Gymnázium, Praha 6, Arabská 14

Obor programování

Maturitní práce

Jakub Kučera

Hra z pohledu třetí osoby

březen 2021

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, Arabská14 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 31. března 2021

Název práce: Hra z pohledu třetí osoby

Autor: Jakub Kučera

Anotace: Mým projektem je programování hry v herním engine *Unity*, který podporuje programovací jazyk *C#*. Hra je cílena na platformu *Android*. Hráč ovládá svoji postavu z pohledu třetí osoby. Postava se dokáže pohybovat, útočit a vyhýbat se útokům. Hráč také může spravovat inventář s předměty, které si zakoupil v obchodě.

Mapy, na kterých hráč hraje, jsou generovány náhodně s různými úkoly ke splnění. Na poslední mapě se vyskytuje boss, kterého hráč potřebuje porazit. Nepřátelé jsou implementováni jako konečné automaty. Nepřátelé vyhledávají cestu k hráči pomocí *A* * algoritmu.

Title: Game from a third-person perspective

Author: Jakub Kučera

Abstract: My project is programming of a game in *Unity* game engine, which supports programming language *C#*. Game is aimed at *Android* platform. Player controls his character from a third-person perspective. Character can move, attack and dodge attacks. Player also manages his inventory with items, which he has bought in a shop.

Maps, on which player plays, are randomly generated with different objectives to complete. There is a boss, which the player needs to defeat, on the last map. Enemies are implemented as a finite-state machines. Enemies search for the path to the player with *A* * algorithm.

Zadání projektu

Mým ročníkovým projektem je vytvoření mobilní 3D hry pro platformu Android v herním engine Unity, který podporuje programovací jazyk C#.

Hráč hraje za postavu, kterou ovládá z pohledu třetí osoby. Postava se například dokáže pohybovat, útočit nebo sbírat předměty na mapě. Součástí hry je také hráčův inventář, ve kterém si může nasadit již získané předměty.

Hráč bojuje proti nepřátelským postavám ovládaným počítačem v náhodně generovaných mapách. Nepřátelé jsou implementováni jako konečné automaty a vyhledávají cestu pomocí *A* * algoritmu.

Cílem hry je porazit finálního bosse ve hře.

Obsah

1. Úvod

Rád hraji počítačové hry, a proto jsem se rozhodl vytvořit vlastní hru. Platforma *Android* byla zvolena, protože se mi zdá, že na ní není mnoho zajímavých her. Vybral jsem si herní engine *Unity*, protože jsem v něm již několik menších projektů vytvořil a také umožňuje vytváření mobilních her.

Hráč ovládá postavu rytíře z pohledu třetí osoby. Postava je ovládána prostřednictvím grafického rozhraní, které se skládá z tlačítek a joysticku. Postava se dokáže pohybovat, vyhýbat útokům, skákat, útočit a sbírat předměty. Hráč může v inventáři měnit vybavení, které mění nejen hodnoty atributů postavy ale i její vzhled.

Nepřátelé neútočí, dokud si hráče nevšimnou, a poté ho pronásledují. K vyhledání cesty k hráči používají *A* * algoritmus. Když se k hráči přiblíží na dostatečnou vzdálenost, tak na něj zaútočí a udělí hráči poškození. Na konci hry se hráč setká s bossem, který má několik různých útoků.

Mapy jsou náhodně generovány v několika krocích a skládají se z předem vytvořených částí místností. Mapy ve hře obsahují různé úkoly ke splnění.

2. Vlastnosti Unity

Projekt se skládá ze scén, což je v podstatě prostředí, do kterého se umisťují předměty typu *Game Object*. *Game Object* je objekt ve scéně, na který lze umístit komponenty a skripty. Většina skriptů je potomkem třídy *Mono Behaviour*, která nabízí funkce pracovního cyklu herního engine *Unity*. Dále skripty mohou být potomky třídy *Scriptable Object*. Ta umožňuje ukládání dat, která mají stálou hodnotu. V projektu také používám *Particle System* (doslovně přeloženo jako systém částic), který zjednodušuje vytváření vizuálních efektů. Další zvláštností v *Unity* jsou metody typu *Coroutine*, které umožňují pozastavit v nich prováděný kód.

3. Struktura projektu

Veškerý kód, modely a textury jsou ve složce *Assets*, a proto popíšu pouze strukturu v ní. Soubory s příponou souboru *.meta* jsou nutné ke správnému fungování hry v prostředí editoru a jejímu sestavení.

- **Animation Controllers** – zde se nachází ovladače animací hráče a nepřátel. Ty určují, které animace u postav se mají přehrávat.
- **Editor** – složka obsahuje skripty, které nebudou obsaženy ve hře a slouží k ulehčení práce ve vývojovém prostředí. Je v ní skript s převzatou metodou (glitchers, 2021).
- **Joystick Pack** – obsahuje joystick, který používám a je převzatý z *Unity Asset Store*. (Fenerax Studios, 2021)
- **Materials** – zahrnuje materiály.
- **Models** – tento adresář obsahuje 3D modely, textury a jejich normálové mapy.
- **Prefabs** – složka obsahuje objekty, které se označují pojmem „*prefab*“. Jsou to v podstatě vzory objektů, podle kterých se během hry vytváří instance objektů.
- **Resources** – v této složce jsou umístěny objekty, které nelze přiřadit ke konkrétnímu objektu, např. protože ten objekt ještě nebyl vytvořen. Jinými slovy jsou načítány během hry pomocí relativní cesty ve složce *Resources*.
- **Scenes** – obsahuje všechny scény v projektu.
- **Scriptable Objects** – obsahuje skripty typu *Scriptable Object*.
- **Scripts** – v této složce se nachází většinu kódu až na skripty typu *Scriptable Object*.
- **Sprites** – tento adresář obsahuje 2D ikony.
- **TextMesh Pro** – obsahuje balíček sloužící k vylepšení grafického rozhraní, který je součástí *Unity*.

