Gymnázium, Praha 6, Arabská 14

Programování

Aim Trainer

2020/2021 Jakub Sedláček

Gymnázium, Praha 6, Arabská 14

Programování

Aim Trainer

Předmět: Programování

Téma: Aim Trainer

Autor: Jakub Sedláček

Třída: 4. E

Školní rok: 2020/2021

Vedoucí práce: Mgr. Jan Lána

Třídní učitel: Mgr. Barbora Novosadová

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 29. března 2021 _____

Anotace

Cílem této ročníkové práce bylo vytvořit hru, díky které by si hráč mohl procvičit pohyb s myší a schopnost rychlé reakce. Hráč by postupně klikal na kruhy (terče) na obrazovce, které by při každém kliknutí na ně zmizely. Cílem hry je by bylo dosáhnout největšího skóre tím, že by hráč kliknul na co největší počet terčů co nejrychleji. Hra by měla dva různé módy. V prvním by se terče objevovaly náhodně na obrazovce a hráč by pouze nastavil, jak a jak dlouho by se terče generovaly. V druhém módu by si hráč vytvořil vlastní mapu, ve které by také mohl nadefinovat, na jaké pozici by se terče postupně pokaždé vytvářely.

Obsah

1 Úvod

1.1 Zadání

Aim Trainer je Java GUI aplikace, kde uživatel bude trénovat svůj pohyb s myší klikáním na terče. Cílem hráče je klikat postupně na terče (nehybné objekty, které se budou v daný čas postupně objevovat náhodně rozmístěné na obrazovce a které se po kliknutí označí jako trefené). Pokud hráč na terč neklikne včas, tak terč sám po chvíli zmizí a označí se jako netrefený. Hráči se bude dosažené skóre (i s datem, kdy tohoto skóre dosáhl) na konkrétních "mapách" zaznamenávat do žebříčku nejlepších hodnocení, který ale nebude propojený s ostatními hráči. Aim Trainer je vhodný zejména pro hráče FPS her nebo her obecně, kde jsou potřeba rychlé pohyby s myší, ale i pro běžné uživatele, kteří si chtějí lépe osvojit pohyb myší.

Uživatel má možnost si vybrat obtížnost a různé aspekty jako rychlost přibližování terčů, jejich velikost nebo počet terčů. Uživatel by mohl vytvořit tzv. mapu terčů, kde by se terče pokaždé objevovaly ve stejném pořadí na stejné pozici. Úspěšnost a dosažené skóre by bylo určováno podle procentuální úspěšnosti klikání na terče a podle toho, jestli uživatel kliknul na terč příliš brzo nebo pozdě.

1.2 Cíle projektu

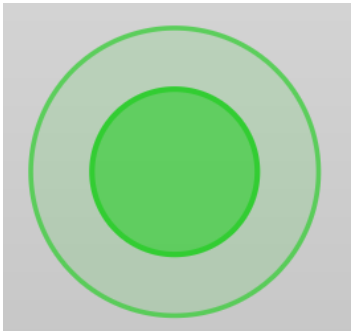
Svojí prací bych chtěl vytvořit jednoduchou pomůcku nejen pro hráče počítačových her, ve kterých je dobrý pohyb s myší velmi důležitý, ale i pro běžného uživatele, který by chtěl vylepšit svůj pohyb s kurzorem. Hra by mohla zaujmout i širší publikum právě díky své zábavné a kompetitivní formě. Hráč by v žebříčku nejlepších výsledků mohl pozorovat svůj pokrok v čase a tím, že by hráč chtěl pokaždé překonat svoje dosavadní nejlepší skóre, by se hráč nakonec zlepšil. Hra by tím také mohla uživatele vtáhnout a zaujmout na delší dobu. Hráč by ale nemusel soupeřit pouze se sebou. Stačí, aby vytvořil vlastní mapu a poslal ji svým kamarádům, se kterými by nakonec soupeřil o nejlepší celkové skóre na dané mapě.

2 Struktura hry

2.1 Terč

Základním prvkem této hry je terč. Terč je ve skutečnosti kruh, na který hráč musí kliknout kurzorem, aby ho trefil. Hráč musí na terč kliknout dříve, než sám zmizí, aby se mu přičetly body. Ty se přičítají podle toho, jak rychle hráč terč trefil.

Terč se skládá ze základního kruhu a z vnějšího přibližovacího kruhu (viz obrázek 2.1). Vnější kruh působí pouze jako indikátor toho, za jak dlouho terč sám zmizí. Jakmile se vnější terč přiblíží k základnímu kruhu, terč zmizí a označí se jako netrefený.



Obrázek 2.1: Terč s vnějším přibližovacím kruhem

Za vytvoření přibližovacího kruhu zodpovídá třída *ApproachCircleGen*. Ta vytvoří objekt *Circle*, který za pomoci *ScaleTransition* a *FadeTransition* příslušně zmenší a pozvolna zobrazí kruh. Přibližovací kruh není instancí této třídy. Tato třída pouze vytvoří přibližovací kruh a vrátí ho v metodě *getCircle*.

