Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Mobilní hra typu Tower defense pro Android

Plzeň 2016 Pavel Brzák (A12B0021P)

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 22. června 2016

Pavel Brzák

Poděkování

Rád bych touto cestou poděkoval vedoucímu své bakalářské práce panu Ing. Ladislavu Pešičkovi za vstřícný přístup, odbornou konzultaci a metodické vedení, které se staly podkladem pro vypracování této práce.

Abstract

Abstrakt

Tato bakalářská práce se zabývá programováním herních aplikací do chytrých mobilních telefonů, které běží na platformě Android. Zaměřuje se na herní typ zvaný „Tower Defense“, v češtině známý pod pojmem „věžovky“. Práce popisuje jednotlivé prvky, které jsou vlastní pro tento typ her a ukazuje několik jejich představitelů. V další části je popsáno jakým způsoben a jakými prostředky jsou implementovány jednotlivé herní prvky (animace, herní smyčka, kolize atd.). Následuje implementace mé vlastní hry typu „Tower Defense“ pro platformu Android.

Abstract

Bachelor thesis is based on programming game applications for the smart phones, which are running on android. It is focused of type of game called ‘Tower Defense’. This thesis describing individual elements, which are typical for this game type. Also shows several of their representatives. The next section describes the ways and instrument which are used for implementing individual game elements (animation, game loop, collision etc.). The last part is my own implementation of ‘Tower Defense’ game for Android platform.

Obsah

1 Úvod

Cílem mé práce je vytvořit mobilní hru typu Tower defense, která bude určena pro platformu Android. Hra by měla být spustitelná na různých typech moderních telefonů s rozdílnými verzemi operačního systému Android. Prvním krokem tedy bude seznámení s platformou Android a programováním her pro tuto platformu. Dále potom prozkoumání některých implementací her tohoto typu pro PC i pro mobilní platformu Android a navržení vhodného scénáře pro vlastní hru, s rozšířeními této koncepce o další logické prvky jako, rozhodování útočících protivníků nebo sledování vynaloženého úsilí pro splnění herní úrovně. Z typu hry, Tower defense, tedy plyne, že uživatel bude na mapě stavět různé typy věží, aby zabránil průniku útočících jednotek.

Jedním z cílů návrhu hry je příjemná 2D grafika a uživatelsky přívětivé prostředí. K ovládání budou použity dotykové senzory. Hra by měla mít více než 10 úrovní. S postupem hrou se bude zvyšovat obtížnost, silnější protivníci, obtížnější mapy, méně zdrojů a času pro budování obrany apod.

2 Hry typu Tower defense

Jedná se o strategické hry. Tower defense v doslovném překladu znamená „věžová obrana“, v češtině se ujalo značení „věžovky“. Hráč staví věže, popřípadě jiné jednotky, které čelí nepřátelským útokům a snaží se je zastavit dříve, než projdou bludištěm mapy z bodu A do bodu B. Nepřátelské jednotky obvykle přicházejí ve vlnách. [2]

Hráč na začátku hry získá obnos peněz, za které se kupují věže. Peníze se přičítají za zabité nepřátelské jednotky a rychlost zničení útočící vlny. Každá další úroveň přináší vyšší obtížnost, je tedy nutné stavět silnější věže, nebo vylepšovat ty stávající. Nepřátelé chodí po vyznačených cestách, které se mohou rozdělit a před cílem se spojit. Nepřátelé mají pevně definované, jakou cestou se dají. Jedním z rozšíření v mé hře bude schopnost nepřátel rozhodovat se, jaká cesta je výhodnější, například z hlediska menší obrany dané cesty hráčem. Pro lepší orientaci je dále uveden slovník základních pojmů:

- **Herní úrovně** jsou jednotlivá herní kola, která se jejich plněním postupně otevírají.
- **Mapa** je prostředí v každé herní úrovni.
- **Cesta** je úsek na mapě, kudy chodí nepřátelé z místa A do místa B, čemuž se snažíme zabránit.
- **Věž** slouží k obraně cesty a ničení nepřátel. Na cestě se věže nedají stavět.
- **Nepřítel** je stvoření nebo objekt snažící se projít cestou.
- **Hrdina** je přátelská jednotka, která má speciální schopnosti a může se postupně zlepšovat.
- **Boss** je nepřátelská jednotka, která je mnohem silnější než obyčejný nepřítel. Obvykle má speciální útok a přichází na konci některých herních úrovní.
- **Útok ve vlnách** je seskupení nepřátel, které jdou cestou pohromadě. Nepřátelé ve vlnách jsou postupně početnější, silnější a odolnější. Každá herní úroveň má několik útočících vln.

Dále se podíváme na několik vybraných her. Jsou vybrány na základě hodnocení a především různorodosti herních mechanismů.

2.1 Kingdom Rush

Tato hra je dílem Ironhide Game Studio, vydána jako volně přístupná hra v prostředí webového prohlížeče v půlce roku 2011, na konci roku potom na iPad a na jaře 2013 i na Android. Hra se dočkala pokračování v létě 2013 s názvem Kingdom Rush: Frontiers a v listopadu 2014 prequel (děj odehrávající se před původní hrou) hry Kingdom Rush: Origins. Později byla pro Android vydá nová verze s dalšími úrovněmi a s možností koupit si hrdinu, nebo další vylepšení. Hrdinové mohou být umístěni kdekoli na cestě. Základní hra Kingdom Rush je bezplatná pro všechny platformy, pokračování jsou na Androidu a iOS placená. [5]

Obrázek 2.1: Ukázka hry Kingdom Rush

Obrázek 2.2: Výběr úrovně v Kingdom Rush

Kingdom Rush je ztvárněna jako středověké fantasy. Hráč může stavět věže na předem definovaná místa (viz Obrázek 2.1). Na výběr jsou čtyři typy věží, věž se střelci, magická věž, věž s dělem a kasárny, z kterých chodí vojáci a brání cestu. Na začátku úrovně má hráč určitý obnos peněz ke koupení věží, za zabití nepřátel nebo za přivolání nepřátelské vlny dřív získává další peníze. V průběhu hry tyto peníze použije k nákupu dalších věží, či k vylepšení těch stávajících. Každý typ věže je dobrý na jiný typ útočících vojáků. Celkem má hra 18 úrovní a ve hře je 48 různých typů nepřátel. Za dobrý průchod úrovní (rychlost, nikdo z nepřátel neprošel) jsou různé bonusy v podobě možnosti nákupu vylepšení rychlosti střelby, ceny věží apod. Tato vylepšení jsou trvalá.

Výběr herní úrovně se provádí pomocí mapy, kde se postupně otevírají nová kola (viz Obrázek 2.2). [6]

2.2 Frontline Defense

Tato hra je od společnosti Max Games. Je to volně hratelná hra v prostředí webového prohlížeče, nebo bezplatná hra pro Android. Cílem je vydržet určitý počet nepřátelských vln, každá následující vlna je silnější. Útočící nepřátelé mají určitý počet životů. Pokud se nám je nepodaří zničit a projdou celou mapou, odečítají se jejich životy od našeho. Když náš život klesne pod nulu, úroveň není splněná. [4]

V této hře stavíte vojenské strážní věže, například: věž s vojákem s puškou, s brokovnicí, nebo s kulometem, nebo věž vysílající blesky. Dále podpůrné věže, které dokáží zvýšit dosah nebo útočnou sílu ostatních věží v dosahu. Celkem je na výběr 12 různých typů, které se navíc dají několikrát vylepšit (viz Obrázek 2.3). Výběr herní úrovně je jako v Kingdom Rush realizován pomocí mapy, kde se postupně odemykají další kola (viz Obrázek 2.4).

Obrázek 2.3: Ukázka hry Frontline Defense

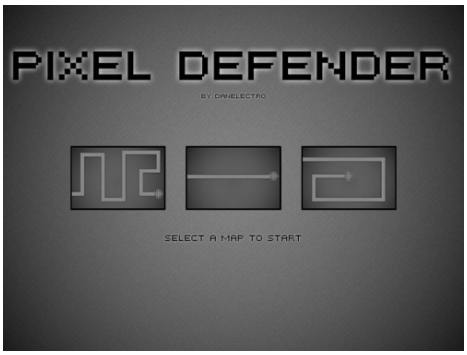
Obrázek 2.4: Výběr úrovně ve Frontline Defense

2.3 Pixel Defender

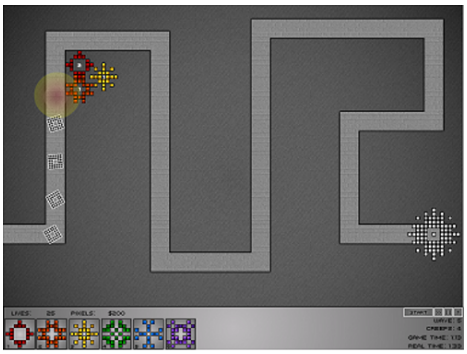
Tato hra je volně dostupná hra v prostředí webového prohlížeče, ztvárněná ve velice jednoduché grafice. Verze pro Android, nebo iOS neexistují. Na

začátku si hráč vybere jednu mapu, na které se potom odehrávají všechna kola (viz Obrázek 2.5).

Úkolem je zabránit útočícím pixelům v průchodu. K tomu slouží 6 jedinečných věží, z kterých mohou být bonusy, pokud jsou postaveny vhodně vedle sebe, dotýkají se a pokud jsou vylepšeny na maximální pátou úroveň. S každým kolem jsou útočící pixely větší a silnější. Takže je nutné nad rozmístěním věží a jejich kombinací opravdu přemýšlet (viz Obrázek 2.6). Hra navíc nabízí variantu hry s téměř neomezeným počtem peněz, například pro vyzkoušení herních mechanismů a kombinací věží. [7]



Obrázek 2.5: Výběr úrovně ve hře Pixel Defender



Obrázek 2.6: Ukázka vybrané cesty

2.4 Zhodnocení

Každá z představených her má rozdílný přístup k ovládání, herním mechanismům, či ke grafickému zpracování. Ve hře Kingdom Rush může hráč stavět jen na předem definovaná místa, má k dispozici bonusové útoky a hrdinu. Je zde velmi propracovaná grafika. Hra Frontline Defense nabízí množství podpůrných věží, které jen posilují útočné věže. Další odlišností je odčítání životů v případě průchodu nepřátel, podle jejich zbylých životů. Ve hře Pixel Defender je zajímavá možnost kombinací jednotlivých věží k získání silnějšího útoku. Je zde jednoduchá 2D grafika, která se do hry výborně hodí. Ve hře je možnost zvolit úroveň bez peněžního omezení a zkoušet tak kombinace věží.

U mé hry bych chtěl použít jednodušší grafiku a pohled na mapu shora. Otestovat si věže a strategie bude možné díky editoru map, kde si hráči budou moct nastavit například větší obnos peněz na začátek hry.

3 O platformě Android

Android je jeden z nejrozšířenějších operačních systémů pro chytré telefony a tablety. Je založen na open source platformě, tedy softwaru s otevřeným kódem. [11]

3.1 Používané verze systému Android

Historie operačního systému Android začala v roce 2007 Androidem Alpha. První komerční verze Androidu, Android 1.0, byla vydána společností Google v září 2008. Nejpoužívanější verze systému Android jsou: [12, p. 16] [11]

- Android 4.0 – 4.0.4 (Ice Cream Sandwich).
- Android 4.1 – 4.3.1 (Jelly Bean).
- Android 4.4 – 4.4.4, 4.4W – 4.4W.2 (KitKat).
- Android 5.0 – 5.1.1 (Lollipop).
- Android 6.0 – 6.0.1 (Marshmallow) je nejnovější, představený v říjnu roku 2015.

