Gymnázium, Praha 6, Arabská 14

SkyJump, vedoucí práce Mgr. J. Lána

SkyJump

Ročníková práce

Šimon Hájek 2.E

Duben 2023

Prohlašuji, že jsem jediným autorem 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 uděluji 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 28. dubna 2023

Anotace

Cílem mé ročníkové práce bylo vytvoření hry, která je inspirovaná hrou *Doodle Jump* či minihrou ve známé hře *Pou*. Program funguje tak, že postavička se po startu objeví na ostrově a hráč se snaží pohybem do stran doskočit na ostrovy výše, které jsou generovány náhodně na obrazovce se stejným výškovým rozestupem. Jakmile se postavička dotkne ostrova, automaticky vyskočí výše a počet bodů se zvýší od 1. Smysl hry je získat co nejvíce bodů, dokud hráč nespadne a hra se neukončí.

Abstract

The goal of my project was to create a game, that is heavily inspired by mobile games like *Doodle Jump* or a minigame in the famous game *Pou*. The way my program works is that once the game starts, a character is spawned on an island, where your goal is to make your way up by moving to the right or left by touching either side of the screen and landing on a randomly generated island. When the character collides with an island, it automatically jumps upwards again and the score is increased by one point. The objective is to achieve the highest score before you die by not landing on an island.

Zadání

Zadáním mé ročníkové práce bylo vytvořit hru na telefon inspirovanou hrami jako *Doodle Jump* nebo minihrou ve hře *Pou*, tedy typická skákací hra s cílem dosáhnout co nejvíce bodů. Postava se odrážením od ostrovů pohybuje směrem nahoru a hráč tak získává body. Hra se ukončí tehdy, když hráč omylem nedopadne na žádný ostrov a spadne tak dolů. Program si ukládá nejvyšší dosáhnuté skóre do souboru, takže i po restartu aplikace lze vidět váš nejvyšší počet dosažených bodů.

Obsah

1. Úvod

Cílem této práce bylo vytvořit již zmíněnou napodobeninu her jako *Doodle Jump* či minihru ve hře *Pou*. Na zpracování aplikace jsem si vybral vývojové prostředí Android Studio, které je založeno na IntelliJ IDEA. Jeden z důvodů, proč jsem program nedělal v IntelliJ IDEA za pomocí JavaFX byl ten, že v porovnání má Android Studio mnohem lepší rozhraní aplikace, zejména tedy při úpravě .xml kódu, které je jakožto alternativa Scene Builderu mnohem pokročilejší a celkově vypadá nověji. Dalším důvodem bylo to, že s vytvářením her a aplikací pro telefon obecně nemám žádné zkušenosti a chtěl jsem se tak něco nového naučit. Jakožto bonus jsem chtěl mít svoji hru vždy u sebe, aby byla možnost kdykoli aplikaci ukázat ostatním bez nutnosti přítomnosti počítače. Tento dokument se v několika kapitolách věnuje dokumentaci kódu a relevantních oblastí spojených se hrou.

2. Inspirace

2.1. Pou

Pou je Mobilní aplikace pro Blackberry, iOS a Android. Vytvořil a vydal ji v roce 2012 Paul Salameh. Představuje malého mimozemšťanka trojhranného tvaru, v základní podobě hnědé barvy. Hráč se o něj stará podobným způsobem, jako o Tamagoči. Poua je třeba krmit, mýt a dopřát mu dostatek pohybu a spánku. Hra obsahuje i různé minihry a za získané „mince“ lze zakoupit kostýmy, dekorace, nebo změny prostředí. Pokud je hráč připojen k internetu, je možná interakce mezi Pouy různých hráčů. [1]