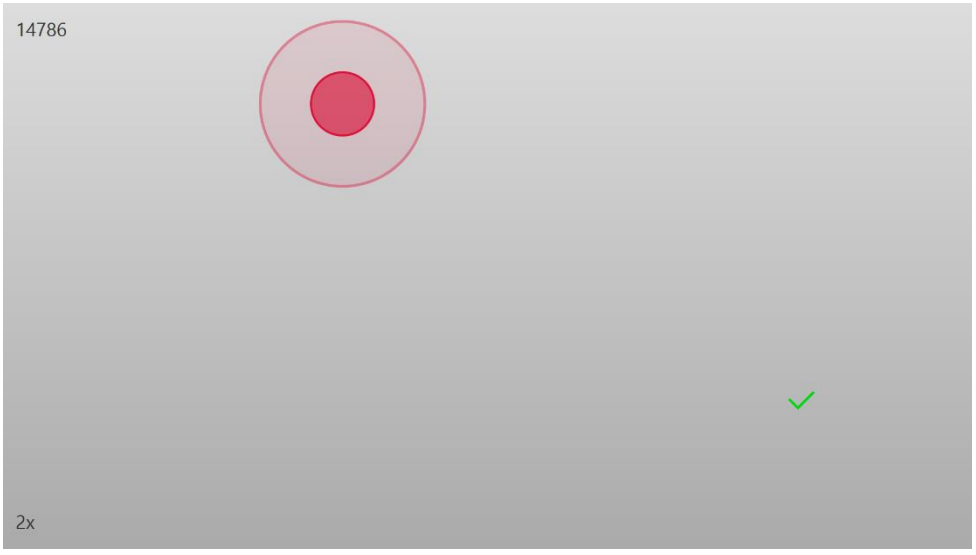
Jakmile terč zmizí, zobrazí se buď zelená fajfka, nebo červený křížek podle toho, jestli hráč terč trefil nebo ne. Pro vykreslení obrázku je potřeba třídy *DrawMark*, která v konstruktoru nadefinuje pozici a velikost obrázku. Poté se musí zavolat metoda *draw*, která rozhodne, jaký obrázek má vykreslit. Tento obrázek se vykreslí pomocí tří animací *FadeTransition*. První obrázek postupně zobrazí (snižuje průhlednost), druhá po určitou dobu nechá obrázek zobrazený (zachová maximální neprůhlednost) a třetí postupně obrázek nechá zmizet (zvyšuje průhlednost).

2.2 Mapy

Na začátek je důležité říct, co vlastně mapa je. Nejedná se totiž o klasickou mapu v programátorském smyslu typu *HashMap* nebo *TreeMap*. V této hře mapa představuje všechny terče, které se postupně objevují na obrazovce.

Každá mapa má své vlastní nastavení, které určuje její náročnost nebo grafický vzhled. Nastavení mapy je uloženo ve třídě *MapSettings*. Proměnná *approachTime* představuje rychlost, jakou se vnější kruh přibližuje. Její hodnota se pohybuje v rozmezí od 1 do 5, kde s rostoucí hodnotou se přibližovací kruh zrychluje. Další proměnná *time* určuje délku mapy ve vteřinách. Tato proměnná se uživatelem nastavuje pouze v náhodně generované mapě. U vlastní mapy se totiž délka mapy určuje podle počtu terčů a podle následující proměnné *speed*. Tato proměnná určuje, jak rychle po sobě se jednotlivé terče generují. Může nabývat hodnoty od 1 do 10 a opět s rostoucí hodnotou se terče generují rychleji po sobě. Ve vlastních mapách lze tuto proměnnou přepsat pomocí proměnné *interval*, která může pro daný terč určit vlastní rychlost vygenerování. Mapě lze nastavit také velikost terčů proměnnou *size*, která opět může mít hodnotu od 1 do 10 a taktéž 10 představuje největší terč. Ve vlastních mapách představuje základní hodnotu terče, pokud daný terč nemá předepsanou hodnotu *radius*. Poslední nastavení mapy jsou barvy terčů. Pole *colors* obsahuje barvy, kterými mohou být terče náhodně vykresleny.