Nové verze opravují chyby předešlých a přidávají novou funkčnost. Jednotlivé verze se od sebe liší i uživatelským rozhraním. Funguje zde systém úrovní android API, které zajišťují kompatibilitu mezi aplikacemi a přístroji. API úroveň je celé číslo. Novější API jsou kompatibilní se staršími verzemi (většina změn je aditivní). Každá verze platformy Android podporuje jednu úroveň API

Před vývoje aplikace je nutné vybrat verzi platformy Android, respektive úroveň API. Informace o úrovni API je uchována v AndroidManifest.xml (viz 4.3). Dále je potřeba definovat minimální úroveň API, na které bude aplikace bez potíží fungovat. Úroveň API sdělí platformě Android, jaké funkce jsou pro správný běh potřeba. Pokud by se aplikace nainstalovala na na zařízení s operačním systémem Android podporující nižší úroveň API, za běhu aplikace by mohlo docházet k haváriím v případě přístupu k API funkcím, které na

této nižší úrovni neexistují. Systém API tedy dovoluje instalaci aplikace do zařízení jen v případě, že je úroveň API vyšší, nebo rovna minimální úrovni API definované v aplikaci. [12, p. 49]

3.2 Vývojové prostředí a nástroje

Android aplikace bývají převážně naprogramovány v jazyce Java. Uvažoval jsem nad vývojovým prostředím Eclipse s nástroji pro tvorbu aplikací pro Android. Google ale ukončil práce na rozšíření, Android Development Tools, pro Eclipse. [18]

Nakonec jsem zvolil Android Studio. Jde o specializované vývojové prostředí na tvorbu aplikací pro Android. Společnost Google jej uvedla v roce 2013. Prostředí obsahuje vše, co je pro vývoj potřeba. [16]

3.2.1 Frameworky pro vývoj her

Existuje spousta frameworků, které mohou pomoci s tvorbou hry pro platformu Android. Frameworky se starají o optimalizaci hry pro zíkání lepšího výkonu, o fyziku objektů, uživatelské rozhraní a mnoho dalších. Některé frameworky umožňují vývoj her i bez znalosti programování. Jedním z takových nástrojů je například GameMaker. V mojí implemetaci žádný z těchto frameworků nepoužívám. Hlavním důvodem byla snaha o komplexní pochopení základních herních principů a prvků, které bych při použití těchto frameworků nemusel do jisté míry řešit. [9]

- **Corona SDK** je často používaný framework u herních vývojářů. Používá programovací jazyk Lua. Podporuje Android, Windows Phone a iOS. [14]
- **Unity Mobile** je mobilní verze frameworku Unity3D. Na rozdíl od ostatních frameworků podporuje i BlackBerry a Tizen. [17]
- **Emo** je založený na OpenGL ES a OpenGL AS. Tento framework umožňuje tvořit hry pro Android i pro iOS. [15]
- **Edgelib** je multiplatformní framework. Má 2D a 3D herní engine. Podporuje Android, Windows Phone, iOS i Symbian. [8]

4 Základní komponenty aplikace

Aplikace jsou složeny z mnoha různých komponent. Protože hry používají grafické rozhraní, jedná se o aplikace na popředí. K důležitým komponentám patří Aktivity, Fragment, View. [13]

4.1 Aktivity

V každé aplikaci musí být alespoň jedna aktivita. V podstatě se stará o to, aby se všechna data, správně zobrazila uživateli. Aktivita tedy spustí dialogové okno, které zpravidla vyplňuje celou obrazovku, nebo se jedná o tzv. "plovoucí okno". Aplikaci většinou tvoří několik aktivit. [1]

Mezi aktivitami se dá přepínat. Po přepnutí se minulá aktivita uloží do zásobníku, který funguje systémem LIFO (last in first out). Tento mechanismus zaručuje, že se můžeme vrátit z jedné aktivity až na tu hlavní, která je v Android Manifestu (viz kapitola 4.3) definována jako spouštěcí. [12, p. 39]

Aktivity mají svůj životní cyklus, který sestává z těchto metod (viz Obrázek 4.1):

- **onCreate():** Metoda volaná při vytváření aktivity. Protože téměř všechny aktivity interagují s uživatelem, nastaví se uživatelské rozhraní pomocí *setContentView(View)*. V tomto stavu je aktivita zastavená, neviditelná a nepřijímá uživatelský vstup.

  Pro potřeby naší aplikace můžeme například v hlavní aktivitě, *MainActivity* načítat *Shared Preferences*. Nastaví se jazyk hry, hudba na pozadí, do proměnných se nastaví šířka a výška displeje pomocí *DisplayMetrics*.

- **onStart():** Metoda volaná při spuštění aktivity, nebo pokud se k této aktivitě chceme vrátit. Zde by se měly provést všechny úkony, které jsou potřeba pro zobrazení aktivity.

- **onResume():** Metoda, která je volaná, když se aktivity dostává na popředí.

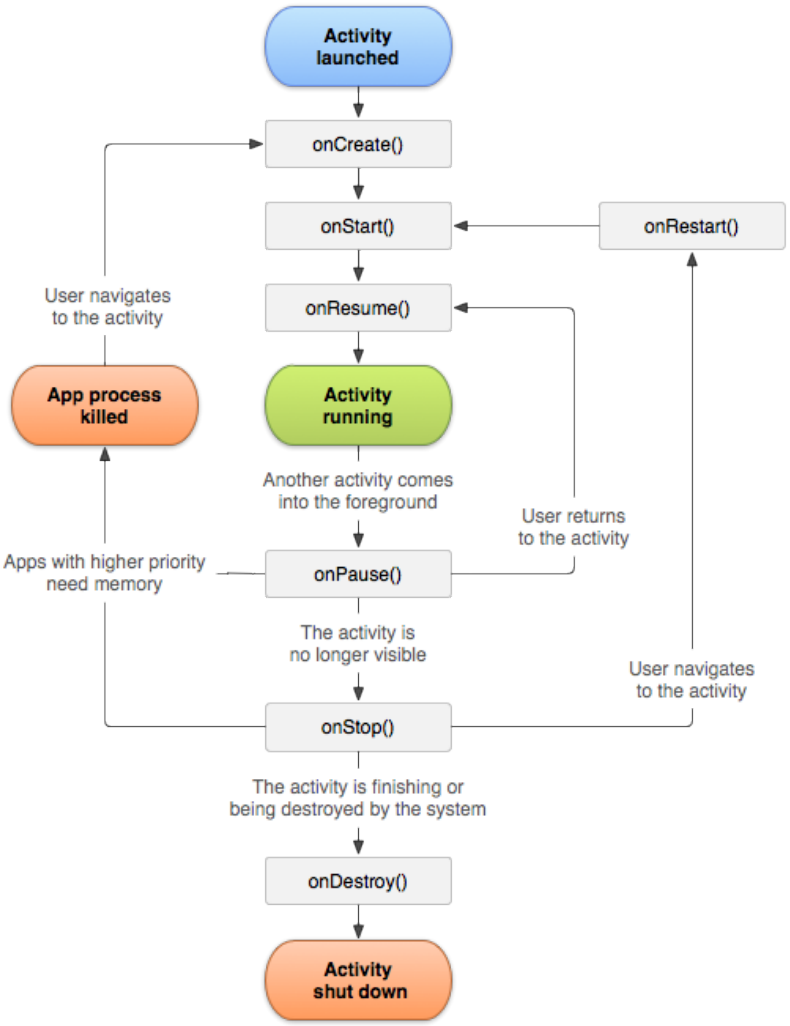
  V této metodě se v hlavní aktivitě zapíná hudba na pozadí.

- **onPause():** Metoda volána, když se uživatel přesune do jiné aktivity. Protiklad k metodě onResume().

  V této metodě se v hlavní aktivitě vypíná hudba na pozadí.

- **onStop():** Protiklad k metodě onStart(). Aktivity z pauzy přechází do stavu zastavená. Zde typicky postaráme o to, aby se zrušilo vše, co jsme v metodě onStart() vytvořili.

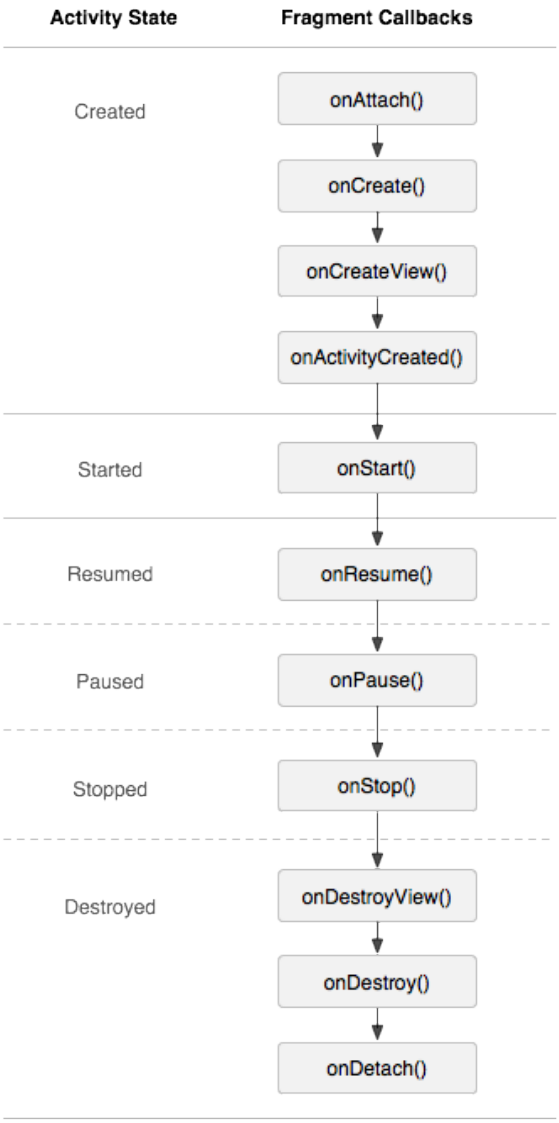
- **onDestroy():** Metoda je volána když aktivita končí, nebo je zničena dočasně, aby bylo uvolněno místo v paměti.



Obrázek 4.1: Životní cyklus Aktivit [1]

4.2 Fragment

Fragmenty existují od API 11, tedy Androidu 3.0. Fragment reprezentuje uživatelské rozhraní nějaké aktivity, nebo jeho část, i s příslušnými metodami. Rozdílem je, že fragment se může stát součástí jiné aktivity. Tím se dají vytvářet znovupoužitelné, flexibilnější prvky uživatelského rozhraní. Bez duplikace kódu jsme schopni vytvářet optimalizovaná prostředí i pro větší displeje, jako tablety apod. [3]



Obrázek 4.2: Životní cyklus fragmentů [3]

Fragment má navíc k metodám životního cyklu aktvity přidané některé metody. (viz Obrázek 4.2):

- **onAttach():** Tato metoda se volá, když je fragment spojen s nějakou aktivitou, která mu je předána jako argument. Volá se před metodou onCreate(). Provádíme zde kontrolu, jestli aktivita implementuje požadované rozhraní.
- **onCreateView():** Volá se po metodě onCreate() a obvykle vrací View, které bude v uživatelském rozhraní fragment reprezentovat.
- **onActivityCreated():** Metoda volaná po onCreateView(), tehdy když skončilo onCreate() v aktivitě. V případě, že byl fragment přidán v kódu dodatečně a onCreate() v aktivitě už doběhlo, zavolá se hned po onCreateView().
- **onDestroyView():** Metoda je volána po onStop(). Zde se odstraní zdroje spojené s tímto view, které byly vytvořeny v metodě *onCreateView()*.
- **onDetach():** Volá se po onDestroy(). Je to protiklad k metodě onAttach().