Jedna z miniher je právě hra, ze které jsem se inspiroval. Originál je však v porovnání s mojí verzí obohacen o různé funkce navíc, např. o mince, které se jednou za čas náhodně objevují na ostrovech a za které si pak můžete pro vašeho Poua něco koupit v obchodě. Další odlišností jsou různé typy ostrovů, na které je možné přistát nebo třeba způsob ovládání, které zde funguje pomocí gyroskopu v telefonu. Kromě základního hliněného bloku se zde můžeme v závislosti s dosaženou výškou setkat například s mraky

[1] https://cs.wikipedia.org/wiki/Pou

nebo letadly, která mají svoje vlastní chování, například jednorázové přistání, po kterém ostrov zmizí a není na něj možné znovu přistát. Celkově se originál odlišuje třeba i průběžnou změnou pozadí, které je za ostrovy umístěno.

Obrázek 1 Minihra ve hře Pou

2.2. Doodle Jump

Doodle Jump je platformová videohra vyvinutá a publikovaná chorvatským studiem Lima Sky. Ve hře je cílem vést čtyřnohého tvora zvaného „The Doodler“ po nekonečné řadě plošin bez pádu. Ovládání postavy funguje pomocí naklánějí zařízení ze strany na stranu

Hráči mohou získat krátkou podporu z různých předmětů, jako jsou klobouky vrtulí, jetpacky, rakety, pružiny, trampolíny a štíty proti nezranitelnosti (pouze některé úrovně). Existují také monstra a UFO, kterým se Doodler musí vyhnout, střílet nebo na ně skočit, aby je odstranil. Zaměřování se provádí poklepáváním na různé části obrazovky, na verzi hry pro Android a Windows Phone je k dispozici i režim automatického zaměřování. V závislosti na herním režimu, který hrajete, mohou projektily létat přímo z obrazovky nebo mohou být ovlivněny gravitací a padat dolů. [2]

3. Program

3.1. Vzhled a GUI

Při zapnutí aplikace je hráč uvítán startovací obrazovkou, kde se uprostřed nachází zvětšená postavička stojící na ostrově s velkým nápisem názvu hry. Na obrazovce se dále nachází tlačítko ***hrát*** spolu s tlačítkem ***reset highscore***. Tlačítko *reset highscore* vymaže nejlepší dosažené skóre, kterého hráč kdy dosáhl. Celá scéna má oblohu v pozadí jako obrázek.

Po kliknutí na tlačítko hrát se hráč přesune na hrací scénu, kde se nachází postavička a několik náhodně rozpoložených ostrovů. V levém horním je počítadlo bodů, kterých hráč dosáhl.

Při pádu mimo ostrov je otevřena nová scéna, kde se nachází text, který oznamuje konec hry, tlačítko restartu hry a text s dosaženým počtem bodů a nejvyšším počtem dosažených bodů.

[2] https://en.wikipedia.org/wiki/Doodle_Jump

3.2. Funkce programu

Aplikace je naprogramována ve vývojovém prostředí Android Studio, což je populární nástroj pro vyvíjení aplikací a her pro operační systém Android. Soubory, ve kterých jsem pracoval byly 3 třídy pro 3 různé scény a k nim příslušné *.xml* soubory. Nachází se ještě soubor *ostrov.xml*, který je použit při generování nových ostrovů na obrazovce a ve kterém se nachází pouze samotný obrázek ostrovu.

Obsah třídy *StartActivity.java* tvoří pouze spuštění další scény po kliknutí na tlačítko spolu s vynulováním nejlepšího skóre po stisknutí na tlačítko *reset highscore*.

Hlavní třída, ve které se nachází veškerá logika programu je třída *HraActivity*. V následujících podkapitolách rozeberu části kódu, které jsou něčím zajímavé nebo s kterými jsem měl problém s tím je vymyslet.

