FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Projekt

Vývoj akční videohry zasazené do prostředí FM

Studijní program: B0613A140005 – Informační technologie
Autor práce: **Radek Mocek**
Vedoucí práce: Ing. Jan Hybš

Liberec 2023

Tento list nahraďte
originálem zadání.

Prohlášení

Prohlašuji, že svůj projekt jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mého projektu a konzultantem.

Jsem si vědom toho, že na můj projekt se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mého projektu pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li projekt nebo poskytnu-li licenci k jeho využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Beru na vědomí, že můj projekt bude zveřejněn Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

20. 5. 2023 Radek Mocek

Vývoj akční videohry zasazené do prostředí FM

Abstrakt

Český abstrakt

Klíčová slova: klíčová slova česky

Development of an action video game set in FM

Abstract

English abstract

Keywords: keywords in English

Obsah

Seznam obrázků

Seznam zkratek

TUL Technická univerzita v Liberci

FM Fakulta mechatroniky, informatiky a mezioborových studií Technické univerzity v Liberci

1 Úvod

2 Herní engine Unity

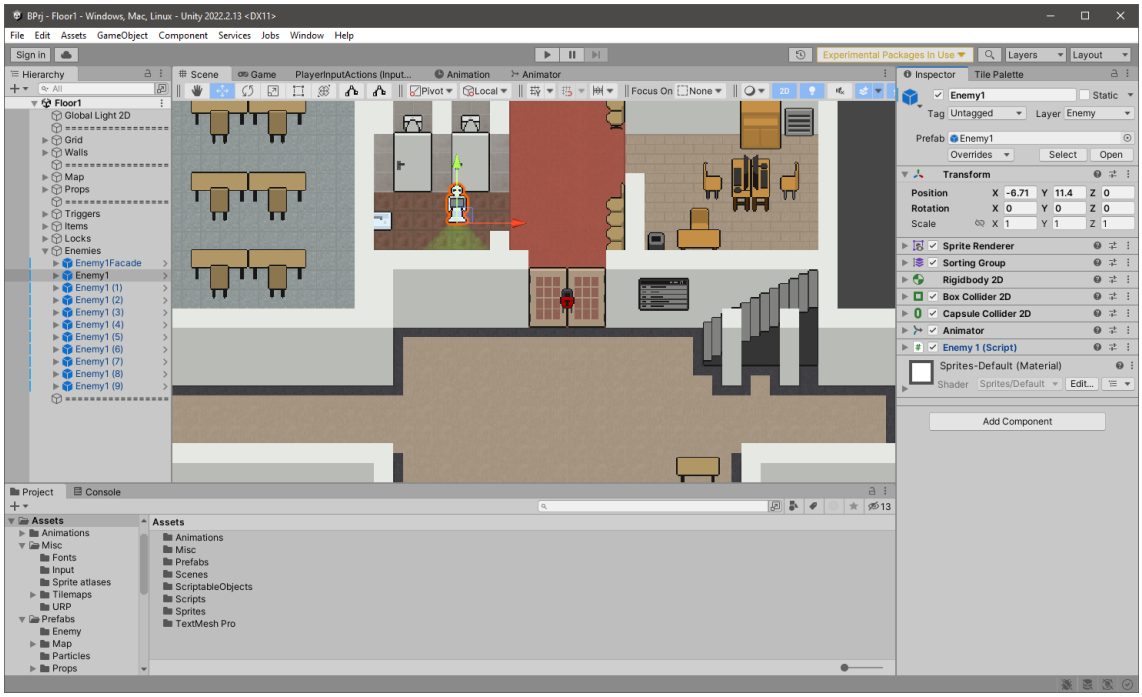
Unity je multiplatformní herní engine s uzavřeným kódem vyvíjený společností Unity Technologies. První veřejná verze vyšla v roce 2005 a od té doby si Unity vybudovalo početnou komunitu a stalo se populární především mezi nezávislými vývojáři. Unity poskytuje nástroje pro vývoj 2D a 3D her, využívá se ale také ve filmovém nebo automobilovém průmyslu. Základní verze Unity je zdarma.