4.3 Android Manifest

Jde o XML soubor, který operačnímu systému sděluje, které komponenty jsou k dispozici. Specifikuje parametry aplikace, jako: název, verze, ikonu (viz Obrázek 4.3) apod. Definuje atributy aplikace (šířka a výška layoutu atd.). Můžeme zde přidělovat aplikaci různá oprávnění, například: přístup k fotoaparátu, kameře. [12, p. 45]

Obrázek 4.3: Ikona aplikace

```
<activity
     android:name=".MainActivity"
     android:configChanges="orientation|keyboardHidden
              |screenSize|locale"
     android:theme="@style/Theme.AppCompat"
     android:screenOrientation="landscape">
     <intent-filter>
         <action android:name="android.intent.action.MAIN" />
         <category android:name=
                "android.intent.category.LAUNCHER" />
     </intent-filter>
</activity>
```

Ukázka zdrojového kódu 4.4: Android Manifest spouštěcí aktivita

5 Analýza

Cílem mojí práce bylo udělat hru typu Tower defense (viz kapitola 2). Hra by měla být ovládána dotykovou obrazovkou. Herní mapa bude na celou obrazovku, bez nutnosti posouvání, pro co nejlepší přehlednost o herní situaci pro hráče. Mapa bude rozdělena do mřížky a bude záležet na hráči, kam si postaví obrané věže. Věže půjde postavit kliknutím na zvolenou věž. Informace o věžích a protivnících budou k dispozici v hlavním menu hry, aby se hráč mohl seznámit se statistikami i mimo hru.

Kromě encyklopedie herních prvků by v menu měl mít hráč možnost podívat se na dosažené skóre v jednotlivých úrovních, nastavit jazyk hry, vypnout hudbu a zvuky, nebo vypnout animace pro snížení náročnosti aplikace na méně výkonných zařízeních.

Hra by měla mít přes 10 herních úrovní s různými typy nepřátel a s několika druhy prostředí map. Několik různých druhů obraných věží, například: běžné střely, ochromující šipky, nebo plošné útoky. Obdobně budou specifikovány útočící jednotky, kde některé budou pomalejší, ale vydrží více zásahů. Útočící jednotky by měly mít určitou míru inteligence. Budou se rozhodovat, jaká cesta pro průchod mapou je pro ně nejlepší, podle toho, jak si hráč postavil věže. Nepřítel ale neuvidí na celou mapu, jen v určitém okruhu kolem sebe a informaci o postavených věžích bude sdílet s ostatními útočníky.

V rámci aplikace bude vytvořen editor úrovní, kde bude možné poskládat si vlastní mapu. Dále je potřeba definovat jaké věže bude možné postavit, jaké nepřátelské jednotky se zde budou moci objevit, nebo zadat počáteční množství peněz hráče.

5.1 Hudba a zvuky ve hře

Hudba hrající na pozadí a zvukové efekty budou využity s licencí pro volné použití. Tyto soubory jsou uloženy ve složce aplikace raw. Pro přehrávání hudby na pozadí bude použita třída *MediaPlayer*. Přehrávač se vytváří metodou *create(Context context, int resid)*. Context je rozhraní pro globální informace o aplikačním prostředí. Tuto abstraktní třídy zajišťuje systém Android. Resid je ID zvukového souboru získané z *R.raw.<název bez přípony>*.

V přehrávači poté spouštíme hudbu metodou *mediaPlayer.start()*, nebo vypínáme metodou *mediaPlayer.stop()*. Opakovaní písničky po jejím dohrání zajistí *mediaPlayer.setLooping(true)*.

5.2 Tvorba textur

Textury použité v mojí hře, jsem tvořil sám. Pro tento účel jsem si vybral program pro vektorovou grafiku Inkscape, protože s ním již mám zkušenosti a považuji jej za kvalitní nástroj. Pro další případné úpravy jsem použil program Gimp. Celkem jsem vytvořil přes 50 textur použitých v menu, nebo ve hře.

5.3 Vykreslování objektů

Objekty se dají vykreslovat dvěma způsoby. Buď použitím View, nebo SurfaceView.

5.3.1 View

Jedná se o elementární stavební jednotku uživatelského rozhraní naší vytvářené aplikace. Používá se pro vykreslování objektů a odchytávání událostí. V případě akční hry je nepoužitelný, protože metoda pro vykreslování je volána na jednom vlákně s metodami pro uživatelské interakce. Pro potřeby akční hry je nutné mít oddělené vlákno pro vykreslování, aby nezpomalovalo ostatní činnosti aplikace. *View* bylo použito pro editor map (viz kapitola B.6).

5.3.2 SurfaceView

SurfaceView je potomek třídy *View*. Na rozdíl od svého předchůdce už poskytuje možnost využití více vláken. Vlákna se vytvářejí standardními konstrukcemi. Obsluha je zajištěna pomocí třídy *SurfaceHolder*, kterou dostaneme voláním metody *getHolder()*. Pro vykreslování je použita metoda *on-*

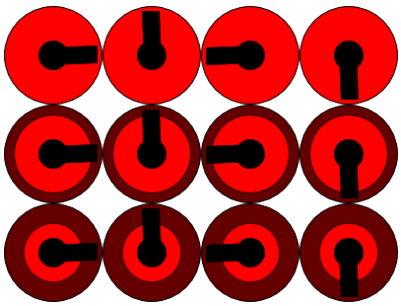
Draw(Canvas canvas). Canvas je třída reprezentující plátno na které můžeme kreslit. Pro vykreslování je nutné v *SurfaceHolder* nastavit zpětné volání, které sloužící k nastavení následujících tří základních metod.

- *surfaceDestroyed (SurfaceHolder holder):* metoda volána při zničení plátna.
- *surfaceChanged (SurfaceHolder holder, int format, int width, int heigh):* metoda volána při strukturální změně, tedy při změně velikosti, nebo při prvním volání.
- *surfaceCreated(SurfaceHolder holder) :* metoda volána při prvním spuštění.

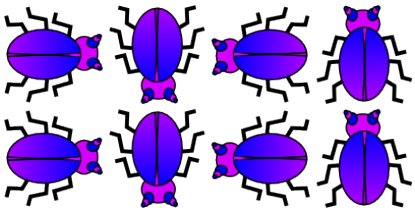
Před začátkem vykreslování na plátno je potřeba dané plátno uzamknout pomocí *SurfaceHolder* a po dokončení volání metody *onDraw()* opět odemknout.

5.3.3 Sprite

Textury (částečně transparentní 2D rastrové obrázky) herních objektů, kde jsou jejich možné polohy seřazeny vedle sebe do mřížky v jednom obrázku nazýváme Sprites. Animace se poté provádí vykreslováním jen určité části Sprites, jedné polohy objektu. Z celkového počtu sloupečků a řádků spočítáme výšku a šířku obrázku. Máme tedy pozici označenou číslem řádku a sloupečku, jejich změnou vykreslujeme i jinou polohu objektu. (viz Obrázek 5.1, 5.2, Ukázka 5.3)



Obrázek 5.1: Sprites věže



Obrázek 5.2: Sprites nepřítele

```
@Override
    public void onDraw(Canvas canvas) {
        int srcX = animC*getWidth();
        int srcY = animR*getHeight();
        Rect src = new Rect(srcX, srcY, srcX + getWidth(),
                                srcY + getHeight());
        Rect dst = new Rect(x, y, x + getWidth(),
                                y + getHeight());

        canvas.drawBitmap(texture, src , dst, null);
    }
```

Ukázka zdrojového kódu 5.3: Vykreslení sprite

5.3.4 Mapa pozadí

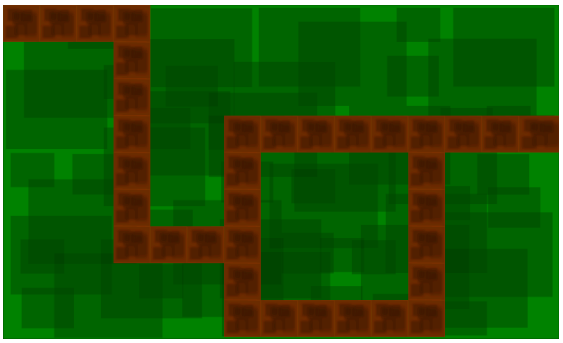
Displej je nutné si rozdělit mřížkou do buněk. Pro tuto hru jsem použil 15 buněk na šířku a 9 na výšku. Každý displej bude mít jinak velké buňky, ale bude jich stejný počet. Velikost textur potom můžeme jednoduše přepočítávat a získat stejně velké a nezkreslené textury v poměru k velikosti displeje.

Textury mapy jsou vykresleny pomocí matice znaků (viz Obrázek 5.4), kde každý znak reprezentuje určitou texturu. Například:

- **0:** běžné pozadí (volná plocha)
- **1:** textura cesty
- **S:** start
- **F:** cíl

5.3.5 Uložení nastavení mapy

Nastavení pro každou mapu, pro každý herní level, se ukládá do samostatného textového souboru. Tyto soubory jsou uloženy v interní paměti telefonu, odkud je pomocí třídy *InputStreamReader* načítám. Struktura souboru je velmi

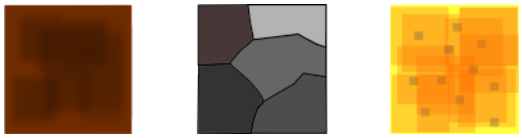


```
S11100000000000
000100000000000
000100000000000
00010011111111F
000100100001000
000100100001000
000111100001000
000000100001000
000000111111000
```

Obrázek 5.4: Mapa herní úrovně s odpovídající reprezentací

jednoduchá, skládá se z několika řádků, na kterých jsou jednotlivé hodnoty. Je zde uložena textura pozadí a cesty (viz Obrázek 5.5), počet nepřátelských vln a jejich časový rozestup, povolené věže pro tento level a počáteční kapitál hráče.

Tato data by se dala uchovávat například ve struktuře XML, nebo JSON, kde je obsah strukturovaný a dá se lépe prohledávat. Ukládání do obyčejného textového souboru jsem zvolil kvůli malému počtu ukládaných parametrů a jednoduchému načítání, které obdobně používám u načítání mapy.



Obrázek 5.5: Textury cesty

5.4 Threads

Pro vykreslování hry bude použit *SufaceView* (viz 5.3.2. Umožňuje použít sekundární vlákno. Nejprve je nutné vytvořit nové vlákno, které se spouští v metodě *surfaceCreated* pomocí *thread.start()*. Při přerušení se vlákno zastaví metodou *surfaceDestroyed*, *voláním thread.join()* čekáme na dokončení vlákna.

Vlákno lze na určitou dobu uspat, čímž můžeme ovlivnit, kolikrát za sekundu proběhne. Určujeme tím tedy FPS, počet snímků za sekundu.

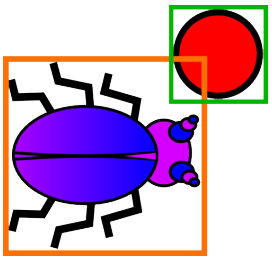
V metodě *run()* vytvořeného vlákna zamkneme *Canvas*, plátno vykreslíme voláním metody *onDraw()* a opět odemkneme.