4. Přehled skriptů

Stručný přehled nejdůležitějších skriptů, které jsou součástí hry:

- **Boss**

  Boss funguje podobně jako nepřátelé, to znamená, že je také implementovaný jako konečný automat. Na rozdíl od nepřátel má více druhů útoků a lze vytvářet jejich další kombinace v editoru. Boss také řeší kolize a působí na něj gravitace.

- **Camera Controller**

  Ovládá kameru. Skript je odpovědný za pohyb a rotaci kamery. Kamera vždy zachycuje hráčovu postavu a je nasměrovávána podle okolností. Hráč může kameru ovládat dotykem, zaměřit kameru na nepřátele, nebo se sama otáčí podle pohybu postavy. *Camera Controller* dostává vstup ze skriptu *Input Manager*.

- **Cell Generator**

  Vygeneruje v obdélníkovém (případně čtvercovém) poli buňky tak, aby všechny byly mezi sebou propojené.

- **Enemy Controller**

  Nepřítel se v každém momentu vyskytuje pouze v jednom stavu, jinými slovy je to konečný automat. Mezi jeho stavy patří pronásledování hráče, útočení, rozhlížení se po hráči atd. Nepřítel je ovlivňován gravitací a řeší kolize.

- **Enemy FSM**

  Skript je zodpovědný za stav protivníka. Obsahuje metody, které jej mění, ukončují starý stav a začínají nový stav. Také volá pravidelně se opakující metody v jednotlivých stavech.

- **Game Manager**

  Je to *singleton*, který má na starosti načítání scén a uchovávání odkazů k důležitým objektům jako jsou správci právě načtených scén nebo k databázi předmětů.

- **Game Physics**

  Je to statická třída obsahující metody, které řeší pohyb a rotaci objektů, kolize mezi nimi a také gravitační síly působící na postavy ve scéně. Výše uvedené věci jsou řešeny více způsoby a je použit ten, který se hodí v dané situaci.

- **Hub Manager**

  Aktivuje portály a obchod a posílá jim nutné odkazy. Také otevírá portály podle postupu hráče.

- **Input Manager**

  Sbírá hráčovy vstupy spojené s ovládáním postavy. Také předává vstupy skriptům *Camera Controller*, potřebné k jejímu pohybu a rotaci, a *Player Controller*.

- **Maze Generator**

  Řídí vytváření mapy. Volá jiné skripty v daném pořadí, aby vznikla mapa a předává skriptům odkazy na objekty vytvořené během předchozích kroků.

- **Maze Manager**

  Poté co hráč vstoupí do portálu a je načtena scéna s mapou, začne vytvářet mapu a taky je vytvořena podmínka ke splnění mapy podle nastavení, které je skriptu předáno.

- **Menu Manager**

  Spravuje grafické rozhraní, které uživatel uvidí, když spustí hru. V menu může uživatel spravovat uložené hry a nastavení.

- **Pathfinding**

  Umí nalézt nejkratší cestu z bodu A do bodu B pomocí *A* * algoritmu. Také dokáže vygenerovat částečně náhodnou cestu.

- **Pathfinding Node Generator**

  Skript vytvoří v každé pod-buňce mřížku s vrcholy sloužícími k vyhledávání cesty.

- **Player Controller**

  Skript odpovídající za pohyb a akce postavy. Přetváří vstup hráče na akce postavy. Během pravidelných intervalů, určených *Unity*, počítá pozici a rotaci postavy. Také ovládá její animace. *Player Controller* dostává vstup ze skriptu *Input Manager*.

- **Spawner**

  Vytváří instance objektů, jako jsou například nepřátelé nebo sbíratelné předměty na zemi.

- **Subcell Generator**

  Rozdělí buňky na menší celky, které lze označit jako „pod-buňky“. Tyto menší celky jsou buď uspřádány ve tvaru místnosti nebo tvoří chodbu.

- **Tile Generator**

  Je zodpovědný za většinu vizuální stránky mapy. Poskládá dílky tak, aby dohromady tvořily uzavřenou oblast.

5. Scény

Následující podkapitoly popisují, co vše je možné ve hře dělat. Každá podkapitola popisuje samostatnou scénu, ze kterých se hra skládá. Hra se skládá ze čtyř scén: *Menu* (hlavní menu), *Player* (scéna s hráčem), *Hub* (výběr úrovní) a *Maze* (mapa). Při otevření aplikace je spuštěna scéna *Menu*. V *Unity* je možné mít více načtených scén najednou.

5.1 Menu

Po zapnutí hry, se hráč ocitne v menu, ze kterého může přejít na výběr uložených her, vstoupit do nastavení nebo hru vypnout. V nastavení si lze upravit rychlost otáčení obrazovkou pro vertikální i horizontální osu zvlášť. Výběr her obsahuje textové pole pro zadání názvu nové uložené hry a tlačítko pro její vytvoření. Když jsou na zařízení již uložené postupy ve hře, tak se zobrazí pro každý postup tlačítko s jeho názvem, které načte tento uložený postup, a také tlačítko, které postup smaže ze zařízení. Během změny úrovní se zobrazuje načítací obrazovka.