3.2.2. GameLoop

První velký úkol, jehož řešení bylo zapotřebí vymyslet, bylo vytvoření jakéhosi cyklu, který se bude opakovat několikrát za sekundu, a umožňovat, tak plynulé zobrazení veškerých pohybů ve hře. I přes jednoduchost konečného řešení jsem s jeho vytvářením měl poměrně velký problém, protože jsem vůbec netušil, jak se při vývoji aplikace s plynulým pohybem postupuje. Problém jsem řešil pomocí interface Runnable() spolu

```
Handler handler = new Handler();

Runnable gameLoop = new Runnable() {
    @Override
    public void run() {

    //kód opakující se 30x za sekundu

    }
};

    //prvotní spuštění gameLoop
    handler.postDelayed(gameLoop, delayMillis: 100 / 3);
```

Obrázek 2 Kód cyklu gameLoop

s třídou Handler(). Dovnitř gameLoopu jsem vložil veškeré funcke a třídy, které bylo nutné spouštět každý možný snímek. Aby toto fungovalo, bylo zapotřebí vložit znovuspuštění Cyklu před jeho konec, ale také po jeho konci, aby poprvé začal a fungoval od té doby sám. Obsah *Runnable gameLoop* budu ještě popisovat v pozdějších kapitolách.

3.2.3. Detekování kolizí

Funkce pro detekování kolizí postavy s ostrovy je aktivní 30x za sekundu, ale pouze v případě, že je vertikální rychlost postavy vyšší než 0, tedy padá směrem dolů. Tato podmínka zajišťuje to, že se průlet ostrovem zdola nezapočítá jako kolize.

Uvnitř funkce nejprve program zjistí pozici postavy na obrazovce a uloží si ji do již dříve založeného pole *location1*. To samé udělá pro všechny ostrovy, které v danou chvíli existují pomocí cyklu for-each. Pomocí jednoduché podmínky zjistíme, zda došlo ke kolizi těchto dvou objektů, a pokud ano, stane se několik věcí: Program nastaví rychlost postavy na -100 a postava tedy „vyskočí“, číslo nejvýše dosaženého ostrova v proměnné *highestTouched* se v případě, že se jedná o nový ostrov zvětší o 1 a nakonec se nepřímo zavolá funkce *generuj()*.

```
//pokud postava pada dolu, kontroluj kolize
if (speedY > 0) {
    imageView1.getLocationOnScreen(location1);
    for (View v :
            ostrovy) {
        v.getLocationOnScreen(location2);

        int left1 = location1[0];
        int top1 = location1[1];
        int right1 = left1 + imageView1.getWidth();
        int bottom1 = top1 + imageView1.getHeight();

        int left2 = location2[0];
        int top2 = location2[1];
        int right2 = left2 + v.getWidth();
        int bottom2 = top2 + v.getHeight();
        //pokud postava pristane na ostrov (uprava pixelu kvuli okrajum obrazku), znovu vyskoc a pricti skore pokud postava jeste na ostrove nebyla
        if (right1 > left2 + 50 && left1 < right2 - 50 && bottom1 - 150 > top2 && top1 < bottom2) {
            speedY = -100;
            if ((int) v.getTag() > highestTouched) {
                highestTouched = (int) v.getTag();
                generovat = true;
            } //zmena textu na aktualni skore
            tv1.setText(String.valueOf(highestTouched));
        }
```

Obrázek 3 Kód pro detekci kolizí

3.2.3. Generování ostrovů

Velmi důležitou funkcí programu bylo generování ostrovů tak, aby byly umístěny náhodně na obrazovce a zároveň na všechny z nich platila všechna pravidla, tedy třeba detekce kolizí. Způsob, jakým byl tento problém řešen, bylo vytvoření *ArrayListu* s ostrovy, kde jich je v jeden moment najednou jen pět, a kde se s každým novým vygenerovaným ostrovem ten nejspodnější z nich odstraní. Při zjištění kolize se spustí funkce pro generování a odstraňování ostrovů.

Náhodné generování funguje díky vytvoření proměnné *view*, do které je přiřazen ostrov z *.xml* souboru. Funkce mu pak následně upravuje parametry a zařazuje do layoutu, a nakonec ho přidá do ArrayListu *ostrovy*. Pokud byla funkce zavolána s booleanem *maz* nastaveným na *true*, nejstarší ostrov z listu se zneviditelní a následně odebere.