5.5 Kolize objektů

K zjištění kolize objektů lze použít několik metod, které se liší jak svou přesností, tak náročností. Pokud by byla potřeba velká přesnost, v úvahu by připadala metoda Per-Pixel Collision. V této metodě se porovnávají jednotlivé pixely dvou oblastí. Pokud jsou pixely na stejném místě netransparentní, stav je vyhodnocen jako kolize. Tento postup je zbytečně náročný na chod hry, kde bude stačit méně přesná metoda Bounding Rectangle Collision.

V algoritmu Bounding Rectangle Collision se porovnávají dva obdélníky, které jsou kolem potencionálně kolidujících objektů (viz Obrázek 5.6). Pokud se překrývají, je vyhodnocena kolize metodou *rectangle1.intersect(rectangle2)*.

Tímto způsobem je řešena kolize nepřátel se střelami i zjišťování objektů v okolí nepřátel (cesta, věže) pro další postup.



Obrázek 5.6: Bounding Rectangle Collision

5.6 Detekce dotyku

Pro detekci dotyku na obrazovce přepíšeme metodu *onTouchEvent(MotionEvent event)*. Tuto metodu můžeme překrýt ve třídě dědící Aktivity, nebo Fragment. Pro naše potřeby bude nejvhodnější překrýt ji ve třídě *GameView* (viz kapitola 6.7), která dědí *SurfaceView* (viz kapitola 5.3.2).

5.7 Přerušení hry

U aplikací je nutné ošetřit situaci při narušení běhu jinou událostí, například příchozím hovorem.

S tímto problémem pádu aplikace musíme počítat hlavně ve hrách a aplikacích, kde běží více vláken. Pád způsobují tato běžící vlákna při opětovném volání metody *onResume()*. Tato metoda se v *SurfaceHolderu* snaží znovu spustit vlákno, o kterém si systém myslí, že stále běží.

SurfaceHolderu se při jeho vytváření nastavují metody *surfaceCreated* a *surfaceDestroyed*. Tato metoda není volána při každém typu přerušení. Pro bezproblémový chod je nutné použít metodu, která je volána vždy. Tuto podmínku splňuje metoda *onPause()*, v ní můžeme přerušit hru a vypnout vlákna.

5.8 Uložení nastavení a pokroku ve hře

Pro ukládání hodnot po vypnutí aplikace je na výběr z několika přístupů. Jednou možností je ukládání do textového souboru. Elegantnějším a v tomto případě výhodnějším řešením bude použití třídy *SharedPreferences*, která používá formát klíč-hodnota pro uložením jednoduchých dat do vnitřního úložiště aplikace. Přístup k nim bude jednodušší, než při použití textového souboru.

Totuto cestou se ve hře například ukládá příznak, jestli má být zapnutá hudba, zvuky, animace věží, jazyková lokalizace a herní level, který má hráč ještě zamčený.

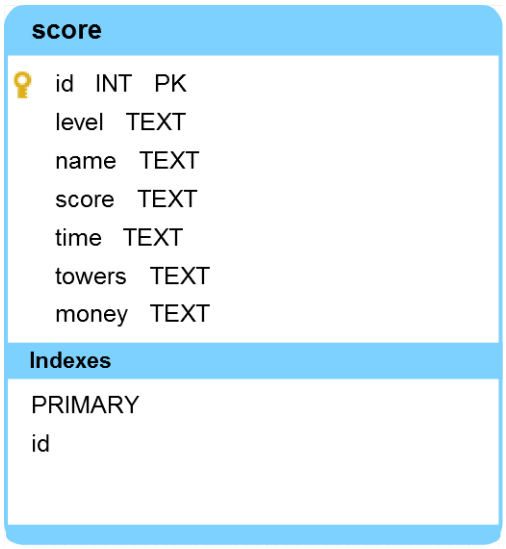
5.9 Uložení skóre a vynaložených prostředků

Na konci vítězné hry je potřeba uložit kromě názvu odehrané úrovně dále jméno hráce, počet získaných bodů a údaje o koupi a prodeji věží s časovým údajem (viz Obrázek 5.7).

V úvahu by připadlo ukládání dat do *XML*. Následná práce s menším

objemem dat je potom rychlejší a velmi výhodná, pokud potřebujeme stromovou strukturu. Je ale možné, že v některých případech bude dat více a vyhledávání jednoho konkrétního prvku také neupotřebíme.

Pro ukládání skóre hry a vynaložených prostředků pro splnění úrovně jsem zvolil *SQLiteDatabase*. Největší výhodou je, že nativně poskytuje zobrazení dat v konkrétním pořadí.

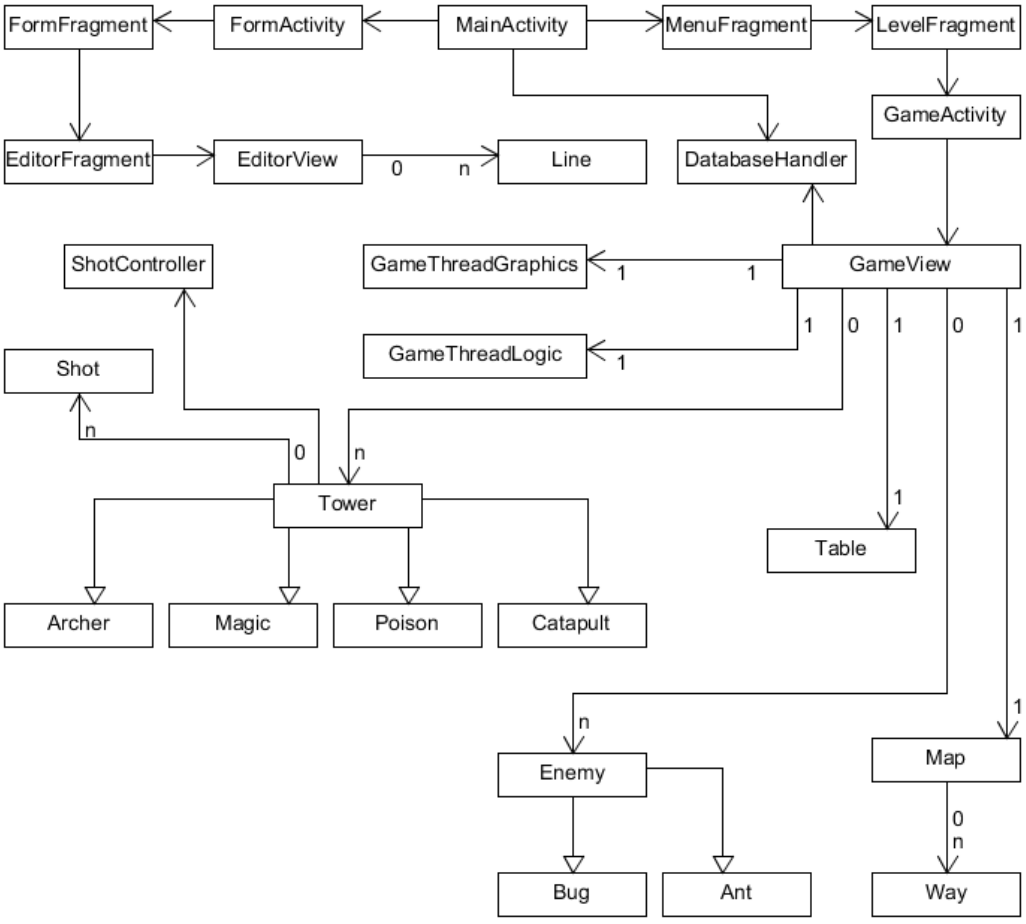


Obrázek 5.7: ERA model databáze

6 Programátorská dokumentace

Aplikace se skládá ze tří aktivit. Hlavní aktivita, která se spustí po zapnutí aplikace je pro hlavní menu (viz Ukázka 4.4), druhá pro samotnou hru a třetí pro editor levelů (viz kapitola B.6). Každá aktivita, která se v aplikaci používá, musí být uvedena v Android Manifestu. Pro větší přehlednost je uveden UML diagram nejvýznamějších tříd (viz Obrázek 6.1).

Všechny třídy jsou popsány v Javadoc dokumentaci, kterou lze nalézt na přiloženém CD ve složce Javadoc. Následuje popis nejvýznamnějších tříd.



Obrázek 6.1: UML diagram nejvýznamnějších tříd

6.1 MainActivity

Je to hlavní aktivita, která pracuje s několika fragmenty a dalšími aktivitami *FormActivity* (viz kapitola B.6) a *GameActivity*. Mezi fragmenty přepíná pomocí třídy *FragmentManager*, která jednotlivé fragmenty ukládá do zásobníku, odkud je následně může jednoduše přepínat a vrátit se až na první. V této aktivitě se inicializuje SharedPreferences a následně se zjišťují jednotlivá nastavení, například jestli má hrát hudba na pozadí pomocí třídy MediaPlayer. Dále se inicializuje databáze pro správu skóre, nebo se vytváří externí složka pro import a export map, pokud již neexistuje.

FragmentManager spouští první fragment, tedy *MenuFragment*.

6.2 Hlavní menu (MenuFragment)

Fragment zobrazující hlavní menu hry (viz Obrázek B.2. Z této obrazovky se můžeme dostat do všech částí aplikace: výběr levelů, nejlepší skóre a vynaložené prostředky, průvodce hrou, editor levelů, nastavení, o programu. Dále zde můžeme importovat (viz Obrázek B.3) a exportovat uživateli vytvořené herní mapy, nebo aplikaci ukončit.

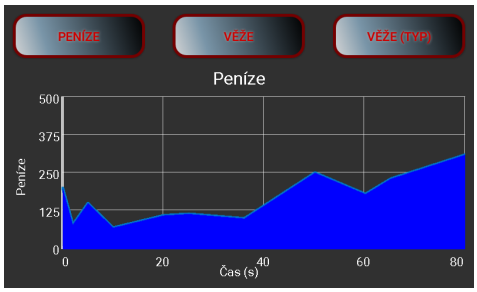
Obrázek 6.2: Hlavní menu

Obrázek 6.3: Import map do aplikace

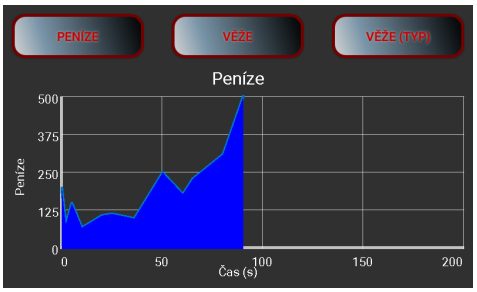
6.3 Skóre

Jedná se o *ListFragment*, který zobrazuje jednotlivé herní úrovně podobně jako při výběru úrovně před začátkem hry (viz kapitola B.4). Po otevření

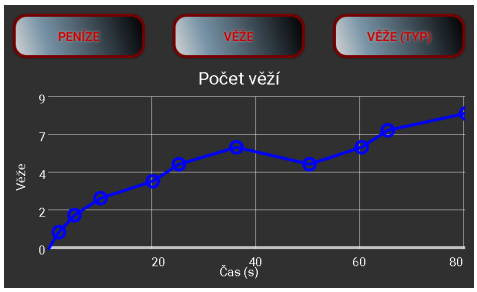
jedné úrovně se zobrazí další *ListFragment* s dosaženými nejlepšími skóre vybrané úrovně. Skóre se počítá jako součet peněz na konci hry, hodnoty věží při prodeji a počtu zbylých životů vynásobených koeficientem. Pokud chceme vidět podrobněji průběh konkrétní hry, kliknutím zobrazíme grafy. První graf ukazuje hospodaření s herní měnou v průběhu hry (viz Obrázek B.4). Grafy se navíc dají scrollovat a oddalovat (viz Obrázek B.5). Druhý graf zobrazuje nákup a prodej věží (viz Obrázek B.6), ve třetím grafu je počet věží vidět pro každý typ zvlášť (viz Obrázek B.7). Grafy jsou vytvořeny pomocí *GraphView*.