5.2 Player

Tato scéna obsahuje hráčovou postavou a také grafické rozhraní s tlačítky a joystickem k ovládání postavy a hráčovým inventářem. Fungování hráčovy postavy a inventáře je podrobněji popsáno v samostatných kapitolách. Tato scéna je načtena spolu se scénami *Hub* nebo *Maze*. Ve scéně není žádné prostředí, a tedy nemůže být načtena samotná.

5.3 Hub

Ve hře není výběr úrovní ztvárněn jako nudné klikání tlačítek s čísly, ale jako portály, do kterých hráč vstupuje. Další portál se zpřístupní, když hráč splní úkol v předchozím portálu. Aktivovaný portál se odlišuje tím, že se v něm točí barevný kruh, který je tvořen systémem částic. V editoru lze ke každému portálu přiřadit nastavení, které popisuje vlastnosti mapy k vygenerování. Ve výběru úrovní se kromě portálu nachází obchodník, který hráči nabízí předměty ke koupi za určitou cenu.

5.4 Maze

V této scéně je jediný *Game Object*, na kterém jsou umístěny skripty, které vygenerují mapu. Mezi zmíněné skripty patří například *Maze Manager*, *Cell Generator*, *Subcell Generator*, *Pathfinding Node Generator* a další. Mapa se začne vytvářet až po zavolání příslušné metody v skriptu *Game Manager*.

6. Hráč

Hráčova postava a s ní související objekty jsou ovládány třemi skripty. První skript (*Input Manager*) sbírá vstupy z tlačítek ve hře a také z dotyku obrazovky. Zpracování kliknutí tlačítka je jednoduché, stačí pouze zavolat metodu v jiném skriptu, která je přiřazena k tlačítku v editoru. Dotek obrazovky může znamenat jednu ze dvou věcí: buď jde o otáčení kamery, nebo o zaměření kamery na protivníka. Druhý případ se rozezná od prvního tak, že dotek trvá kratší dobu a je v podstatě stále na stejném místě. *Input Manager* poté předává vstup ovladači hráče a kamery (*Player Controller* a *Camera Controller*).

Camera Controller udržuje svoji pozici relativně ke hráči. Přejíždění prstem přes obrazovku otáčí kamerou. Kamera se přiblíží k hráči, když se vyskytne překážka mezi hráčem a kamerou. Po kliknutí na protivníka kamera udržuje uprostřed zorného pole nejen hráče, ale i protivníka, dokud není s kamerou pohnuto.

Player Controller ovládá hráčovu postavu. Nejdůležitější funkcí je reagování na hráčův vstup. Postava dokáže měnit pozici a rotaci a vykonávat akce podle hráčova vstupu. V pravidelných intervalech se hráč pohne a otočí podle pozice joysticku a řeší kolize. Dále hráč dostává od skriptu *Input Manager* informace o stisknutí tlačítek, a pokud je to možné, tak příslušnou akci vykoná. Proveditelnými akcemi je zaútočení, provedení úhybného manévru a vyskočení. V neposlední řadě obsahuje *Player Controller* také odkaz na hráčovy atributy, jako jsou jeho životy a poškození.

Obrázek 1 - Postava ovládaná hráčem

7. Inventář

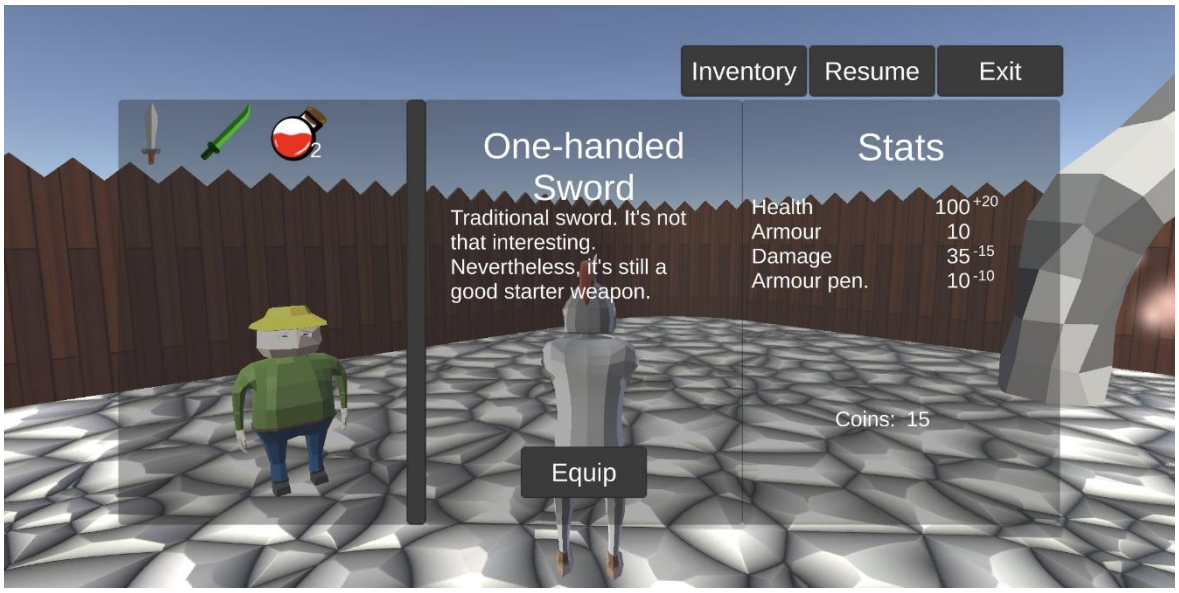
Skript, který ovládá inventář, je *Inventory Mono Behaviour*. Grafické rozhraní se ve hře zobrazí po interakci s obchodem, nebo otevřením menu. V inventáři jsou vidět atributy hráčovy postavy a předměty v inventáři, které lze vybavit, nebo použít.