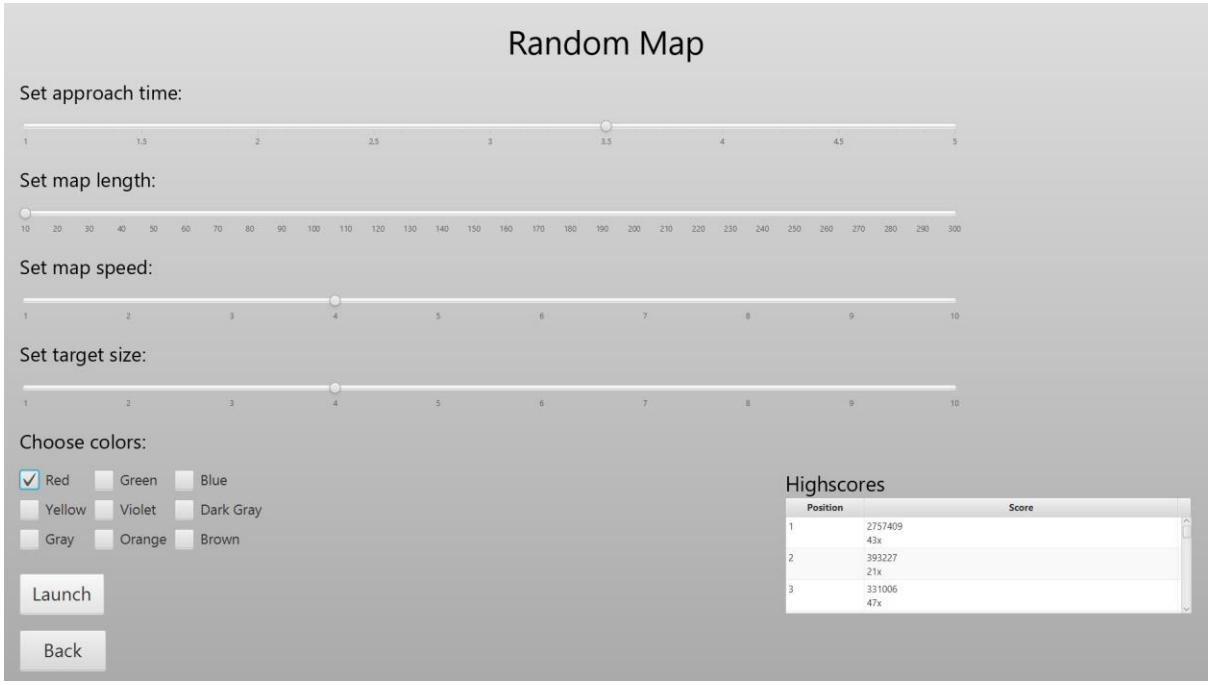
Mapy jsou ve hře zastoupeny třídami *RandomMap* a *CustomMap*, které jsou spustitelné, a právě při jejich spuštění začne postupné generování terčů. Generování probíhá v novém vlákně. Ve vlastní mapě je toho nezbytné, jelikož se vlákno uspává po vytvoření každého terče na dobu určenou proměnnou *interval*. Jak mapa nakonec vypadá, je zobrazeno na obrázku 2.2. V levém horním rohu je aktuální skóre a v levém dolním horním je aktuální kombo.



Obrázek 2.2: Mapa

2.3 Náhodně generovaná mapa

První herní mód je založen na náhodně generované pozici terčů. Jakmile uživatel v hlavním menu vybere tento mód, musí zvolit nastavení pro generování terčů. Nastavení obtížnosti mapy upravuje pomocí *Slideru* a barvy terčů pomocí *CheckBoxu* (viz obrázek 2.3). Na této obrazovce má také tabulku s nejlepšími dosaženými výsledky v tomto módu. U každého skóre lze vidět pozice v tabulce, dosažené skóre a nejvyšší kombo (největší počet terčů trefených v řadě). Pro podrobné informace ohledně nějakého skóre může hráč přejet kurzorem přes dané skóre a zobrazí se mu popisek. V něm může hráč dále vidět celkový počet trefených terčů, celkový počet vynechaných terčů a datum dosažení tohoto skóre.