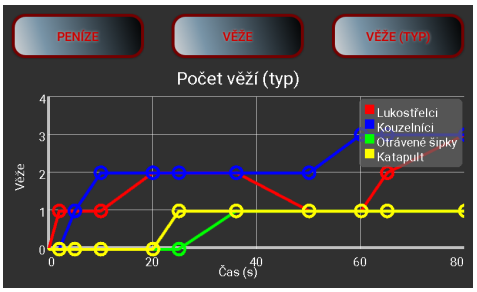
Obrázek 6.4: Nárůst peněz



Obrázek 6.5: Nárůst peněz, oddálení grafu



Obrázek 6.6: Nákup věží



Obrázek 6.7: Nákup věží podle typu

6.3.1 GraphView

Volně použitelný nástroj pro tvorbu grafů. Po seznámení s jeho funkcionalitou jde o jednoduchý a velmi šikovný nástroj. Dají se vytvářet různé typy grafů, včetně realtime grafů. Pokud tento nástroj chceme používat, je nutné do bloku *Dependencies* v souboru *build.gradle* vložit řádek *compile 'com.jjoe64:graphview:4.0.1'*. Alternativou k tomuto kroku je stažení a vložení .jar souboru do projektu, nebo použití *GraphView* zdrojových kódů jako submodulu. [10]

6.4 Průvodce (GuideFragment)

V tomto fragmentu jsou popsány parametry nepřátel a věží, jejich útočná síla v jednotlivých úrovních.

6.5 Nastavení (OptionFragment)

V této části si uživatel může přizpůsobit hru. Lze přepínat jazyk hry, vypnout a zapnout hudbu, zvuky a vibrace. Další možností je vypnutí animací věží ve hře, tímto způsobem můžeme hru zpřístupnit méně výkonným zařízením.

6.6 Výběr levelů (LevelFragment)

Jedná se o *ListFragment*, který zobrazuje jednotlivé herní úrovně pomocí třídy *LevelAdapter*, která dědí od *ArrayAdapter* a zobrazuje prvky třídy *LevelItem*. Po vybrání herní úrovně se spustí nová aktivita *GameActivity*, která spouští a zobrazí kreslicí plátno *GameView* s příslušnou herní mapou a nastavením. Poté začíná vlastní hra.

6.7 GameView

Tato třída se stará o celkovou logiku a vykreslování hry. Nejdříve načte všechny textury a provede jejich změnu velikosti. Nové hodnoty se spočítají podle šířky a výšky displeje telefonu. Tím je zaručen stejný vzhled na různých rozlišeních displeje. Dále se načte herní mapa (viz kapitola 6.8) a příslušné nastavení. Zobrazuje aktuální herní scénu voláním metod *update()* a *onDraw(Canvas canvas)*. Je zde spuštění a zastavení herní smyčky. Třída obsahuje *LinkedList<>* pro aktuálně vytvořené věže i nepřátele. Dále jsou popsány nejdůležitější metody.

6.7.1 update()

Voláji se zde metody update() všech právě používaných herních prvků. Navíc se v případě potřeby odebírají nepřátelé z příslušného *LinkedList<>*. V poslední řadě se kontroluje stav hry, jestli se nemá ukončit. Důvodem pro konec hry může být ubránění cesty proti nepřátelským vlnám, nebo naopak nezabránění v průchodu většímu počtu nepřátel, než je pro danou mapu povoleno.

6.7.2 onDraw(Canvas canvas)

V této metodě se volají všechny metody *onDraw(Canvas canvas)* všech právě používaných herních prvků. Parametr *canvas* označuje instanci plátna, na které se herní objekty vykreslí.

6.7.3 startTimer()

Metoda je volána při spuštění herní smyčky. Jsou zde vytvořeny dva *CountDownTimery*. První časovač spouští nepřátelské vlny a druhý přidává jednotlivé nepřátele.

6.7.4 onTouchEvent(MotionEvent event)

V této metodě je řešena interakce s hráčem. V samotné hře se využivá jen *MotionEvent.ACTION_DOWN*. Pokud hráč dotykem zvolí volnou plochu kolem cesty, zobrazí se tabulka pro výběr věží. V případě, že hráč dytokem vybere již vytvořenou věž, zobrazí se tabulka s parametry věže a s možností věž vylepšit, nebo prodat. Dotykem na texturu pauzy v levém horním rohu, hru pozastavíme (viz kapitola 6.14). Dále se zde obsluhuje dialogové okno po skončení hry.

6.7.5 checkPosition(int x, int y)

Tato metoda vrací stav buňky vzniklé rozdělením herní plochy do mřížky, která obsahuje vstupní bod x, y. Zjišťujeme, jestli je buňka prázdná, nebo na její pozici stojí věž.

6.8 Map

Třída starající se o načtení a správné zobrazení prvků mapy. Nejprve se do matice načte soubor s mapou a poté se podle jednotlivých znaků určí, kde je cesta a její počáteční a koncové souřadnice. Těmito údaji se plní *LinkedList<Way>*. Nakonec se v metodě *onDraw* vykreslí pozadí mapy a v cyklu se volá vykreslovací metoda cesty.

6.9 Way

Třída, která nese informace o textuře a poloze jednotlivých částí cesty. Obsahuje metodu *onDraw*, která se stará o vykreslení textur na dané souřadnice.

6.10 Tower

Třída věží, které hráč staví na obranu cesty před nepřáteli. Tuto třídu dědí třídy jednotlivých typů věží jako *Archers*, *Magic*, *Poison* a *Catapult*. Každá věž má několik parametrů, například texturu pro vykreslování, souřadnice polohy, cenu, aktuální level, působené poškození, vzdálenost dostřelu, rychlost střel, nebo rychlost střelby. Dále obsahuje seznam střel, které věž vystřelila a instanci třídy *shotController* pro zjišťování zásahu nepřátel (viz kapitola 6.15). Třída má tři základní metody *update()*, *updateShot()* a *onDraw()*.

V metodě *update* měníme vykreslovanou část sprite (viz kapitola 5.3.3). Metoda *updateShot* kontroluje přítomnost nepřátel a případně zajišťuje střelbu. Dalším úkolem je kontrolovat kolize střel s nepřáteli pomocí třídy *shotController* (viz kapitola 6.15).

Metoda *onDraw* se stará o vykreslení sprite (viz Ukázka 5.3).

6.11 Enemy

Třída nepřátel, kteří se snaží projít po cestě ze startu do cíle. Tuto třídu dědí třídy pro jednotlivé typy nepřátel jako *Ant*, nebo *Bug*. Každý nepřítel má několik parametrů, například texturu pro vykreslování, souřadnice polohy, pole pro ukládání objektů v okolí, rychlost, počet životů, nebo odměnu za zničení. Třída má dvě základní metody *update()* a *onDraw()*.

V metodě *update* měníme vykreslovanou část sprite (viz kapitola 5.3.3). Zjišťují se aktuální okolní objekty (cesta, věže) pro vyhodnocování dalšího kroku a následně se podle získaných údajů nepřítel posune.

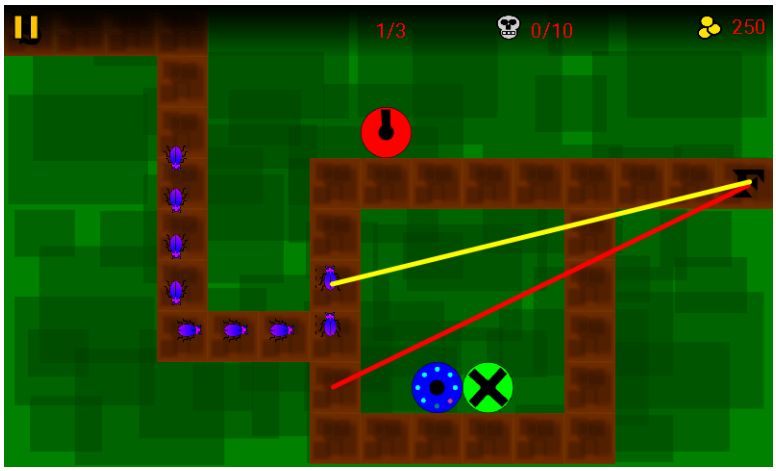
Metoda *onDraw* se stará o vykreslení sprite (viz Ukázka 5.3).

6.12 EnemyCommander

Tato třída "velitele" shromažďuje údaje o mapě v podobě matice od všech nepřátel a dává je ostatním k dispozici. Nepřátelé si aktualizují své mapy podle mapy "velitele" a tím i nově vytvoření nepřátelé mají do jisté míry odkrytou mapu. Na začátku hry vědí nepřátelé jen kde je začátek a konec cesty.

6.12.1 Rozhodnutí podle vzdálenosti k cíli

Každý nepřítel kontroluje své okolí a zjišťuje jaké objekty se v něm vyskytují. Tato kontrola se provádí pomocí obdelníkové kolize (viz kapitola 5.5). Do matice, která představuje mapu, se ukládají znaky zastupující cesty a věže. Podle souřadnic konce cesty se na rozcestích vypočítává vzdálenost k cíli všemi směry, kterými je možné jít. Nepřítel se vydá cestou s nejkratší vypočtenou vzdáleností, protože je pravděpodobné, že tato cesta bude nejrychlejší (viz Obrázek 6.8).



Obrázek 6.8: Rozhodnutí na základě vzdálenosti k cíli

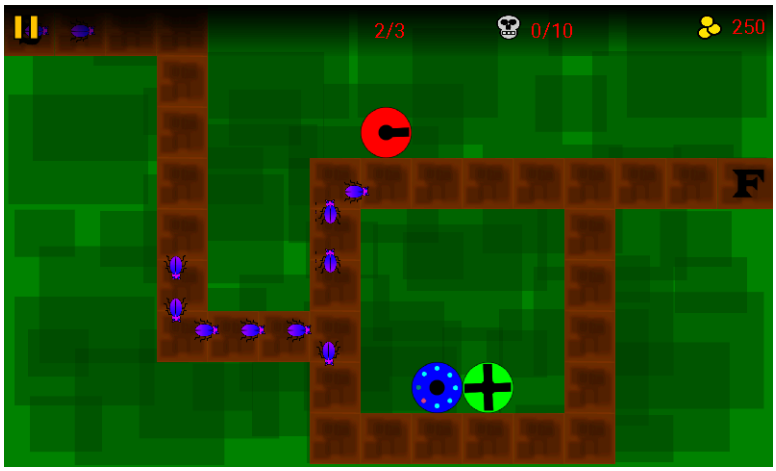
6.12.2 Rozhodnutí podle náročnosti cesty

Pokud se na mapě vyskytuje věž po rozvětvení cesty, nepřítel to při průchodu kolem ní zjistí a předá informaci ostatním. Nepřátelé potom půjdou druhou cestou. Tedy cestou, která je méně náročná, méně nebezpečná (viz Obrázek 6.9). Pokud při průchodu druhou cestou zjistí větší počet věží, je opět vybrána první cesta (viz Obrázek 6.10).