Předměty jsou zobrazeny jako ikony v mřížce. Po kliknutí na ikonu předmětu se o předmětu zobrazí informace v prostřední kartě. Když má daný předmět atributy, tak se v pravé kartě zobrazí změna atributů oproti právě vybavenému předmětu stejné kategorie. Animace hráčovy postavy se mění podle typu vybavené zbraně.

Předměty jsou implementovány jako objekty typu *Scriptable Object*, které lze upravovat v *Unity* editoru. Každý předmět se skládá z názvu, popisu předmětu, ikony, typu předmětu a ceny. Dále může mít předmět definované atributy nebo typ animace, ale to závisí na typu předmětu.

Výše uvedený skript *Inventory Mono Behaviour* obsahuje odkaz na třídu *Inventory Slot Container* s hráčovými předměty a případně také odkaz na další *Inventory Slot Container* s předměty v obchodu. *Inventory Slot Container* se skládá z pole tříd *Inventory Slot*, ve kterých jsou odkazy na předměty.

Hráč i obchod sdílí kód pro zobrazování inventáře, protože fungují velmi podobně. Když hráč navštíví obchod, může přepínat mezi inventářem a nabídkou obchodu tlačítky na liště nad inventářem. Struktura předmětů a inventáře byla založena na převzatých zdrojích (Coding With Unity, 2021).



Obrázek 2 - Hráčův inventář

8. Nepřátelé

Nepřátelé, včetně bosse, jsou implementováni jako konečné automaty, ale jsou mezi nimi drobné rozdíly ve způsobu implementace. Společné mají to, že se snaží porazit hráče.

8.1 Běžný nepřítel

Zde je popsán způsob fungování obyčejného nepřítele, se kterým se hráč ve hře potkává neustále. Jeho implementace se skládá z několika skriptů. Hlavními skripty jsou *Enemy Controller*, který obsahuje metody přímo ovlivňující protivníka, a *Enemy FSM*, jenž mění stav, ve kterém se nepřítel nachází a volá metody vykonávající akce, které patří k danému stavu. Zbylé skripty týkající se protivníka jsou implementace jednotlivých stavů.

Každý stav protivníka je potomkem třídy *Enemy State*. Když se stav změní, tak je nejprve zavolána metoda *On Exit* starého stavu, která zakončí probíhající aktivity starého stavu. Poté je stav změněn na nový a je zavolána jeho metoda *On Entered*, která nastaví proměnné na počáteční hodnoty a např. ještě najde cestu k hráči. Jednotlivé stavy ještě obsahují metody *Frame Update* a *Physics Update*. *Frame Update* je volán při vykreslení každého snímku a otáčí ukazatelem životů protivníka směrem ke kameře. Metoda *Physics Update* je volána v pravidelných intervalech daných herním engine a u ní se již implementace jednotlivých stavů liší.

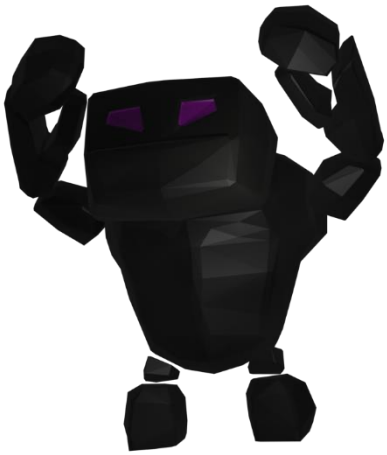
Krátký popis všech stavů protivníka:

- **Attack Target**
  Do toho stavu nepřítel vstoupí, když se nepřítel dostatečně přiblíží k hráči. Spustí se animace útoku a po chvíli udělí hráči poškození, pokud se hráč vyskytuje v dosahu útoku.
- **Follow Path To Target**
  Protivník začne následovat hráče, když se hráč ocitne v jeho zorném poli. Při vstupu do stavu protivník získá cestu k hráči zavoláním metody k nalezení cesty ve skriptu *Pathfinding*. Ze stavu vystoupí, když projde celou cestou nebo se přiblíží k hráči na dosah zbraně.
- **Follow Target**
  V tomto stavu hráč jde přímo za protivníkem. Tento stav se používá k následování hráče, když je hráč poměrně blízko k nepříteli a hráč není skryt za překážkou.
- **Walk To Random Place**
  Aby protivník nestál na místě a pouze nevyhlížel hráče, existuje tento stav. V něm se náhodně vytvoří cesta, kterou nepřítel poté následuje.
- **Face Random Direction**
  Ze stejného důvodu jako u předchozího stavu se v tomto stavu nepřítel pootočí.
- **Wait For Next Action**
  Aby se nepřátelé neustále neotáčeli nebo nepochodovali, tak v tomto stavu stojí na místě a rozhlíží se po protivníkovi.