2.1 Editor a základní stavební prvky

Základem každého Unity projektu jsou tzv. assety. Asset je jakýkoliv prvek, který je v projektu využit k vytvoření finální hry nebo aplikace. Asset je reprezentován ve formě souboru a často se jedná o prvek tvořící audiovizuální stránku hry (např. 3D model, textura, sprite, zvukový soubor). Dalšími důležitými assety jsou scény, které slouží jako herní prostředí a uchovávají v sobě ostatní assety v podobě herních objektů, a skripty psané v jazyce C#, které definují chování daných herních objektů. [1]

Každý prvek použitý ve scéně je herním objektem. Herní objekt nemá sám o sobě žádnou funkcionalitu, jedná se pouze o kontejner pro komponenty, které pak určují jeho vzhled a chování [2]. Základní komponenta umístěná na téměř každém herním objektu je *Transform*, která určuje jeho umístění, rotaci a měřítko v prostoru scény. Do některých komponent lze jako argument dosadit asset, např. dosazení spritu do komponenty *Sprite Renderer*, která zajistí jeho vykreslení. Pro opakované použití stejného herního objektu je vhodné z něj vytvořit tzv. prefab. Prefab je asset reprezentující herní objekt a změna hodnot v jeho komponentách se projeví ve všech jeho instancích.

Unity editor lze ve svém výchozím rozložení rozdělit na čtyři základní části. Dole se nachází průzkumník projektu, ten umožňuje procházení, třídění a importování assetů do projektu. Uprostřed je editor scény, ve kterém si lze scénu prohlížet a vybírat v ní obsažené herní objekty pro následné úpravy. Nalevo od editoru scény je průzkumník její hierarchie, v něm jsou vidět všechny herní objekty (včetně instancí prefabů), které jsou obsaženy v právě prohlížené scéně. V neposlední řadě se na pravé straně nachází inspektor, který zobrazuje detailní informace o právě vybraném herním objektu nebo assetu a umožňuje jeho modifikaci. Pro herní objekt zobrazuje všechny jeho komponenty, zatímco u assetů se zobrazení informací liší podle typu zvoleného assetu.



Obrázek 2.1: Unity editor

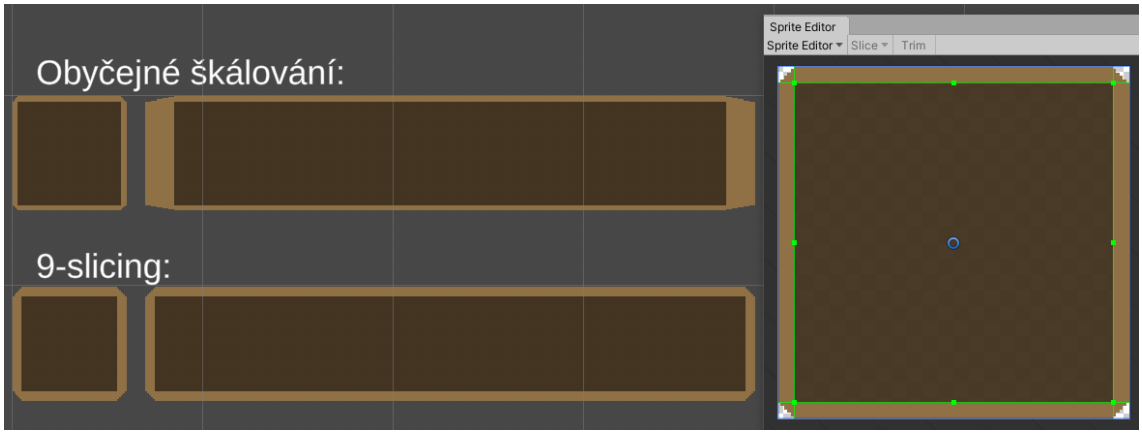
2.2 2D tvorba

Unity vzniklo původně jako engine pro vývoj 3D her a nástroje pro 2D tvorbu byly přidány až v pozdějších verzích. Volba mezi 2D a 3D je prvním krokem při zakládání nového projektu, rozdíl mezi nimi ale není velký a oba přístupy lze během vývoje kombinovat. Hlavním viditelným rozdílem pro 2D projekty je editor scény, který je přepnut do dvourozměrného režimu. V tomto módu je změněn význam osy Z, která místo popisu třetího rozměru značí hloubku (podobně jako např. vlastnost z-index v CSS). [3]

2.2.1 Sprite

Sprite je jeden z nejběžnějších assetů ve 2D videohře. V Unity vzniká přetažením bitmapového obrázku do průzkumníka projektu (obvykle do složky *Sprites*) a má několik vlastností, které lze změnit v inspektoru. Za zmínku stojí hodnota *Pixels Per Unit*, která určuje, kolik pixelů ve spritu odpovídá jednotce vzdálenosti v herním světě (ve scéně). Druhý důležitý parametr je *Pivot*, ten značí bod, kolem kterého sprite rotuje, a také může ovlivnit pořadí vykreslení daného spritu.