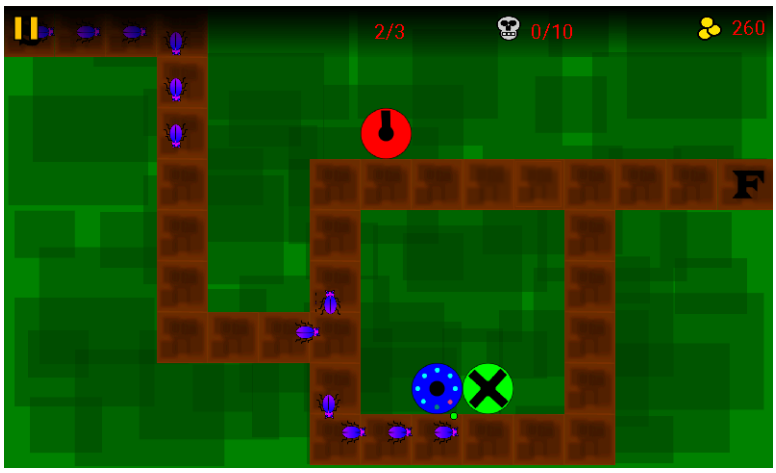
Po příchodu na rozcestí se postupně prohledají všechny směry, kudy se nepřítel může vydat. V cyklu se prohledá matice mapy, od rozcestí se v matici pohybujeme po znacích představující cestu a postupně procházíme jejich okolí, jestli se v něm nevyskytují věže. Cesta s nejmenším ohodnocením (nejmenším počtem věží v okolí cesty, které mohou ohrozit nepřátele při průchodu) je vybrána pro další postup.

6.13 Shot

Třída, která nese informaci o střele, její texturu, polohu, rychlost a id nepřítele, na kterého byla vystřelena. Metoda *update* mění polohu střely v závislosti na rychlosti a poloze zaměřeného nepřítele. Střela se tedy nepohybuje rovně, ale je naváděna (zatáčí) na nepřítele. Metoda *onDraw* střelu vykresluje.



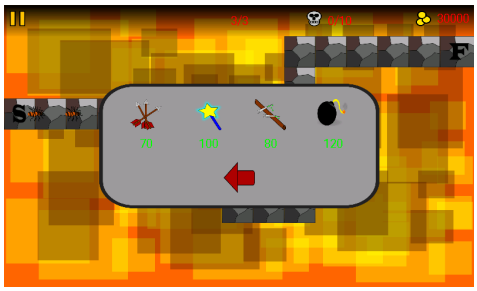
Obrázek 6.9: Rozhodnutí podle počtu věží - objevení první věže



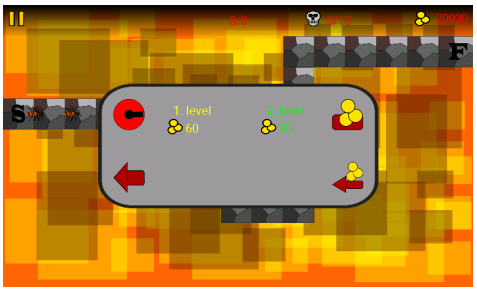
Obrázek 6.10: Rozhodnutí podle počtu věží - zkouška druhé cesty

6.14 Table

Tato třída během hry zobrazuje buď tabulku s výběrem věží ke koupi (viz Obrázek B.8), nebo tabulku pro vylepšení/prodej věže (viz Obrázek B.9), nebo tabulku při pozastavené hře v závislosti na zvolené metodě *onDraw* pro typ tabulky. Metoda *getTouch* potom kontroluje, jestli bylo kliknuto na nějaký prvek tabulky a podle toho vrací do *GameView* informaci o tom, co se má stát.



Obrázek 6.11: Tabulka pro nákup věží



Obrázek 6.12: Tabulka pro vylepšení, nebo prodej věže

6.15 ShotController

Třída starající se o kontrolu kolize mezi nepřáteli a střelami. Kolize jsou posuzovány na základě algoritmu Bounding Rectangle Collision (viz kapitola 5.5).

7 Editor levelů

Jedná se o nástroj pro vytváření vlastních úrovní do hry, navíc je možné si nadefinovat pozadí a další parametry mapy.

Pro tuto sekci je v aplikaci vytvořena nová aktivita *FormActivity*. Aktivita se stará o přehrávání hudby a *FragmentManager* spouští první fragment *FormFragment*.

7.1 Nastavení mapy - FormFragment

Fragment obsahuje několik objektů *TextView*, *EditText*, *Spinner* a *CheckBox* pro zadání příslušných údajů nastavení mapy. Uživatel pro vytvoření nastavení mapy musí zadat: název mapy, počáteční kapitál (množství peněz, které má od začátku k dispozici pro nákup a vylepšení věží), počet útočících vln, čas mezi jednotlivými vlnami a počet nepřátel, kteří mohou projít do cíle bez ukončení hry. Dále uživatel vybírá pozadí mapy, texturu cesty a typy věží, které bude možné ve vytvářené mapě kupovat (viz Obrázek 7.1).

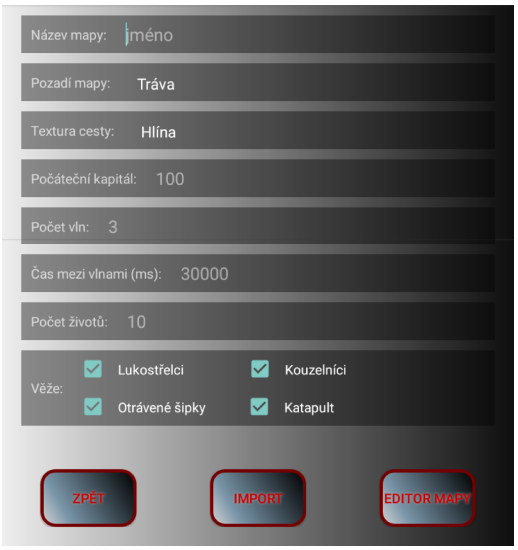
K dispozici je i možnost importovat uživateli vytvořené mapy a upravovat je. Tyto mapy potom v hlavním menu můžeme exportovat a importovat do jiného zařízení.

Tlačítkem *editor mapy* uložíme do souboru *opt_název.txt* zadané nastavení a dostaneme se do druhé části editoru k vytváření mapy.

7.2 Tvorba mapy - EditorFragment

V tomto fragmentu se vytvoří nový *EditorView* dědící *View* (viz kapitola 5.3.1).

Cesta se kreslí tahem po displeji. Jednotlivé tahy jsou přidány jako část cesty do *LinkedList<Line>* (viz kapitola 7.2.3) odkud jsou vykreslovány. Pokud chceme část cesty zrušit, pomocí dvojitého poklepání z *LinkedList<>* postupně odebereme poslední přidanou část až na prázdnou mapu. Dlouhým



Obrázek 7.1: Formulář nastavení mapy

podržením vyvoláme nabídku k uložení mapy. Následně zbývá vybrat začátek a konec cesty a potvrdit uložení. Mapa se ukládá do souboru *map_název.txt*.

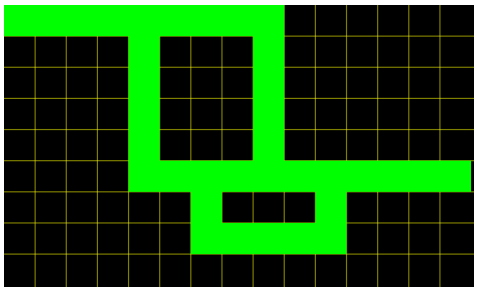
Pokud byla v předchozí části importováno nastavení mapy, načte se příslušný soubor s mapou a vykreslí se do mřížkové struktury. Následně můžeme přidávat, nebo odebírat cestu a stejně jako v prvním případě uložit.

7.2.1 onDraw(Canvas canvas)

Tato metoda se stará o vykreslení všech objektů, mřížky pozadí a cesty voláním metody *onDraw(Canvas canvas)* třídy *Line* (viz kapitola 7.2.3). Pokud úsek cesty ještě není uložen do listu, uživatel stále táhne po displeji, zobrazuje se červeně. Uložený úsek cesty se zobrazuje zeleně (viz Obrázek 7.2). Při výběru začátku a konce cesty jsou tyto souřadnice označeny modře (viz Obrázek 7.3).

7.2.2 onTouchEvent(MotionEvent event)

V této metodě se obstarává interakce s uživatelem. Do listu se přidávají úseky cesty. Využívá se *MotionEvent.ACTION_MOVE* pro získání počátečních souřadnic úseku cesty a *MotionEvent.ACTION_UP* pro určení konco-



Obrázek 7.2: Kreslení mapy

Obrázek 7.3: Dokončení mapy a uložení

vých souřadnic. Hlídá se doba stisku, pokud je větší něž maximální definovaná, zobrazí se dialogové okno s výzvou pro zadání začátku a konce cesty. V tomto případě se používá *MotionEvent.ACTION_DOWN*. Následně je se souhlasem uživatele mapa uložena.

7.2.3 Line

Tato třída představuje úsek cesty. Obsahuje počáteční a koncové souřadnice x, y. V metodě *onDraw(Canvas canvas)* se s těmito souřadnicemi vykresluje obdelník pomocí *drawRect*.

8 Testování

Aplikace byla testována na několika různých zařízení (viz tabulka 8.1).

Zařízení	**Verze systému Android**	**Rozlišení displeje**	**RAM [MB]**	**CPU [GHz]**	**počet jader**
HTC One V T320e	4.0	800x480	512	1.0	1
Samsung Nexus S	4.1.2	800x480	512	1.0	1
LG G2 D802	4.2	1920x1080	2048	2.3	4
Samsung Galaxy Core Prime VE G361	5.1	800x480	1024	1.2	4
Samsung Galaxy S3 mini VE I8200	4.2	800x480	1024	1.2	2
Samsung Galaxy S3 I9300	4.3	1280x720	1024	1.4	4

Tabulka 8.1: Přehled testovacích zařízení

8.1 Popis testování

Hru testovalo několik uživatelů. Nejprve se zkoušelo správné zobrazení na různých rozlišeních displejů. Správné vykreslení jednotlivých buňek mapy, rychlost chůze nepřátel a animace věží. Následně se testovala schopnost nepřátel chodit po vyznačené cestě s náhodným rozhodováním, jakým směrem se na rozcestí vydají. Dalším krokem bylo testování střelby věží, základní dohrání úrovně, uložení a následné zobrazení skóre a grafů. Testoval se editor levelů, import a export map, vyváženost síly nepřátel a věží. V poslední řadě se testovala rozhodovací schopnost nepřátel při výběru nejlepší cesty k cíly a náročnost aplikace, aby hra běžela plynule bez sekání.

8.2 Výsledky testování

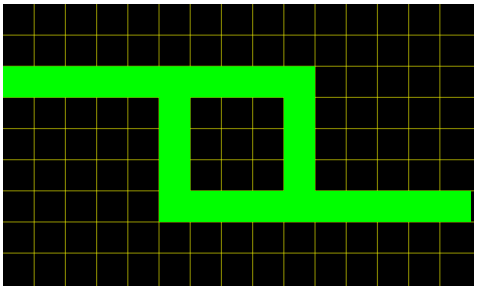
Díky testování byly odhaleny problémy s animací věží, které se na slabších zařízeních pohybovaly trhaně. Následně byla do nastavení přidána možnost tyto animace vypnout a tím snížit náročnost aplikace.

Při testování se přišlo na nepřesnost střelby věží. Přesnost se zlepšila zaměřením střely na konkrétního nepřítele a dopočítávání nové polohy v průběhu letu střely.