Obrázek 3 – Nepřítel

8.2 Boss

Boss je implementován jako konečný automat stejně jako nepřátelé. Skript ovládající bosse je nazvaný *Boss* a obsahuje kromě metod řešících pohyb, kolize a útočení také akce. Akce odpovídají implementaci stavu v podobě *Coroutine*. Boss v každé chvíli provádí pouze jednu akci. V editoru lze vytvářet z jednotlivých akcí bosse kombinace útoků. Kombinace útoků je tvořena instancí *ComboSO* typu *Scriptable Object*. V kombinaci útoků se nastaví pořadí akcí, časové prodlevy mezi nimi a preferované podmínky pro provedení akce.



Obrázek 4 – Boss

9. Kolize

Řešení kolizí probíhá v pravidelných intervalech daných *Unity*. Tento interval také odpovídá frekvenci volání metody *Fixed Update*. Kolize je nutné řešit, když dva objekty do sebe narazí, při pohybu postav, a dokonce i když postavy stojí na místě. Řešení kolizí se skládá ze dvou složek, které by bylo možné nazvat aktivním a pasivním řešením kolizí. Metody, které se o to starají, jsou ve statické třídě *Game Physics*.

Jako pasivní řešení kolizí lze označit specifické komponenty, které jsou součástí Unity. Tyto komponenty jsou označovány jako *Colliders* a nabývají například tvar koule nebo hranolu. To, že je na objektu umístěn komponent *Collider*, znamená, že je detekovatelný jinými objekty.

Aktivní řešení kolizí se skládá z vysílaní paprsků (*Raycast*) a počítání průniku dvou *Collider* komponentů. *Raycast* je zjednodušeně řečeno paprsek, který detekuje komponenty typu *Collider* a zapamatuje si informace o objektu, do kterého narazil.

Součástí aktivního řešení kolizí je udržení objektu na zemi. Aby se objekt udržel nad zemí, je potřeba zjistit jeho vzdálenost od země. To se změří prostřednictvím objektu *Raycast*, který se určí počátečním bodem a směrovým vektorem. S jeho pomocí se změří vzdálenost od země a upraví výška objektů nad zemí.

10. Vytváření mapy

Mapa se generuje náhodně v několika krocích. Vytváření mapy je kontrolováno parametry ze *Scriptable Object Maze Settings*. V něm lze nastavit velikost mapy, počet nepřátel na ní, různé pravděpodobnosti generace a také úkol, který je potřeba splnit k odemčení další mapy. Každá úroveň má vlastní *Maze Settings*, které lze upravit v editoru.

Nejprve se z *Maze Settings* převede *Enum* s vítěznou podmínkou na objekt implementující interface *I Win Condition*, která může ovlivnit generaci pomocí svých speciálních pravidel. Poté *Cell Generator* vytvoří pole s buňkami různých šířek a délek. Šířka buňky je dána náhodným číslem v rozmezí, které je specifikováno v *Maze Settings*, a je uložena v poli. Totéž platí pro délku. Buňky jsou vytvářeny tak, že se do zásobníku vloží pozice sousedící s první buňkou, která je ze všech stran otevřená (budou okolo ní vygenerovány sousedící buňky), a dokud není zásobník prázdný, tak se opakuje následující cyklus: vytvoří se nová buňka na pozici vyjmuté ze zásobníku a zkopíruje zdi okolních buněk. U stran, kde ještě neexistuje buňka, se podle pravděpodobností uvedených v *Maze Settings* rozhodne, jestli bude z té strany buňka otevřená. A nakonec se do zásobníku vloží sousední pozice s ještě nevygenerovanými buňkami. *Cell Generator* vrací třídu *Cell Data*, která obsahuje vše, co bylo v něm vygenerováno.

V dalším kroku pracuje *Subcell Generator* s *Cell Data* a podle nich opět *Maze Settings* rozdělí buňky na menší celky, které nazvu pod-buňky. U každé buňky, kromě první, se rozhoduje podle pravděpodobnosti v *Maze Settings*, jestli budou pod-buňky tvořit místnosti nebo chodby. Chodby, na rozdíl od místností, se skládají pouze z pod-buněk, které propojují sousední buňky. I zde se uplatní zásobník, do kterého se vloží první buňka a

dokud není zásobník prázdný, tak se vyjme první buňka ze zásobníku, ta se rozdělí na pod-buňky a vloží do zásobníku své sousedy. *Maze Generator* obdrží zpět třídu *Subcell Data*, která obsahuje informace o vytvořených pod-buňkách. Poté se případně ještě vytvoří instance místnosti s bossem, pokud je to v pravidlech generace u aktuální *I Win Condition*.

Na řadu přijde *Tile Generator*, který k práci potřebuje *Subcell Data* z předchozího kroku. *Tile Generator* vytvoří na místě každé pod-buňky instanci části místnosti podle propojených okolních pod-buněk. Části místnosti můžou být náhodně zvoleny, když je na výběr z více setů.

Předposledním krokem je vytvoření mřížky k hledání cesty pro nepřátele. V *Maze Settings* je uvedeno kolik vrcholů pro hledání cesty by mělo být v jedné pod-buňce. Ve hře je pod-buňka rozdělena na 25 vrcholů ve čtvercové síti. Výsledné vrcholy jsou předány *Pathfinder* skriptu, který slouží k vyhledávání cest.