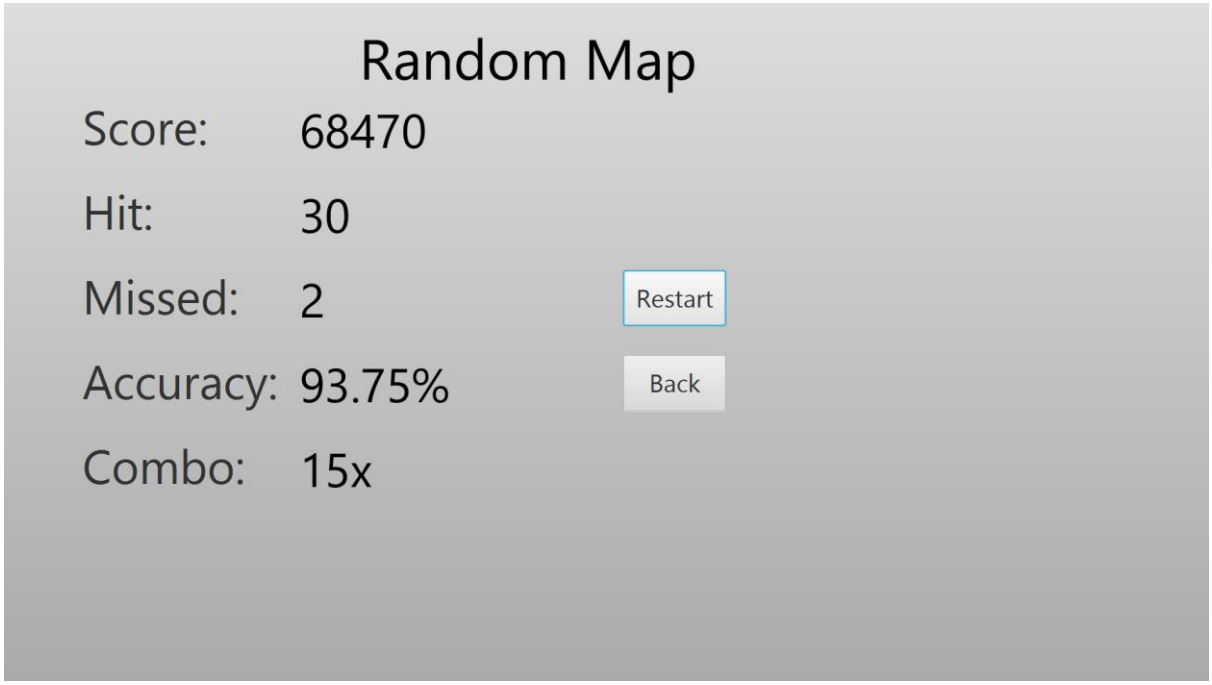


Obrázek 2.3: Nastavení náhodně generované mapy

Princip náhodně generovaných terčů závisí na náhodném generování jejich souřadnic. Při kliknutí na tlačítko spustit se ze zadaných hodnot ve *Sliderech* definuje nastavení mapy a spustí se třída *RandomMap*. Nejdříve se zobrazí odpočet, za jak dlouho se začnou zobrazovat terče. Odpočet je graficky doplněn o *FadeTransition* pro příjemnější mizení čísel. Terče jsou následně generovány pomocí *Timeline*. V každém jejím cyklu se spustí metoda *generateRandom*, ve které se vytvoří nový terč. V této metodě se náhodně definují souřadnice terče. Terč se musí nacházet v oblasti obrazovky tak, aby například půlka terče nebyla mimo obrazovku. Proto jsou definované minimální a maximální hodnoty souřadnic x a y, které vytváří pomyslný okraj pro terče, za kterým se terče nesmí vykreslovat. S každým terčem se zde také vytvoří přibližovací kruh. Dále pomocí *PauseTransition* se nastaví, aby jakmile se

vnější kruh přiblíží k terči, tak terč zmizí a vynuluje se kombo. Ale to se provede pouze, pokud terč ještě existuje. Terč totiž může zmizet tím, že se na něj klikne a v tom případě *PauseTransition* neprovede svůj kód při dokončení. Při kliknutí na terč se zaznamená čas v milisekundách, jak dlouho hráči trvalo kliknout na terč, který se poté používá při počítání skóre.

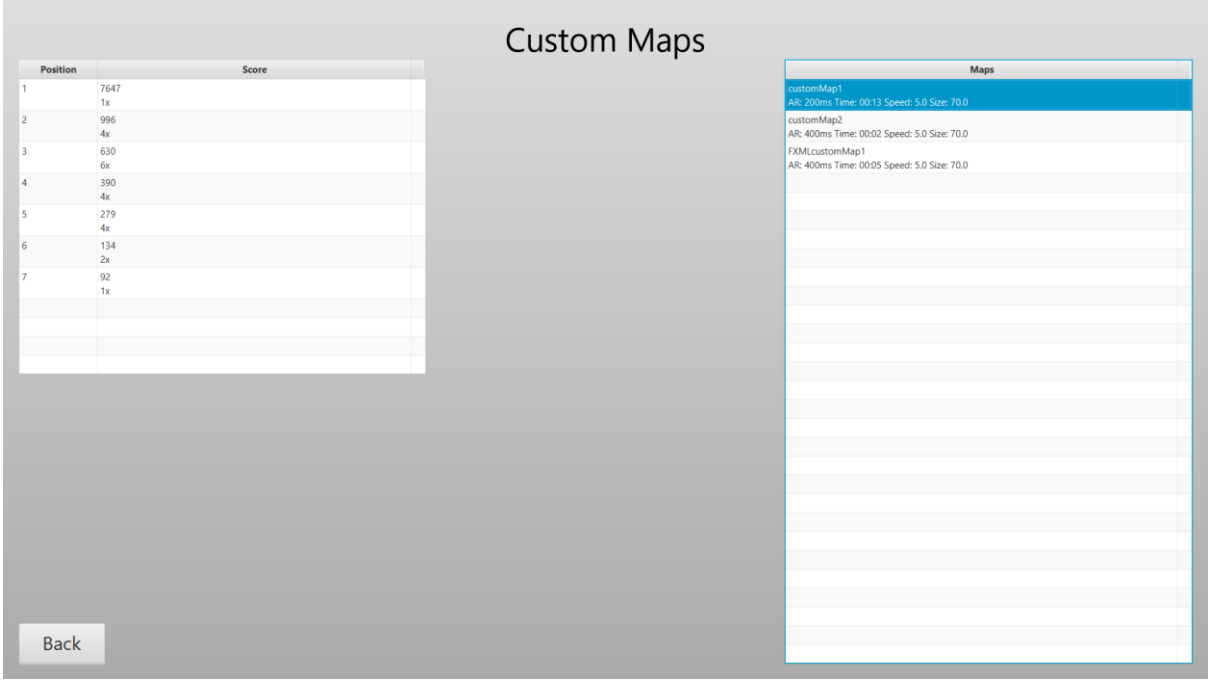
Po postupném vygenerování všech terčů se zaznamená dosažené skóre a další detaily o výsledku a přepne se na obrazovku s mým dosaženým výsledkem (viz obrázek 2.4). Hráč má následně možnost opakovat mapu se stejným nastavením nebo se vrátit do hlavní nabídky.