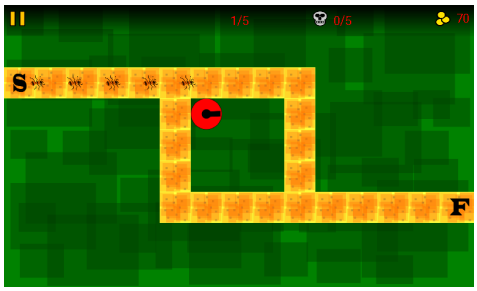
Další nalezený problém byl špatný pohyb nepřátel po cestě. Na některých zařízeních se nepřítel dostal mimo cestu. V důsledku této chyby byl celý systém pohybu a rozpoznávání okolí nepřátel změněn. Nyní se tento problém již nevyskytuje. Pohyb nepřátel a jejich rozhodování výběru nejlepší cesty se díky testování postupně vyvíjel. Nejprve volili cestu náhodně, poté byla přidána schopnost vybrat směr cesty podle vzdálenosti od cíle. Směr blíže k cíli bude předpokládaná kratší cesta. Následně se testovalo předávání informací mezi jednotlivými nepřáteli přes „velitele“. Tím se postupně pro nepřátele odkryla mapa, kde byly zobrazeny věže a možné cesty.

8.2.1 Výsledky testování editoru levelů

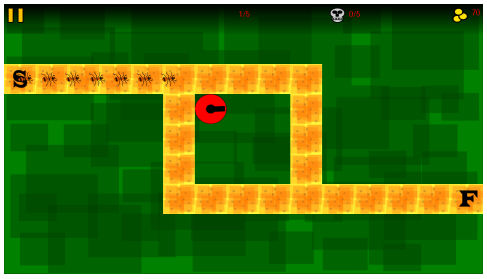
Samostatnou kapitolou pro testování byl editor levelů, správné zobrazení formuláře, import existující mapy. Díky testování se upravilo vykreslování cesty v editoru i zjednodušilo ovládání. Nutností bylo otestovat, jak se bude zobrazovat vytvořená mapa v editoru na zařízení X (viz Obrázek 8.1), spuštěná na stejném zařízení (viz Obrázek 8.2) a po importu a spuštění na zařízení Y (viz Obrázek 8.3).



Obrázek 8.1: Vytvářená testovací mapa v zařízení X



Obrázek 8.2: Vytvořená testovací mapa spuštěná v původním zařízení X



Obrázek 8.3: Vytvořená testovací mapa zobrazená v druhém zařízení Y po importu

Zobrazení i chování hry je na obou zařízeních v pořádku. Pro tento konkrétní případ byla použita testovací zařízení X - Samsung Galaxy Core Prime VE G361, Y - Samsung Galaxy S3 I9300.

8.2.2 Možná rozšíření hry

V závisloti na testování navrhuji možná rozšíření hry. Dali by se přidat další typy nepřátel, pro větší variabilitu a různorodost, nebo jiné útoky věží. Například v závisloti na úrovni věže. V této verzi aplikace roste útočná síla a rychlost střelby, ale nemění se typ střel.

Aplikace by se dále mohla rozšířit o síťovou hru, kdy by jeden hráč stavěl věže a bránil cestu. Druhý hráč by měl za úkol posílat různé typy nepřátel podle postavené obrany Jeho cílem by bylo dostat co nejvíce nepřátel do cíle.

9 Závěr

Cílem tohoto projektu bylo analyzování a prozkoumání některých her typu Tower Defense jak v mobilních zařízeních, tak na počítač. Dalším úkolem bylo načerpané poznatky použít k návrhu vlastní hry s prvky, které se v těchto hrách většinou nevyskytují.

Příkladem může být rozhodovací schopnost nepřátel k výběru nejlepší cesty, která se může větvit. Cestu nepřítel vybírá na základě předpokládané vzdálenosti k cíli, nebo podle obtížnosti průchodu daným úsekem cesty. Obtížnost je vypočtena podle počtu věží, které na nepřítele mohou vystřelit, pokud se vydá touto cestou. Dalším rozšířením bylo shromažďování dosažených výsledků a jejich zobrazení ve formě grafů. Grafy byly vytvořeny pomocí frameworku *GraphView*. Poslední důležitou rozšiřující částí aplikace byl editor úrovní, kde lze vytvořit vlastní mapu a nadefinovat použitelné prostředky pro hraní této úrovně. K dispozici je možnost exportovat mapu, která se potom dá importovat na jiném zařízení. Importovanou mapu například lze vložit do editoru a upravit ji.

Aplikace a především hry v mobilních zařízeních by měly mít intuitivní a jednoduché ovládání, které zvládneme i při zhoršených podmínkách, nejen doma v křesle. Další důležitou vlastností je příjemná a chytlavá grafika. Těmito zásadami jsem se snažil při tvorbě této hry řídit.

Literatura

[1] Activity. URL: <http://developer.android.com/reference/android/app/Activity.html>, [online], cit. 2016-01-04.

[2] Co jsou věžovky. URL: <http://vezovky.cz/co-jsou-vezovky>, [online], cit. 2015-06-15.

[3] Fragment. URL: <https://developer.android.com/guide/components/fragments.html>, [online], cit. 2016-01-04.

[4] Frontline Defense. URL: <http://poki.cz/g/frontline-defense>, [online], cit. 2015-10-01.

[5] Kingdom Rush. URL: <http://www.kingdomrush.com/>, [online], cit. 2013-01-01.

[6] Kingdom Rush. URL: <http://poki.cz/g/kingdom-rush>, [online], cit. 2015-11-01.

[7] Pixel Defender. URL: <http://poki.cz/g/pixelova-obrana>, [online], cit. 2015-10-23.

[8] B.V., E. I. M.: Cross-platform mobile development at your fingertips. URL:<http://www.edgelib.com>, [online], cit. 2016-03-11.

[9] DiMarzio, J. F.: *Programujeme hry pro Android*. Brno: Computer Press, 2012, ISBN 978-80-247-4863-4.

[10] Gehring, J.: GraphView - open source graph plotting library for Android. URL:<http://www.android-graphview.org/>, [online], cit. 2015-04-29.

[11] Hoog, A.: *Android Forensics*. 225 Wyman Street, USA: Elsevier Inc., 2011, ISBN 978-1-59749-651-3.

[12] Jíří Vávrů, M. U.: *Programujeme pro Android*. U Průhonu 22, Praha: Grada Publishing a.s., 2013, ISBN 978-80-247-4863-4.

[13] Konečný, M.: Seriál: Vyvíjíme pro Android. URL:<https://www.zdrojak.cz/serialy/vyvijime-pro-android/>, [online], cit. 2012-09-14.

[14] Labs, C.: Build 10x Faster. URL:<https://coronalabs.com/>, [online], cit. 2016-06-1.

[15] emo-framework project: Emo open source framework for games. URL:<http://www.emo-framework.com/>, [online], cit. 2011-04-5.

[16] Semecký, V.: Android Studio - nové vývojové prostředí. URL: <https://www.zdrojak.cz/clanky/android-studio-nove-vyvojove-prostredi/>, [online], cit. 2015-10-06.

[17] Technologies, U.: The best development platform for creating games. URL:<https://unity3d.com/unity>, [online], cit. 2014-04-27.

[18] Zapata, B. C.: *Android Studio Application Development*. 35 Livery Street, UK: Packt Publishing, 2013, ISBN 978-1-78328-527-3.

Seznam příloh

- Příloha A: Instalace
- Příloha B: Uživatelská dokumentace
- Příloha C: ScreenShoty
- Příloha D: Struktura přiloženého CD

A Instalace

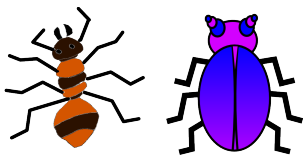
Před instalací aplikace je nutné v nastavení telefonu povolit instalaci z neznámých zdrojů. Většinou se nachází v *Nastavení -> Zabezpečení -> Neznámé zdroje*.

Na přiloženém CD lze nalézt soubor BugWars.apk, který stačí zkopírovat do jakéhokoliv adresáře ve vašem mobilním zařízení se systémem Android. Pomocí některého nástroje na prohledávání složek (např. ES File Explorer) najdeme soubor BugWars.apk. Po kliknutí na soubor se zařízení nejdříve zeptá, jestli chceme tuto aplikaci nainstalovat a poté proběhne instalace. Po úspěšné instalaci můžeme spustit aplikaci BugWars a začít hrát.

Pokud chceme instalovat testovací verzi aplikace, která má odemčené všechny herní mapy, postupujeme obdobně u souboru BugWarsTest.apk.

B Uživatelská dokumentace

Vytvořená hra je typu „Tower Defense“. Hráč má za úkol pomocí věží ubránit cestu před průchodem nepřátel. Každá věž má jiné poškození a rychlost střelby, tyto atributy se zlepšují s upgradováním věží na vyšší úroveň. Každý typ nepřátel má také rozdílné množství životů a rychlost pohybu (viz Obrázek B.1).



Obrázek B.1: Nepřátelé

B.1 Hlavní menu

Po spuštění aplikace se nejprve zobrazí hlavní menu hry (viz Obrázek B.2. Z této nabídky se hráč může dostat do všech dalších částí hry. Zobrazit dialog „O programu“, upravit nastavení, prohlédnout si parametry věží a nepřátel a vytvořit si vlastní mapu. Po odehrání hry se lze z hlavního menu dostat do tabulky nejlepších skóre.

B.2 Import a export

Do hry lze importovat uživateli vytvořené mapy. Hráč umístí získané soubory „opt_název.txt“ a „map_název.txt“ do hrou vytvořené složky „BUGWARS“ v paměti zařízení. Ve hře následně pomocí tlačítka import (tlačítko červené šipky v hlavním menu) vloží vybranou mapu do hry. Mapa je poté připravená k hraní ve výběru levelů se speciální ikonou, aby šlo uživatelské mapy snadno rozpoznat (viz Obrázek B.3).

Obdobně lze vytvořenou mapu do složky „BUGWARS“ ze hry vložit tlačítkem export (tlačítko zelené šipky v hlavním menu).

Obrázek B.2: Hlavní menu

Obrázek B.3: Import map do aplikace

B.3 Nastavení

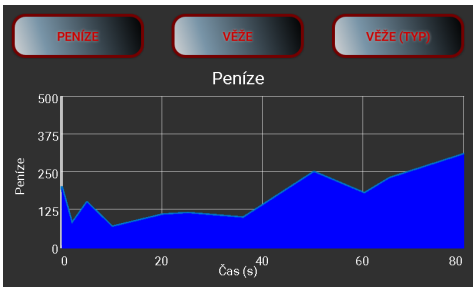
V nastavení lze přepínat jazyk hry, vypínat, nebo zapínat hudbu a animace. Pokud chceme nastavení ponechat i pro příští spuštění, je potřeba nastavení uložit. Při změně jazyka je navíc nutné hru restartovat.

B.4 Výběr levelů

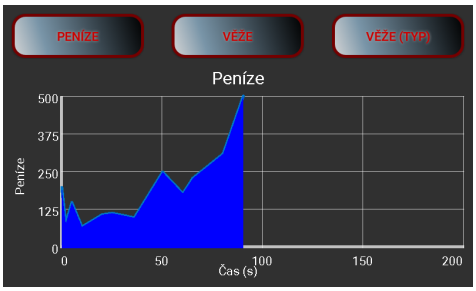
Z hlavního menu se tlačítkem „HRA“ dostaneme do výběru levelů. Zelená ikona značí odemčenou úroveň, červená zamčenou. Taková úroveň se odemkne jen dohráním předešlé. Pokud je ikona zelená s písmenem „U“, jedná se o úroveň vloženou uživatelem. Po kliknutí na vybranou položku začíná hra.

