, alpha: 1);
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

    stage.act(Math.min(Gdx.graphics.getDeltaTime(), 1 / 30f));
    stage.getBatch().begin();
    stage.draw();
    stage.getBatch().end();

}
```

Obr. 11

5.3. MainScreen

V hlavní obrazovce se odehrává nejdůležitější část - a to samotná hra. Vše se nám tu tedy vykresluje.

Jednou z těchto věcí je mapa hry. Pro načtení mapy jsme potřebovali instance třídy *OrthographicCamera*, jelikož pohled téhle kamery je ideální pro 2D hry. Dále jsme potřebovali samotný objekt mapy, třídu *TiledMap*, a pro vykreslení mapy třídu *TiledMapRenderer*.

Vše vypadalo v pořádku, dokud se nám nezačay objevovat 2 chyby. Jedním z problémů byla serializační chyba, což byla chyba se špatným formátem souboru. Druhým problémem bylo, že náš projekt nedokázal naši mapu načíst. Řešením pro oba problémy bylo umístit mapu do správného adresáře.

Poté začal program vypisovat jiný error. Tento error hlásil, že chybí jakési dokumenty. Uvědomili jsme si, že do adresáře asset patří všechno, co se týkalo té mapy (tzn. i všechny dokumenty typu *.tsx* a *.png*). Po implementaci se pak mapa konečně načetla.

Pro vykreslení ostatních objektů jsme opět použili třídu *SpriteBatch* a metodu *draw()*.

5.3.1. PlayerEntity

Tato třída reprezentuje hráče. Má dva konstruktory. Jeden konstruktor přidá tělo hráče do našeho modelového světa. Druhý konstruktor poskytuje informaci o své pozici pro multiplayer. Původně měla mít tato třída více atributů, ale kvůli časové tísni jsme se od toho oprostili.

5.3.2. Projectile

Jednou z mechanik naší hry je možnost hráče střílet tam, kam klikne myš[8]. Z toho důvodu jsme vytvořili třídu *Projectile*. Tento problém se dal řešit třemi způsoby.

Nejdříve jsme vyřešili otázku vertikálního letu střely. Aby letěla vertikálně, stačilo k hodnotě *y* přičítat chtěnou vzdálenost. Pro vykreslení projektilů se ve třídě

[8] *Firing bullet from gun's current position to mouse x & y position.* Dostupné z URL: <http://www.java-gaming.org/topics/firing-bullet-from-gun-s-current-position-to-mouse-x-y-position/35020/view.html> (navštíveno 5.5.2019)
Shooting bullet towards mouse position in Java/Slick2D. Dostupné z URL: <https://stackoverflow.com/questions/34122380/shooting-bullet-towards-mouse-position-in-java-slick2d> (navštíveno 5.5.2019)

MainScreen vytvořil *ArrayList*, do kterého jsme přidávali naše projektily. Všechny projektily, co byly v ArrayListu se pak vykreslily.

Samozřejmě jsme chtěli, aby naše projektily po určité vzdálenosti zmizely. Pro tento účel jsme vytvořili další *ArrayList*, protože nám problém hlásil, že nemůže odebírat a přidávat naše projektily zároveň.

Poté, co jsme to vyřešili pro vertikální pohyb, začli jsme zkoumat způsoby pro klik na myši. Našli jsme 3 matematické způsoby:

- přes výpočet úhlu vůči hráči a cíli (pomocí arkus tangens)
- Pythagorovou větou
- Vektory

Po vyzkoušení všech metod fungovaly vektory nejlépe a bylo méňě řádků kódu. Ani tak to nebylo ideální kvůli odlišným souřadnicovým systémům myši a zobrazování textur. Souřadnicový systém myši má počátek vlevo nahoře, kdežto zobrazování textur má počátek uprostřed okna. Projektily tedy lítají různě a ne kam jsme chtěli. Problém jsme kvůli časové tísni nevyřešili.

6. Multiplayer[9]

Java podporuje pouze sockety, takže jsme udělali multiplayer přes sockety. To znamená, že jsme vytvořili server, který na požadavek vysílal data ve formátu *JSON*.

Server jsme psali v *„node.js"*, kam jsme stáhli 2 knihovny: *express* a *socket.io*[10]. Aby server mohl komunikovat s naší aplikací, museli jsme stáhnout *socket.io* pro Javu a poté ještě změnit v android studiu soubor *„Gradle Scripts"* (viz obr. 12).