Obrázek 2.4: Obrazovka s dosaženým výsledkem

2.4 Vlastní mapa

Chce-li hráč hrát vlastní mapu, musí si nejprve nějakou vytvořit nebo stáhnout. Po vybrání módu vlastní mapy se na pravé straně zobrazí tabulka se všemi dostupnými mapami. U každé mapy vidí název a její základní vlastnosti (přibližovací rychlost, délku, rychlost mapy a velikost terčů). Prvky této tabulky jsou instance třídy *CustomMapProp*. Data v tabulce potřebují *SimpleStringProperty*, který tato třída vytvoří. Tato třída také obsahuje metodu, která spustí danou mapu. Pokud hráč klikne v tabulce na nějakou mapu, zobrazí se vlevo tabulka s výsledky. Ta vypadá úplně stejně jako tabulka u náhodně generované mapy, obsahuje ale nejlepší dosažené skóre vybrané vlastní mapy. Pokud hráč klikne na mapu dvakrát, tak se vybraná mapa spustí. Obrazovka s vlastními mapami je ukázána na obrázku 2.5.



Obrázek 2.5: Obrazovka s vlastními mapami

Princip generování terčů ve třídě *CustomMap* je téměř stejný jako u náhodně generovaných map (viz. Kapitola 2.3). Je tu ovšem pár nutných přizpůsobení pro vlastní mapy. Jedním z nich je způsob, kterým se terče zobrazovaly. Zde se namísto *Timeline*, používá obyčejné pole s instancemi třídy *Target*. Tato třída dědí třídu *Circle* a obsahuje ještě další proměnnou *interval*. Pro vytváření terčů třída *CustomMap* prochází každý prvek pole, kde se dodatečně zeptá, jestli je proměnná *interval* definovaná a pokud není, tak mu přidá výchozí *interval* mapy. Zobrazování terčů je také trochu jiné. Terč by měl mít nezbytné vlastnosti definované už ze souboru vlastní mapy, ale pokud mu některé chybí, dosadí se místo nich výchozí nastavení mapy. U pozice terče je to trochu složitější, protože v některých SceneBuilderech nelze nastavit *CenterX* a *CenterY*. Místo toho se jim nastavuje *LayoutX* a *LayoutY*, a proto je nutné tyto souřadnice předat. Nakonec po každém vytvoření terče se vlákno uspí na dobu určenou proměnnou *interval*.

Stejně jako u náhodně generovaných map se po vytvoření všech terčů uloží dosažené skóre a přejde se na výslednou obrazovku. Každá vlastní mapa si ukládá své skóre do vlastního souboru s nejlepšími dosaženými výsledky, aby nedošlo ke sloučení výsledků ze dvou odlišných vlastních map.

3 Algoritmy

3.1 Výpočet celkového skóre

Skóre jako nejdůležitější ukazatel úspěšnosti hráče se počítá podle několika různých parametrů. Základním prvkem při výpočtu je rychlost, za jakou hráč na terč kliknul. Cílem je kliknout na terč co nejrychleji, a proto tato hodnota zásadně ovlivňuje dosažené skóre. Tato rychlost se počítá podle rozdílů dvou časů. První čas je vytvoření terče a druhý kliknutí na terč. Tato hodnota je v milisekundách, a aby za co nejnižší hodnotu bylo největší skóre, tak se z ní vypočítá převrácená hodnota. Jenže to je příliš nízká hodnota na to, aby měla ve skóre nějakou váhu, a proto je ještě vynásobena tisícem a uložena do proměnné *base* (viz obrázek 3.1).

```
double base = (1.0/reaction) * 1000;
```

Obrázek 3.1: Výpočet základu skóre

Dále se do skóre musí zahrnout obtížnost mapy, proto se pro každou hodnotu musí vypočítat nějaká hodnota charakterizující její obtížnost. Kdyby tento prvek nebyl zahrnut do výpočtu skóre, tak by hráč mohl zapnout naprosto jednoduchou mapu a dosáhnout vyššího skóre než jiný hráč, který by zapnul jinou mapu, při které by musel prokázat mnohem větší dovednosti. Obtížnost každé mapy určuje při výpočtu proměnná *multiplier*. Ta se počítá podle rychlosti přibližovacího kruhu, rychlosti mapy a velikosti terčů, tedy proměnných *approachTime, speed* a *size* respektive. Zde je opravdu těžké snažit se vybalancovat každý parametr, aby měly odpovídající váhu ve vzorci. Pokusil jsem se, aby to bylo elementárně vyrovnané, ale pravděpodobně pro úplné vybalancování budou potřeba další a podrobnější testy. Finální vzorec pro výpočet obtížnost je tedy na obrázku 3.2.

```
multiplier = speed * (500/size) * (3000/approachTime);
```