Náhodná pozice je vytvořena pomocí *Math.random()*, které vytvoří náhodné číslo ve správném rozsahu jako souřadnici pro osu X. Dále je zde nastavena i výšková vzdálenost ostrovů od sebe, která je aktuálně 700 pixelů. Identifikace ostrovů v listu funguje díky *.setTag()*.

```
//generovani ostrovu na nahodnou horizontalni souradnici s vertikalnim odstupem 700
public void generuj(boolean maz) {
    View view = LayoutInflater.from(HraActivity.this).inflate(R.layout.ostrov,  root: null);
    parentLayout.addView(view);

    view.setTag(Integer.parseInt(ostrovy.get(ostrovy.size() - 1).getTag().toString()) + 1);
    view.setX((float) (Math.random() * (parentLayout.getWidth() - imageView2.getWidth())));
    view.setY(ostrovy.get(ostrovy.size() - 1).getY() - 700);
    view.setLayoutParams(new ViewGroup.LayoutParams(imageView2.getWidth(), imageView2.getHeight()));
    ostrovy.add(view);
    //mazani ostrovu
    if (maz) {
        ostrovy.get(0).setVisibility(View.INVISIBLE); //zmizeni ostrova
        ostrovy.remove( index: 0); //odstraneni z ArrayListu ostrovu
    }
}
}
```

Obrázek 4 Funkce pro generování nových ostrovů

3.2.4. Zákaz vytvoření druhého ostrova veprostřed

Dalším problémem, s kterým jsem se potýkal, bylo zajištění generování druhého ostrova na jiném místě než uprostřed. Důvod vytvoření této podmínky bylo to, že při náhodném generování se druhý ostrov někdy vytvořil přímo nad ostrovem prvním a postava tak hned vyskočila na druhý ostrov, někdy i výše. Podmínka se nachází přímo pod generováním prvních pěti ostrovů s booleanem *maz* nastaveným na *false*, tedy bez odstraňování spodních ostrovů. Toto je také důvod přítomnosti této metody. Existovaly by samozřejmě i jiné způsoby, jak stejné věci docílit, ale tato mi přišla nejjednodušší. Ve voidu *onGlobalLayout()*, který se spustí pouze jednou na začátku programu je mimo generování ostrovů část kódu věnovaná znovu umístění druhého ostrova v případě, že se vygeneroval moc blízko prvnímu.

Původně jsem měl v plánu implementovat tuto podmínku pro všechny ostrovy na mapě, nicméně kvůli nedostatku času jsem tak neučinil. Od třetího ostrova výše je tak možné spatřit několik ostrovů na stejném místě hned za sebou.

```
ViewTreeObserver vto = parentLayout.getViewTreeObserver();
vto.addOnGlobalLayoutListener(new ViewTreeObserver.OnGlobalLayoutListener() {
    //generovani ostrovu
    @Override
    public void onGlobalLayout() {
        for (int i = 0; i < 5; i++) {
            generuj( maz: false);
        }
        //zakaz spawnu druheho ostrova primo nad prvnim
        while (ostrovy.get(1).getX() < ostrovy.get(0).getX() + imageView2.getWidth() && ostrovy.get(1).getX() > ostrovy.get(0).getX() - imageView2.getWidth()) {
            ostrovy.get(1).setX((float) (Math.random() * (parentLayout.getWidth() - imageView2.getWidth())));
        }
        ViewTreeObserver obs = parentLayout.getViewTreeObserver();
        obs.removeOnGlobalLayoutListener( victim: this);
    }
});
```

Obrázek 5 Vytvoření druhého ostrova mimo střed

3.2.4. Ukládání nejlepšího skóre

Ukládání nejvyššího počtu bodů bylo něco, co jsem měl již od začátku vytváření programu v plánu. Vždy se mi líbila představa, že si můj program něco zapamatuje a není tak jako nový při každém startu. I přes velkou jednoduchost této implementace jsem rád, že v programu něco takového je.