B.5 Skóre

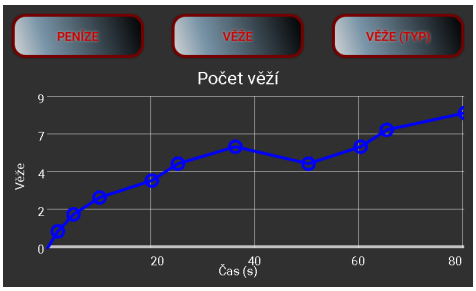
Pokud v hlavním menu vybereme položku „SKÓRE“, dostaneme se přes seznam úrovní až na jednotlivé výsledky. Po rozkliknutí konkrétního výsledku se zobrazí grafy vykreslující zisk peněz a věží v průběhu hry. Grafy se dají scrollovat a oddalovat (viz Obrázek B.4, B.5, B.6, B.7).



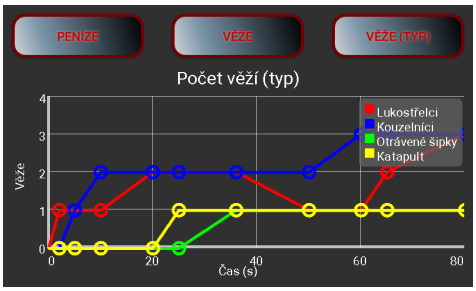
Obrázek B.4: Nárůst peněz



Obrázek B.5: Nárůst peněz, oddálení grafu



Obrázek B.6: Nákup věží



Obrázek B.7: Nákup věží podle typu

B.6 Editor levelů

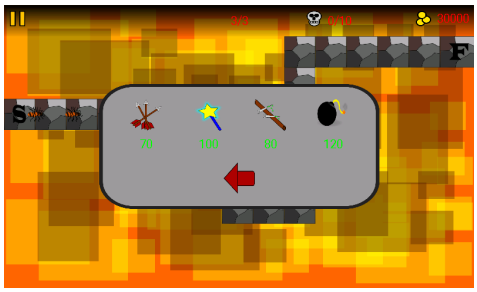
Uživatel má za úkol vyplnit formulář s nastavením mapy, nebo vložit existující mapu, která již je do hry importována, pro její úpravu. Tlačítkem „EDITOR MAPY“ se uživatel dostane do druhé části kde tahem po displeji kreslí jednotlivé úseky cesty. Pokud se chce vrátit o krok zpět a odstranit poslední vloženou část cesty, provede dvojklik. V případě, že je mapa hotová, pomocí dlouhého stisku vyvolá dialog s výzvou k označení začátku a konce cesty. Po získání těchto dvou souřadnic je mapa uložena.

B.7 Ovládání hry

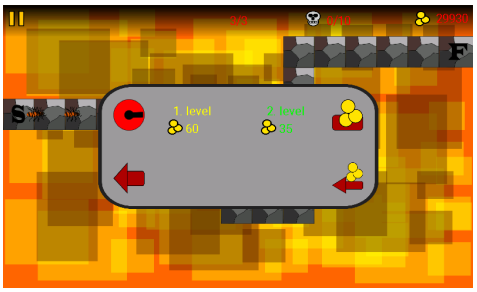
Hráč kliknutím na herní mapu mimo vyznačenou cestu vyvolá nabídku pro koupi věží (viz Obrázek B.8), kliknutím na některou zobrazenou věž ji koupí. Kliknutím na již vytvořenou věž je zobrazena tabulka s úrovní věže, její cenou

pro případný prodej a cenou za zvýšení úrovně. Ikonou v pravém horním rohu tabulky se věž vylepší a ikonou v pravém spodním rohu tabulky se prodá. Ikona samostatné šipky umožňuje pokračování ve hře bez provedení jiné akce (viz Obrázek B.9). Hra lze pozastavit tlačítkem v levém horním rohu.

Úkolem hráče je nenechat projít víc nepřátel než je označeno v horní liště u ikony lebky. Peníze za které se kupují věže se získávají za ničení nepřátel a jsou zobrazeny v pravém horním rohu. Číslo nahoře uprostřed značí kolik nepřátelských vln musí hráč překonat (viz Obrázek B.10).



Obrázek B.8: Tabulka pro nákup věží

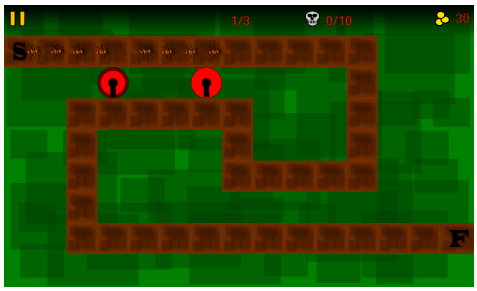


Obrázek B.9: Tabulka pro vylepšení, nebo prodej věže

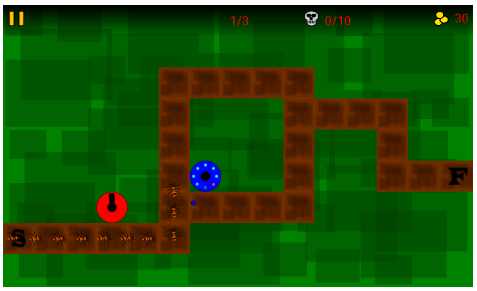
Obrázek B.10: Ukázka herní úrovně

C ScreenShoty

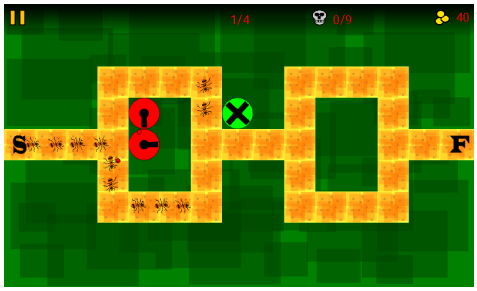
ScreenShoty zobrazující jednotlivé herní úrovně a seznam jejich výběru.



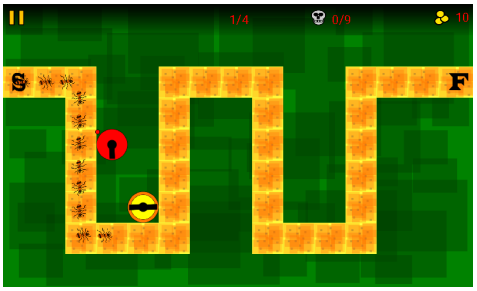
Obrázek C.1: Mapa první úrovně



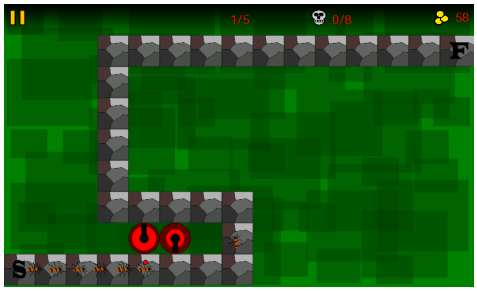
Obrázek C.2: Mapa druhé úrovně



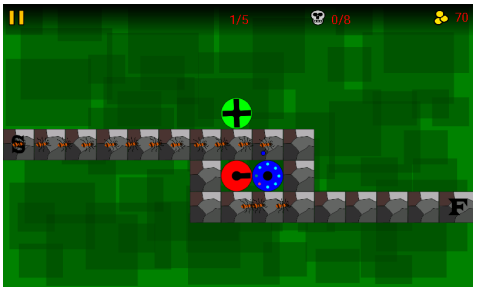
Obrázek C.3: Mapa třetí úrovně



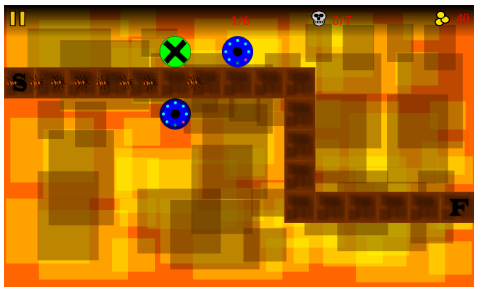
Obrázek C.4: Mapa čtvrté úrovně



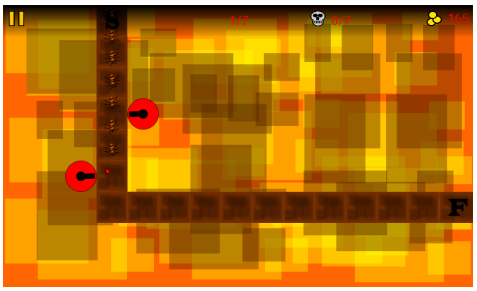
Obrázek C.5: Mapa páté úrovně



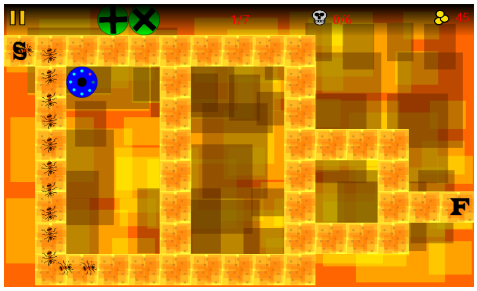
Obrázek C.6: Mapa šesté úrovně



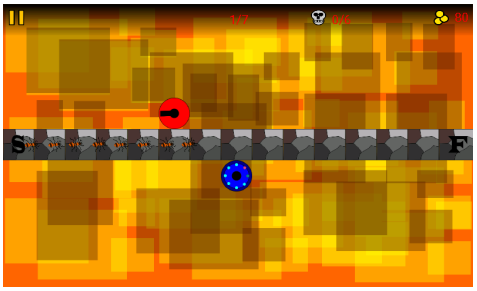
Obrázek C.7: Mapa sedmé úrovně



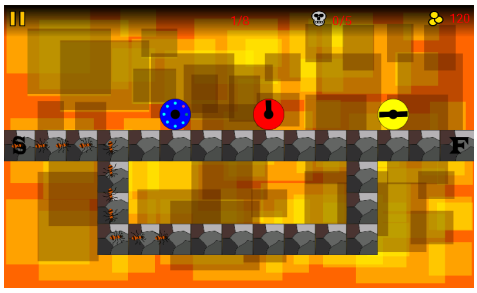
Obrázek C.8: Mapa osmé úrovně



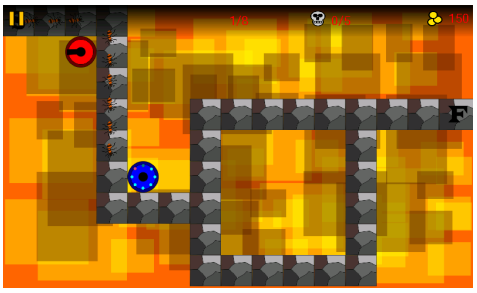
Obrázek C.9: Mapa deváté úrovně



Obrázek C.10: Mapa desáté úrovně



Obrázek C.11: Mapa jedenácté úrovně



Obrázek C.12: Mapa dvanácté úrovně

Obrázek C.13: Výběr herních úrovní

D Struktura přiloženého CD

1. *apk:* Obsahuje dva soubory BugWars.apk a BugWarsTest.apk, pomocí nich můžeme nainstalovat aplikaci na mobilní zařízení. BugWarsTest.apk slouží k testování, od začátku jsou dostupné všechny úrovně.
2. *doc:* Obsahuje dokument bakalářské práce ve formátu PDF.
3. *javadoc:* Javadoc dokumentace k projektu.
4. *projekt:* Projekt aplikace v Android Studiu (zdrojové kódy, textury, apod.).
5. *tex:* Zdrojový kód dokumentu se seznamem literatury a obrázky.