Nakonec je mapa zaplněna protivníky. Minimální a maximální počet protivníků je nastavitelný v *Maze Settings*, ale je také ovlivněn pravděpodobností, se kterou se pokouší vytvořit instanci nepřítele na každém vrcholu k hledání cesty. *I Win Condition* také může ovlivnit vytvoření instance předmětu místo protivníka.

11. Hledání cesty

K vyhledávání cesty je použit *A* * algoritmus, který se podobá *Dijkstrovu* algoritmu. Oproti němu obsahuje heuristický prvek. Popis algoritmu:

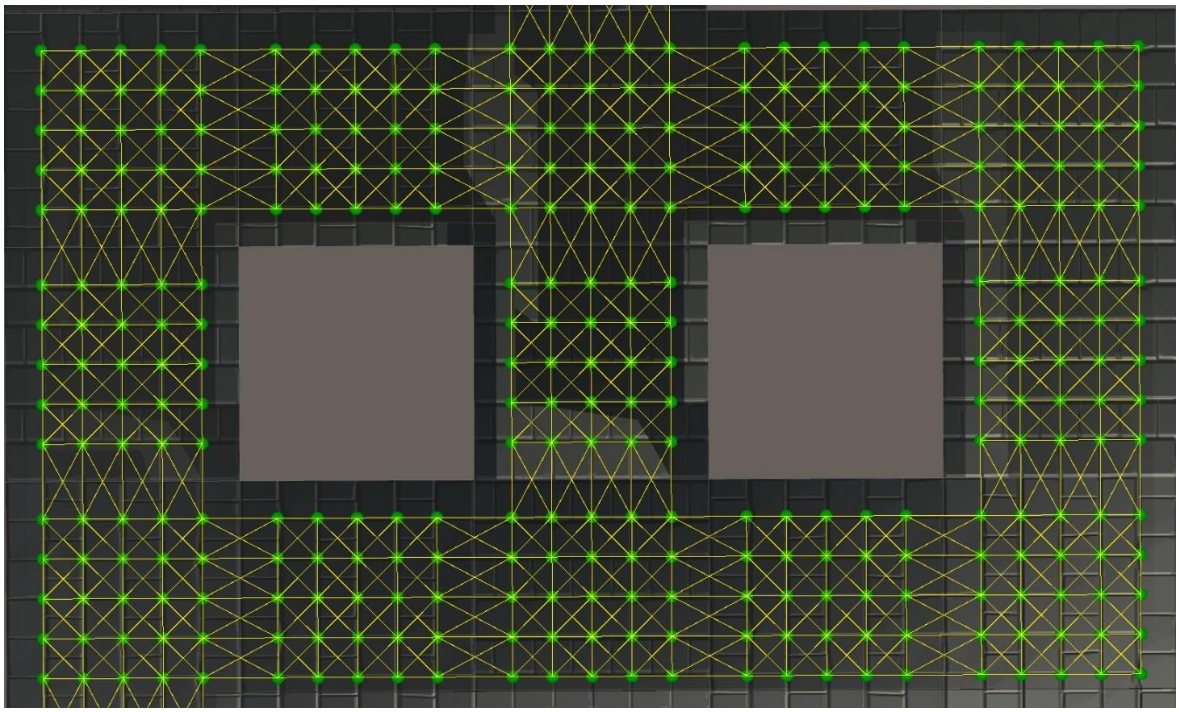
Na počátku jsou dvě množiny. První lze nazvat otevřenou množinou, protože obsahuje zpracovávané prvky. Ta obsahuje zprvu počáteční uzel. Druhá množina se nazývá uzavřená a jsou do ní vkládány prvky, které nabyly konečnou hodnotu. Dokud není otevřená množina prázdná, tak se opakuje následující posloupnost příkazů:

1. Z otevřené množiny se vyjme prvek *X* s nejmenší hodnotou *F*
2. Když je *X* cílovým prvkem, tak je cesta nalezena
3. *X* se z otevřené množiny přesune do množiny uzavřené
4. Pro všechny sousedy prvku *X* proběhne následující cyklus:
    1. Když je soused v uzavřené množině, tak se přeskočí
    2. Když není v otevřené množině, tak je do ní přidán a spočítá se jeho *F* hodnota
    3. Když je v otevřené množině, tak je případně jeho hodnota *F* nahrazena novou nižší hodnotou

Kde hodnota *F* je součet vzdálenosti od počátku (*G*) a odhadované vzdálenosti od konce (*H*).

Implementace haldy ve hře je dále optimalizovaná tím, že je otevřená množina tvořena haldou, takže vyhledání vrcholu s nejmenší hodnotou je velmi rychlé.

Na obrázku na následující stránce je znázorněna síť vrcholů pro vyhledávání cesty. Každý úsek chodby obsahuje čtvercovou síť vrcholů, které jsou mezi sebou propojeny.



Obrázek 5 - Vrcholy k vyhledávání cesty a jejich propojení

12. Modely

Modely ve hře jsem sám vytvořil. Byly vytvořeny v programu *Blender* (Blender, 2021). *Blender* je bezplatný open-source 3D nástroj určený pro vytváření kreslených filmů, vizuálních efektů a 3D modelů. Styl modelů, který jsem si zvolil se nazývá *low poly*. Důvodem byl nízký počet trojúhelníků, ze kterých se modely skládají, a jedinečný vzhled.