Každý sprite nemusí být reprezentován samostatným bitmapovým souborem, místo toho lze naimportovat tzv. spritesheet. Jedná se o větší obrázek obsahující více spritů uspořádaných do mřížky, ty jsou pak rozděleny na jednotlivé assety v nástroji editor spritů. Tento nástroj také umožňuje připravit obrázek pro specifický typ škálování zvaný 9-slicing (obrázek 2.2).



Obrázek 2.2: 9-slicing

2.2.2 Animace

Unity umožňuje tvorbu animací po jednotlivých snímcích, kdy je každý snímek animace reprezentován samostatným spritem. Jednotlivé sprity jsou pokládány na časovou osu v nástroji *Animation*. Nevýhoda této metody se projeví v situaci, kdy je potřeba např. nějaké postavě změnit barvu vlasů. V tomto případě je nutné všechny sprity všech animací postavy upravit v (externím) grafickém editoru. Problém by šel také vyřešit tvorbou speciálního shaderu nebo editací jednotlivých pixelů ze skriptu, nejedná se ale o standardní postupy. Alternativně lze použít metodu trojúhelníkové sítě a kostry. Jedná se o běžný způsob tvorby 3D animací, Unity jej ale umožňuje aplikovat i na 2D sprity.

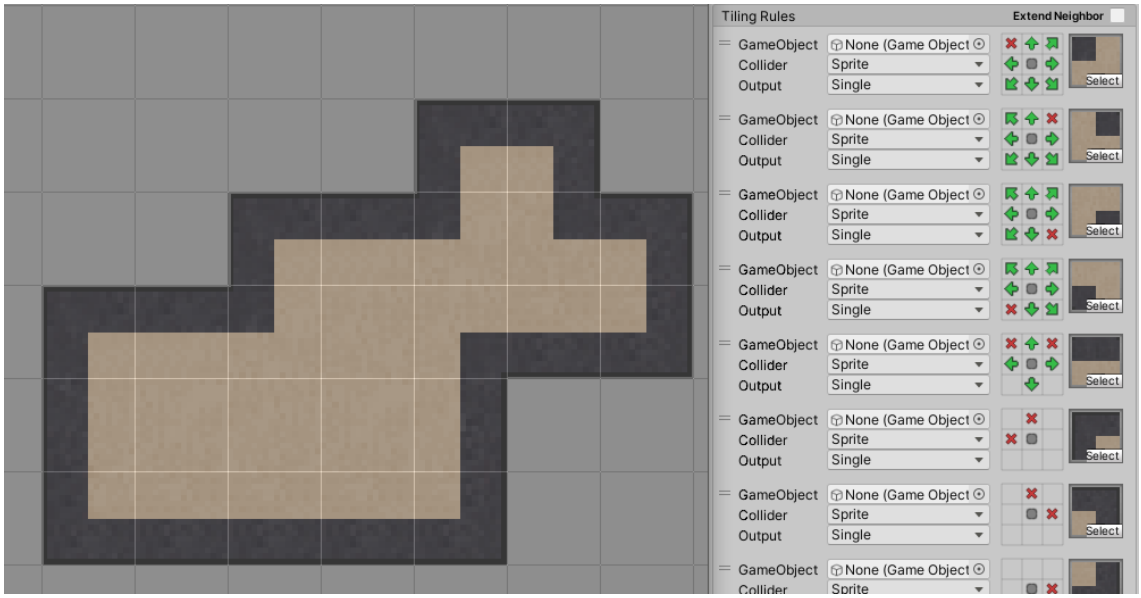
Každý herní objekt schopný animace musí mít přiřazený asset typu *Animator Controller* (dále jen AC). Ten je vytvořen a upravován přímo v Unity editoru a jeho diagram připomíná schéma stavového automatu. Všechny animace, ve kterých se herní objekt může nacházet, jsou v diagramu reprezentovány obdélníkem. Obdélníky lze spojovat šipkami, které značí podmíněný přechod mezi jednotlivými animacemi. AC umožňuje definovat si v něm interní proměnné (*parameters*), jejichž hodnoty lze měnit ze skriptu a využívají se pro sestavení podmínek přechodu. AC diagramy složitějších herních objektů mohou značně nabýt na složitosti, je možné se jim ale vyhnout. Volání funkce *Animator.CrossFade* dovoluje změnit animaci kdykoliv bez ohledu na stav AC diagramu.