```
project(":core") {
    apply plugin: "java"


    dependencies {
        compile "com.badlogicgames.gdx:gdx:$gdxVersion"
        compile "com.badlogicgames.gdx:gdx-box2d:$gdxVersion"
        compile "com.github.nkzawa:socket.io-client:0.3.0"
    }
}
```

Obr. 12

[9] Brent Aureli. *Demo Multiplayer LibGDX game using a NodeJS server and Socket.IO for event firing*. Dostupné z URL: <https://github.com/BrentAureli/MultiplayerDemo> (navštíveno 8.5.2019)
[10] *Socket.IO-client Java.* Dostupné z URL: <https://github.com/socketio/socket.io-client-java> (navštíveno 3.5.2019)

7. Grafická stránka projektu:

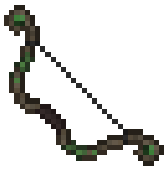
Vzhled našeho projektu se skládá z jednotlivých obrázků tvořených převážně ve webové aplikaci *Piskel*. *Piskel* je open-source editor pro pixel art. Umožňuje tvorbu jednoduchých obrázků, animací a jejich import a export v mnohých formátech. Aplikace má jednoduché uživatelské rozhraní a mnohé užitečné klávesové zkratky, čili její ovládání je velice jednoduché a intuitivní.

7.1. Tvorba textur

Tvorba jedné textury trvá (pokud chceme alespoň nějakou kvalitu) od 15 minut do několika hodin. Čas se odvíjí od velikosti a typu. Například zbraň je na tvorbu výrazně snazší, než budova nebo character (viz obr. 13-15).

Časově náročnější jsou pak *animace*. Důvodem je to, že každá animace je tvořena z jednotlivých obrázků, které musí autor postupně vytvořit. Proto tvorba složitějších animací, jako například vlaku (viz projekt samotný), trvá i několik hodin.

Další složitou záležitostí při tvorbě vzhledu hry je např. *textura země*. Pokud chce autor mít „na pohled hezky vypadající výsledek“, musí svůj výtvor optimalizovat pro spojení dohromady z toho důvodu, že jednotlivě vykreslovat každý pixel velkého herního plánu by byl úkol skoro lidsky nemožný. Proto se většinou používá *technika dlaždic*. Autor vytvoří texturu pro jednu, či více dlaždic, ze kterých následně poskládá celou mapu. Výhodou tohoto způsobu je následné usnadnění programování projektu. Hra totiž potom může fungovat trochu více jako např. šachy. Další výhodou tohoto stylu je možné využití nástroje, jako např. *Tiled*. Tento nástroj umožňuje tvorbu map, jejich vrstev, objektů a jednoduchého vložení do projektu.



Obr. 13

Obr. 14

Obr. 15

8. ZÁVĚR

Díky této ročníkové práci jsme si procvičili práci v Javě a naučili se spoustu nových věcí o tom, jak se tvoří hry. Také jsme se naučili pracovat s nástroji pro grafiku a tvorbu textur.

K této hře bychom se v budoucnu chtěli ještě vrátit, dodělat věci, které jsme nestihli a určitě ji ještě vylepšit.

9. ZDROJE:

1. *libGDX. Dostupné z URL:* <https://libgdx.badlogicgames.com/> *(navštíveno 7.3.2019)*
2. *Objects/Bodies.* Dostupné z URL: <https://github.com/libgdx/libgdx/wiki/box2d#objectsbodies/> (navštíveno 8.5.2019)
3. *Full LibGDX Game Tutorial – Box2D Body Factory* Dostupné z URL: <https://www.gamedevelopment.blog/full-libgdx-game-tutorial-box2d-body-factory/> (navštíveno 25.4.2019)
4. *Event driven keyboard and mouse handling.* Dostupné z URL: <https://www.gamefromscratch.com/post/2013/10/15/LibGDX-Tutorial-4-Handling-the-mouse-and-keyboard.aspx/> (navštíveno 4.5.2019)
5. *Full LibGDX Game Tutorial – Input Controller.* Dostupné z URL: <https://www.gamedevelopment.blog/full-libgdx-game-tutorial-input-controller/> (navštíveno 25.4.2019)
6. *Full LibGDX Game Tutorial – Rendering and Asset manager.* Dostupné z URL: <https://www.gamedevelopment.blog/full-libgdx-game-tutorial-rendering-asset-manager/> (navštíveno 1.5.2019)
7. *Full LibGDX Game Tutorial – Loading Screen.* Dostupné z URL: <https://www.gamedevelopment.blog/full-libgdx-game-tutorial-loading-screen/> (navštíveno 1.5.2019)
8. *Firing bullet from gun's current position to mouse x & y position.* Dostupné z URL: <http://www.java-gaming.org/topics/firing-bullet-from-gun-s-current-position-to-mouse-x-y-position/35020/view.html> (navštíveno 5.5.2019)
   Shooting bullet towards mouse position in Java/Slick2D. Dostupné z URL: <https://stackoverflow.com/questions/34122380/shooting-bullet-towards-mouse-position-in-java-slick2d> (navštíveno 5.5.2019)
9. Brent Aureli. *Demo Multiplayer LibGDX game using a NodeJS server and Socket.IO for event firing.* Dostupné z URL: <https://github.com/BrentAureli/MultiplayerDemo> (navštíveno 8.5.2019)
10. *Socket.IO-client Java.* Dostupné z URL: <https://github.com/socketio/socket.io-client-java> (navštíveno 3.5.2019)

OBRÁZKY:

- *Obr. 6.* Dostupný z URL: <https://github.com/libgdx/libgdx/wiki/Event-handling>