Modely byly vytvářeny v několika krocích. Nejprve byl vytvořen model skládající se z vrcholů, hran a stěn. Poté byla vytvořena pro model specifická textura, nebo byly stěnám modelu přiřazeny barvy z barevné palety, kterou jsem našel na internetu (Imphenzia, 2021). Když se u modelu jednalo o postavu, která se ve hře pohybuje, bylo potřeba pro ni vytvořit také kostru a animace. Modely byly nakonec exportovány do *Unity* ve formátu *FBX*.

13. Prostředí

Hra je napsána v jazyce *C#*, protože je to jediný jazyk, který podporuje herní engine *Unity* (Unity, 2021). Multiplatformní herní engine *Unity* byl vytvořen *Unity Technologies*. Výrazná část vytváření hry se odehrává v *Unity* editoru, který má podobu aplikace s grafickým rozhraním. *IDE* mého výběru je *Microsoft Visual Studio 2017* (Microsoft Corporation, 2021).

14. Testování

Testování hry by se dalo rozdělit na dvě fáze. V první fázi byly odhalovány chyby v programu. Mezi ně například patří detekce objektů na špatné vrstvě nebo nekonečný cyklus při změně stavu protivníka. Kvůli nespokojenosti s první implementací konečného automatu protivníka jsem tento systém přepracoval. Druhá část testování obsahovala úpravu hodnot hráče a protivníků, aby byla hra lépe hratelná. Jako příklad lze uvést upravení dosahu nepřátel nebo atributů zbraní.

Hratelnost hry byla testována v *Unity* editoru a na mobilním zařízení *Samsung Galaxy S10* s verzí *Android 11*.

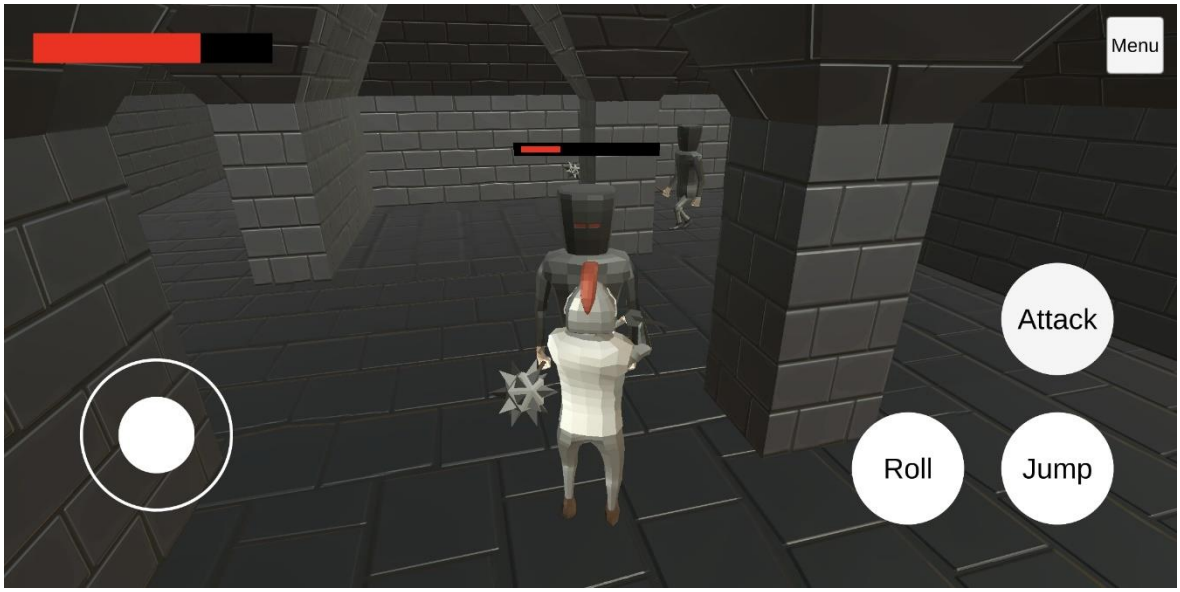
15. Instalace

Pro instalaci hry stáhněte soubor *HraZPohleduTretiOsoby.apk* z *GitHub* repozitáře ze složky *Build*. Poté připojte mobilní zařízení s operačním systémem *Android*. Ujistěte se, že je povolen přenos souborů na mobilním zařízení. Poté klikněte pravým tlačítkem myši na stažený soubor, najeďte na možnost „odeslat“ a vyberte připojené zařízení. Na mobilu povolte instalaci aplikací z neznámých zdrojů, najděte aplikaci v paměti na mobilním zařízení a nainstalujte ji.