2.2.3 Tilemap

Tilemap je nástroj pro tvorbu herních světů převážně ve 2D. Základem je pravidelná mřížka, do které jsou vkládány dlaždice (*tiles*). Dlaždice je malý sprite nakreslený tak, aby více dlaždic položených vedle sebe vypadalo přirozeně a dala se z nich takto poskládat celá herní mapa. Využitím tilemap lze tvořit velké herní mapy bez dopadu na výkon a zároveň je kdykoliv snadno editovat. Unity nabízí obdélníkovou tilemap (nejběžnější, vhodné pro plošinovky a top-down videohry), ale také šestiúhelníkovou nebo izometrickou. V jedné scéně se může nacházet více tilemap a každá může být

v jiné vrstvě. To např. umožňuje, aby se některé dlaždice vykreslovaly před a některé za postavou hráče.

Rule Tile je speciální druh dlaždice, která mění svůj vzhled na základě jednoduchých nastavitelných pravidel. Sprite této dlaždice se mění podle toho, zdali se v jejím okolí nachází dlaždice stejného druhu. Funkcionalita je ale omezená, nelze např. nastavit, aby dlaždice reagovala na své sousedy z jiných vrstev nebo na jiné dlaždice ve stejné vrstvě.



Obrázek 2.3: Rule Tile

2.2.4 Universal Render Pipeline

Universal Render Pipeline (URP) je skriptovatelný vykreslovací řetězec, který je do Unity projektu přidán ve formě balíku. Otázka použití alternativního vykreslovacího řetězce bývá častější u vývoje 3D her, i pro 2D projekty ale URP přináší nové grafické možnosti. Tím nejzásadnějším jsou dvě nové komponenty *Light 2D* a *Shadow Caster 2D*, díky kterým lze do scény přidat dynamická světla a stíny.

3 Návrh videohry

3.1 Příběh

Navrhovaná hra tohoto projektu uvádí hráče do role maturanta, který se chystá na FM podat přihlášku ke studiu. Po vstupu na fakultu se za ním zavřou a uzamknou dveře. Novým cílem je tedy zjistit, co se děje. Posléze je hráč kontaktován osobou, která ho informuje, že celou budovu fakulty má pod kontrolou místní umělá inteligence, jež se vymkla kontrole.

Hlavním cílem hry je zastavit umělou inteligenci a uniknout z fakulty. V budově se nachází nepřátelští roboti, kteří se v tom hráči pokusí zabránit. Zhruba v druhé polovině příběhu se hráč dozvídá, že osoba, jež ho kontaktovala, byla ve skutečnosti samotná umělá inteligence, která ho tímto využívala. Koncept příběhu je inspirován herní sérií System Shock.

3.2 Žánr

Obecně by se hra dala kategorizovat jako akční a nabízí dva základní styly hraní. Tím prvním je boj na blízko. Hráč má k dispozici zbraň, konkrétně se jedná o hasák, se kterou může útočit na nepřátele. Ti mohou stejným způsobem zase útočit na hráče. Po každém úspěšném zásahu je cíli útoku ubráno jeho zdraví, které je v případě této hry reprezentováno celým číslem, a jakmile se dostane na nulu, tak je cíl zneškodněn.

Druhou možností pro hráče je vyhnout se přímé konfrontaci s nepřáteli a namísto toho se před nimi skrývat. Žánr, kde se hráč snaží, aby nebyl odhalen nepřátelskými jednotkami, se ve videoherní terminologii nazývá stealth game. Slovem stealth se označuje i hlavní mechanika tohoto žánru – plížení, které hráči pomáhá nebýt spatřen.