Obrázek 3.2: Výpočet číselné hodnoty odpovídající obtížnosti mapy

Poslední proměnná, která se při výpočtu skóre používá, je poměrně obyčejná. Hra musí hráče nějak potrestat za to, že terč netrefí, jinak by na tom v podstatě nezáleželo. Proto je tu proměnná *combo*. Tato proměnná představuje počet trefených terčů v řadě. Pokud hráč terč mine, tak se tato proměnná vyresetuje na nulu. Naopak při každém trefeném terči se tato hodnota zvýší o jedna.

```
int i = score.get() + (int) (base * multiplier * combo.get());
score.set(i);
```

Obrázek 3.3: Výpočet skóre, které se přičte při trefení terče

Skóre se tedy získá vynásobením všech těchto tří proměnných (viz obrázek 3.3). Tento výpočet proběhne po každém trefení terče a ukládá do *SimpleIntegerProperty*. Neukládá se do *Integeru*, protože skóre se vypisuje na obrazovce a musí se pokaždé aktualizovat. A *Integer* není schopný při každé změně hodnoty změnit i zobrazenou hodnotu.

3.2 Ukládání výsledků

Každá vlastní mapa má svůj vlastní soubor, kde jsou uloženy všechny výsledky. Pro náhodně generované mapy je soubor pouze jeden. Výsledky jsou ukládány do souborů typu CSV, protože tento formát je poměrně jednoduchý a čitelný. Jeden řádek představuje jeden výsledek a na každém řádku jsou čárkami rozdělené detaily výsledku.

Po ukončení mapy jsou výsledky nejprve ukládány jako instance třídy *Result*, v jejímž konstruktoru se nadefinují detaily výsledku. Ukládají se zde proměnné *hit* a *missed*, které značí počet trefených a netrefených terčů. Z těchto proměnných se dále vypočítá proměnná *accuracy*, která představuje procentuální přesnost trefených k minutým terčům. Dále se zde definují dvě pravděpodobně nejdůležitější proměnné a to *score* a *highestCombo*, které pochopitelně obsahují dosažené skóre a nejvyšší dosažené kombo. Poslední proměnná, která se zde ukládá je *timeAchieved* a ta ukládá čas, kdy bylo dosaženo tohoto výsledku. Čas ukládá jako počet milisekundách od 1. 1. 1970.

```
public void addResultToFile(Result res){
    //prida dosazeny vysledek na konec souboru
    PrintWriter writer = null;
    String output = res.toString();
    try {
        File f = new File(path);
        writer = new  PrintWriter(new FileWriter(f, append: true));
        writer.append(output + "\r\n");
    } catch (IOException ex) {
        Logger.getLogger(ResultScreen.class.getName()).log(Level.SEVERE, msg: null, ex);
    } finally {
        writer.close();
    }
}
```

Obrázek 3.4: Metoda, která ukládá výsledek do souboru se všemi výsledky na dané mapě

Do souboru se výsledek ukládá ve třídě *ResultScreen*, která také zodpovídá za vykreslení obrazovky s výsledkem. Hned v konstruktoru, tedy ještě předtím než se tato obrazovka zobrazí, se zavolá metoda *addResultToFile*, ale pouze pokud je skóre nenulové (viz obrázek 3.4). Nechceme přece ukládat skóre, kde hráč netrefil ani jeden terč. To by zbytečně zaplňovalo místo v souboru nulovými záznamy. Tato metoda si zjistí, do jakého souboru má skóre uložit. Všechny soubory s výsledky se nacházejí ve složce *highscores*. Soubor s výsledky vlastních map jsou pojmenovány podle názvu mapy. Soubor s výsledky náhodně generované mapy se jmenuje „*random_highscores*.csv“. To, co se má do tohoto souboru uložit, metoda získá pomocí metody *toString* třídy *Result*, která je přepsaná tak, aby přímo dodržovala syntaxi formátu CSV. To znamená, že vrátí všechny proměnné oddělené čárkou. Do souboru se poté každý tento textový řetězec zapisuje na konec souboru na nový řádek. Obsah souboru tedy může vypadat nějak jako na obrázku 3.5.

```
1  5,4,1627,3,1615282215599
2  1,5,376,1,1615285942870
3  2,4,60,2,1615287134730
4  8,0,77318,8,1615290662488
5  8,0,80065,8,1615290683576
6  5,2,23282,5,1615290778743
7  1,7,178,1,1615294744563
8  7,2,35230,6,1615295158347
9  2,5,5019,1,1615295172859
10 3,4,8013,2,1615295312742
11 5,4,678,3,1615538110571
12 14,5,21200,9,1615538203035
13 5,4,3917,4,1615538428346
14 7,1,8031,4,1615538442698
15 8,1,7991,6,1615538596479
16 2,5,261,2,1615543492879
17 5,4,7154,5,1615543734110
18 4,7,197,1,1615544984210
19 6,3,602,4,1615544997555
20 5,5,396,2,1615545160711
21 1,7,80,1,1615545323062
22 3,8,38,1,1615545431396
23 3,6,66,1,1615545490248
24 3,6,61,2,1615545747374
25 4,5,309,2,1615545778078
26 3,5,101,1,1615545805246
```

Obrázek 3.5: Obsah souboru „random_highscores.csv“. Na jednom řádku proměnné oddělené čárkami představují počet trefených terčů, počet netrefených terčů, dosažené skóre, nejvyšší kombo a čas dosažení skóre.