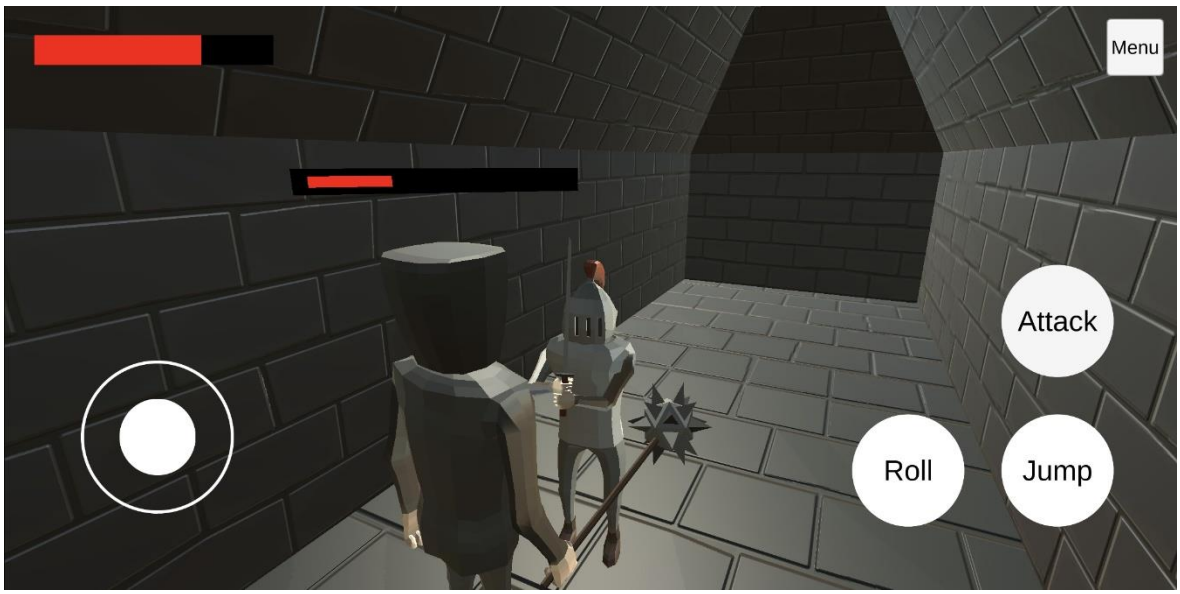
Pro sestavení aplikace v editoru, místo užití již sestavené hry, je potřeba mít nainstalovaný herní engine *Unity*. Hra byla vyvíjena ve verzi *Unity 2018.4.32f1*. Kromě *Unity* je potřeba mít stažený *Android SDK* a *Android NDK* ve verzi *r16b*. Cestu k *Android SDK* a *Android NDK* lze nastavit v záložce „External tools“ dialogového okna, které se objeví po rozkliknutí „Edit“ na horní liště editoru a zvolení možnosti „Preferences“. Aby bylo možné projekt otevřít, je nutné mít uživatelský učet v *Unity*. Po otevření projektu v editoru rozklikněte možnost „File“ na horní liště a vyberte možnost „Build Settings“. Otevře se dialogové okno. Zvolte platformu *Android* a klikněte na tlačítko „Switch Platform“. Poté, co se skončí převádění projektu na požadovanou platformu, klikněte na tlačítko „Build“.

16. Snímky obrazovky

Snímky obrazovky z mobilního telefonu s operačním systémem Android pořízené během hraní hry.



Obrázek 6 - Snímek obrazovky 1



Obrázek 7 - Snímek obrazovky 2

17. Závěr

Podařilo se mi vytvořit funkční aplikaci, která běží na platformě *Android*. Hráč může ovládat svoji postavu prostřednictvím dotykové obrazovky. Hráč má také inventář, ve kterém si může vybavovat předměty, které si lze zakoupit v obchodě.

Nepřátelé jsou implementováni jako konečné automaty. Rozhodují se pro různé akce a ty také vykonávají. K hledání cesty používají optimalizovaný *A* * algoritmus. Hráč se také utká s bossem, kterého potřebuje porazit k dokončení hry.

Projekt je nyní dokončený podle zadání, práce na projektu mě ale natolik zaujala, že vidím mnoho dalších příležitostí na jeho rozšíření. Takové rozšíření by mohlo zahrnovat více druhů protivníků, jiné vzhledy mapy a dalšího bosse.

Po všech stánkách projektu, včetně programovací, jsem s výsledkem spokojen. Kromě naprogramování kódu, jsem vytvořil i modely pro hráče, protivníky atd.

18. Seznam obrázků:

19. Zdroje

Blender. (22. Leden 2021). Načteno z Blender: https://www.blender.org/

Coding With Unity. (22. Leden 2021). *Unity3D - Item Buffs/Stats | Scriptable Object Inventory System | Part 3*. Načteno z YouTube: https://www.youtube.com/watch?v=LcizwQ7ogGA

Coding With Unity. (22. Leden 2021). *Unity3D - Saving and Loading your Inventory with Scriptable Objects | Part 2*. Načteno z YouTube: https://www.youtube.com/watch?v=232EqU1k9yQ

Coding With Unity. (22. Leden 2021). *Unity3D - Scriptable Object Inventory System | Part 1*. Načteno z YouTube: https://www.youtube.com/watch?v=_IqTeruf3-s

Fenerax Studios. (22. Leden 2021). *Joystick Pack*. Načteno z Unity Asset Store: https://assetstore.unity.com/packages/tools/input-management/joystick-pack-107631

glitchers. (22. Leden 2021). *Answers: Unity*. Načteno z Unity: https://answers.unity.com/questions/486545/getting-all-assets-of-the-specified-type.html

Imphenzia. (22. Leden 2021). Načteno z Dropbox: https://www.dropbox.com/s/c5olic38j8fopet/ImphenziaPalette01.png

Microsoft Corporation. (22. Březen 2021). *Visual Studio*. Načteno z Visual Studio: https://visualstudio.microsoft.com/cs/

Unity. (21. Leden 2021). Načteno z Unity: https://unity.com/