Na rozdíl u programování aplikací v Javě pro počítač zde bylo nutné místo jednoduchého souboru proměnnou ukládat do něčeho jiného. Zvolil jsem si asi nejjednodušší způsob, a to ukládání pomocí objektu *SharedPreferences* a souboru, který objekt vytváří. Odsud je to otázka jen několika řádků kódu pro získání a uložení této hodnoty na základě toho, zda hráč přesáhl rekordní počet bodů či nikoli. Tlačítko, které jsem zmiňoval na začátku tuto hodnotu v souboru nastavuje zpět na 0.

3.2.4. Ovládání postavy do stran

Pohyb postavy do stran jsem řešil pomocí metody, která se zavolá při stisku obrazovky. Dotyky se sledují na layoutu parentLayout, který překrývá celou obrazovku a můžeme tak ovládat pohyb dotykem kdekoliv na displeji. Při stisknutí se zjistí poloha dotyku pomocí getX() a getWidth() a na základě toho se rozhoduje, zda byla stisknuta levá nebo pravá polovina obrazovky. Při uvolnění se tyto proměnné nastavují zpět na *false*, aby se pohyb zastavil. Pokud se uživatel obrazovky nedotýká, vrací se *false*.

```
//rozpoznani dotyku na parentLayout pro pohyb do stran
parentLayout.setOnTouchListener(new View.OnTouchListener() {
    @Override
    public boolean onTouch(View view, MotionEvent event) {
        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                // Rozpoznani toho, jaka polovina obrazovky byla stisknuta
                float x = event.getX();
                int width = view.getWidth();

                if (x < width / 2) {
                    doleva = true;
                } else {
                    doprava = true;
                }
                return true;
            case MotionEvent.ACTION_UP:
                doleva = false;
                doprava = false;
                return true;
            default:
                return false;
```

Obrázek 6 Kód pro pohyb postavy do stran

4. Závěr

Na závěr bych rád můj program a průběh práce zhodnotil. S programem jsem i navzdory menším nedokonalostem spokojen, jelikož i přes velmi malé množství času, které jsem si na vytvoření nechal se finální výtvor podobá tomu, co jsem si na začátku představoval. Mezi chyby a nedostatky, které jsem při psaní neuměl vyřešit nebo které jsem vyřešit nestíhal patří například mizení ostrovů na místě, kde je lze pořád vidět nebo třeba velký rozdíl vizuální (a tím pádem i funkční) stránky programu na jakémkoliv zařízení jiném, než je Google Pixel 3a, který jsem měl nastavený jako emulátor při psaní programu. Někdy je to jen menší vizuální rozdíl, ale někdy to může způsobit značné nevýhody při hraní hry. Je to způsobeno zejména pracováním s pixely místo velikosti obrazovky, např. generování ostrovů 700px nad sebou bude vypadat jinak na každém telefonu. Mimo jiné hra postrádá jakékoliv ztížení obtížnosti postupem času, které má za výsledek velmi nudný zážitek z hraní. Abyste něčeho takového docílili, tak bohužel nelze pouze postupem času zvyšovat hodnotu proměnných pro fyziku, protože to má pak za následek to, že postava například nedoletí na další ostrov a hra se tím pádem stává nehratelnou.

Celkově jsem však s programem velice spokojený a rozhodně mohu říci, že jsem se při programování naučil i něco nového a že jsem se obecně v programování zlepšil.

5. Zdroje

1 **Wikipedie. [Online] [Citace: 28. 4. 2023]**

https://cs.wikipedia.org/wiki/Pou

2 **Wikipedie. [Online] [Citace: 28. 4. 2023]**

https://en.wikipedia.org/wiki/Doodle_Jump

6. Seznam obrázků

Obrázky v práci