Aby bylo možné nějak výsledky navzájem porovnávat, musí třída *Result* implementovat třídu *Comparable*. Z té si převezme metodu *compareTo*, která porovnává číselné hodnoty dosaženého skóre. Tato metoda se používá při výpisu nejlepších dosažených skóre v tabulce, protože když se výsledky získávají ze souboru, tak potřebují ještě setřídit podle nejlepšího dosaženého skóre. Výsledky se pokaždé zapisují na konec souboru, takže by se dalo říct, že jsou seřazené podle dosaženého času.

3.3 Načítání vlastních map

Dlouho jsem se rozhodoval nad tím, jakým způsobem budou vlastní mapy vytvářeny. Nejdříve mě napadlo, že by byly vytvářeny přímo v aplikaci, jenže to by bylo složité ještě vymýšlet další UI, kde by hráč pravděpodobně pomocí drag-and-drop systému přemisťoval kruhy a upravoval jejich vlastnosti jako velikost nebo čas zobrazení. Poté jsem si vzpomněl na pomůcku JavaFX, která toto vlastně už umí, a to SceneBuilder. Ten je právě založen na tomto systému a každému objektu může nastavit pozici, velikost atd. Tento návrh se zapisuje do souboru typu FXML, což je značkovací jazyk založený na XML. Značky (tagy) v FXML mohou být třídy z JavaFX například *BorderPane* nebo *Button*. Na vytváření terčů jsem si ale musel vytvořit vlastní značku, pro kterou jsem musel vytvořit a importovat třídu *Target* (viz obrázek 3.6). Ta je potomkem třídy *Circle* a má ještě dodatečnou proměnnou *interval*, aby se přímo v SceneBuilderu nebo při psaní kódu dala nastavit rychlost vytvoření pro každý terč zvlášť. Všechny ostatní důležité vlastnosti terče je možné nastavit díky třídě *Circle*.

```
public class Target extends Circle{
    private int interval;

    public int getInterval() { return interval; }

    public void setInterval(int interval) { this.interval = interval; }

}
```

Obrázek 3.6: třída *Target*, která je potomkem třídy *Circle*

Vlastní mapy se ukládají do složky *maps* jako FXML soubory. Při vypsání všech vlastních map do tabulky projde třída *CustomMapSettings* všechny soubory této složky a vybere pouze ty s koncovkou „.fxml“. Název vlastní mapy se definuje podle názvu souboru. Dále se tento soubor načte jako scéna a zpracuje se třídou *CustomMapController* (viz obrázek 3.7). Ta implementuje interface *Initializable* a uloží si nastavení mapy a do pole si uloží všechny terče. S těmito parametry se vytvoří instance třídy *CustomMap* a s tou poté program dále pracuje.

```
@Override
public void initialize(URL url, ResourceBundle rb) {
    //ziskani informaci o mape ze souboru fxml
    approachTime = (int) (2000.0 / Double.parseDouble((String) pane.getProperties().get("approachTime")));
    speed = Double.parseDouble((String) pane.getProperties().get("speed"));

    //defaultni velikost pro terce, ktere nemaji preddefinovanou vlastni velikost
    size = (Double.parseDouble((String) pane.getProperties().get("size")) * 10) + 10;

    colors = getColors(((String) pane.getProperties().get("colors")).split( regex: ","));
    MapSettings settings = new MapSettings(approachTime, time, speed, size, colors);
    children = pane.getChildren();

    cm = new CustomMap(settings, children);
}
```

Obrázek 3.7: metoda *initialize* třídy *CustomMapController*

4 Tvorba vlastní mapy

Vytvořit vlastní mapu lze poměrně jednoduše. Uživatel si může práci ještě usnadnit pomocí SceneBuilderu. Soubor FXML nejdříve musí importovat knihovnu *Target* a libovolný JavaFX layout, kterým může být například obyčejný *Pane*. Tento layout musí mít nastavený parametr *fx:controller*, který musí odkazovat na třídu *CustomMapController*. Layout dále musí mít definované *properties*, které obsahují proměnné *approachTime*, *colors*, *size* a *speed*. Všechny proměnné slouží jako výchozí hodnoty pro terče, které tuto hodnotu samy nepřepisují. Jediná proměnná, která nelze přepsat je *approachTime*. *Properties* se musí nadefinovat manuálně, protože SceneBuilder ve svém UI nedokáže vytvořit a zobrazit vlastní proměnné. *ApproachTime*, *size* a *speed* mohou nabývat libovolných kladných reálných čísel. Do proměnné *colors* lze ukládat barvy buď hexadecimálním kódem ve formátu „#FF00FF“ nebo anglickým názvem podporovaných barev. Podporované barvy jsou red, green, blue, yellow, violet, gray, dark gray, orange a brown. Barvy musí být v proměnné oddělené čárkami bez mezer.

Terče se musí nacházet ve vybraném layoutu. Každý terč se definuje jako element *Target*. Každému terči se musí nastavit atributy *centerX* a *centerY*, které určují pozici jeho středu. Pokud některý SceneBuilder nedokáže definovat *centerX/Y*, stačí definovat *LayoutX/Y* a program si tyto souřadnice sám předá. Dále mu lze nastavit 3 volitelné atributy *fill*, *interval* a *radius*. *Fill* umožňuje obejít náhodně vybírání barvy terče z pole s výchozími barvami a může mu nastavit naprosto libovolnou. Hodnota je hexadecimální kód vybrané barvy. *Interval* je počet milisekund, které vlákno počká, než vykreslí další terč. *Radius* na rozdíl od *size* nenabývá hodnot 1 až 10. Tato proměnná určuje poloměr terče v pixelech a má přednost před proměnnou *size*. Terče se na obrazovku vykreslují v pořadí, ve kterém jsou v souboru napsány. Jak takový FXML soubor může vypadat je ukázáno na obrázku 4.1.

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.layout.*?>
<?import target.*?>

<Pane fx:id="pane" xmlns:fx="http://javafx.com/fxml/1"
      xmlns="http://javafx.com/javafx/11.0.1" fx:controller="aimtrainer.CustomMapController">
<properties approachTime="5" colors="green,red" size="6.0" speed="5.0" />
    <Target fill="#790606" interval="3000" centerX="290.0" centerY="200.0" radius="128.0" />
    <Target centerX="500" centerY="500" />
    <Target centerX="600" centerY="700" />
</Pane>
```

Obrázek 4.1: FXML soubor pro vlastní mapu, která obsahuje 3 terče. První z nich přepisuje výchozí hodnoty mapy, zatímco zbylé dva výchozí hodnoty mapy přebírají.

5 Závěr

Tento projekt naplnil má očekávání. Hra funguje přesně tak, jak jsem si to původně naplánoval. Nenarazil jsem na žádné větší překážky, které by výrazně překazily mé původní plány.

Během vývoje mě napadlo mnoho dalších vylepšení, které bych v budoucnu mohl zaimplementovat. Mohl bych například zkusit přidat mnohem větší možnosti přizpůsobení hry každému hráči jako například přidání vlastního obrázku jako pozadí mapy nebo různé zvukové efekty a přehrávání hudby při hraní mapy. Uživatelské rozhraní je nyní velmi přehledné, ale určitě by se s ním dalo ještě pohrát a zdokonalit ho. Pro dosažené výsledky by se dala vytvořit webová databáze, ze které by hra stahovala nejlepší skóre a porovnávala je s ostatními hráči po celém světě. Také by se dala vytvořit stránka, kde by si hráči mezi sebou mohli sdílet vlastní mapy.

Tento projekt má dle mého názoru velký potenciál a s dalšími zejména grafickými úpravami by se mohl chytnout nejen mezi hráči počítačových her, ale i u běžných uživatelů.

6 Bibliografie

- ScaleTransition (JavaFX 2.2). (2013, December 09). Retrieved March 22, 2021, from https://docs.oracle.com/javafx/2/api/javafx/animation/ScaleTransition.html
- FadeTransition (JavaFX 2.2). (2013, December 09). Retrieved March 22, 2021, from https://docs.oracle.com/javafx/2/api/javafx/animation/FadeTransition.html
- Java thread sleep() method with examples - javatpoint. (n.d.). Retrieved March 22, 2021, from https://www.javatpoint.com/java-thread-sleep-method
- JavaFX controller: Learn How Does FXML controller work in JavaFX? (2021, March 04). Retrieved March 22, 2021, from https://www.educba.com/javafx-controller/
- Javainterviewpoint. (n.d.). How to export data into a csv file. Retrieved March 22, 2021, from https://www.javainterviewpoint.com/how-to-export-data-into-a-csv-file/
- Release: Javafx 2.2.40. (2013, August 30). Retrieved March 22, 2021, from https://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm
- SequentialTransition (JavaFX 8). (2015, February 10). Retrieved March 22, 2021, from https://docs.oracle.com/javase/8/javafx/api/javafx/animation/SequentialTransition.html
- How to control the JAVAFX Tooltip's delay? (2014, November 10). Retrieved March 22, 2021, from https://stackoverflow.com/questions/26854301/how-to-control-the-javafx-tooltips-delay
- Adding custom classes to fxml files. (2017, March 04). Retrieved March 22, 2021, from https://stackoverflow.com/questions/42602186/adding-custom-classes-to-fxml-files