3.3 Vizuální stránka

Hra je zobrazována pohledem kamery, která na mapu pohlíží svrchu. Tato perspektiva bývá obecně nazývána jako top-down. V případě tohoto projektu není kamera namířena přímo kolmo na plochu mapy, ale sleduje herní svět pod úhlem přibližně 45 stupňů. Pohled pod určitým úhlem odhaluje hloubku vyobrazovaných objektů a přesněji by se dal nazvat jako axonometrická projekce. [4]

Grafika hry je nakreslena ve stylu pixel art. Jedná se o druh digitálního grafického umění, kde každý pixel ve výsledném obraze hraje svou roli a změna jediného pixelu může značně ovlivnit, jak obraz na pozorovatele působí. Počátky pixel artu sahají do dob 8 a 16bitových konzolí, kde byla omezená paleta barev a rozlišení zobrazovaných spritů. Svou důležitost si ale pixel art zachoval dodnes. Je oblíbený zejména mezi nezávislými vývojáři kvůli jednoduchosti a rychlosti tvorby. [5]

V grafickém stylu této hry není dodrženo pravidlo užití omezené palety barev. Sprite postavy hráče má rozměry 16 na 32 pixelů a od toho jsou přibližně odvozeny velikosti všech ostatních spritů. Kromě dlaždic pro tilemap mají všechny sprity přidán tmavý obrys o šířce jednoho pixelu, aby vynikly a nesplývaly s pozadím.

3.4 Herní mechaniky

3.4.1 Pohyb

Postava hráče se může pohybovat v osmi směrech pomocí kláves W, A, S a D. Stiskem mezerníku se provede rychlý úskok (dash), během kterého je postava nezranitelná. Dash ubírá hráčovu staminu, která se časem regeneruje na svou původní hodnotu, a nelze ho provést, pokud není staminy dostatek.

Aby hráč nenavštěvoval jednotlivé lokace na herní mapě v pořadí, které by porušilo konzistenci příběhu, je jeho pohyb omezen pomocí uzamčených dveří. Aby tyto dveře mohly být otevřeny, musí nejprve hráč nalézt klíč v barvě odpovídající zámku na daných dveřích.

Stiskem klávesy Ctrl se postava hráče skrčí a přepne do režimu plížení. V tomto módu se pohybuje pomaleji, ale může pro přesun využít větracích šachet rozmístěných po mapě. Hráč ve větrací šachtě nemůže být spatřen nepřítelem. To samé platí, pokud je v režimu plížení schovaný za nějakým objektem (např. za stolem).

3.4.2 Boj

Stiskem levého tlačítka myši (LMB) se provede útok na blízko, ten si bere malou část staminy a na chvíli pozastavuje její regeneraci. Útok má také svou vlastní dobíjecí dobu, aby nemohl být opakován jeden za druhým bez jakékoliv pauzy (attack spamming). Po vypršení dobíjecí doby následuje krátké časové okénko, během kterého stisknutí LMB provede útok s kritickým zásahem, který ubírá více bodů zdraví. Pokud naopak hráč stiskne LMB během dobíjecí doby, již není možné provést útok s kritickým zásahem a dobíjecí doba se o toto časové okénko prodlouží.

Hráč má svou hladinu zdraví, která se, pokud nebyl po určitou dobu zraněn nepřítelem, regeneruje do poloviny své maximální kapacity. Pokud hráčovo zdraví klesne na nulu, neznamená to konec hry. Namísto toho je hráč oživen (respawn) ve vestibulu budovy s tím, že jeho aktuální úkol, sebrané předměty a zneškodnění nepřátelé nejsou resetováni.

3.4.3 Grafické uživatelské rozhraní

Pro informování hráče ohledně hodnot určitých atributů se používá heads-up display (HUD). Jedná se o samostatnou vrstvu, která je vykreslena nad herním světem a její pozice na obrazovce je fixní. V pravé dolní části obrazovky je napsána informace o právě zkoumaném objektu ve hře. Jedná se o objekt, na který uživatel najede kurzorem. Na prvním řádku je zobrazen název tohoto objektu a pokud s ním lze nějakým způsobem interagovat, tak je na druhém řádku napsán název příslušné akce (např. otevření dveří). Při zkoumání nepřátelské jednotky je na druhém řádku vykreslena hladina jejího zdraví. Ukazatele hráčova zdraví a staminy se nachází v levé dolní části obrazovky.

Pokud je pro hráče k dispozici nový tutoriál, tak se nad těmito dvěma ukazateli zobrazí oznámení. Stiskem klávesy tabulátoru se otevře okno s tutoriálem. Jedná se o stručnou nápovědu popisující, jak se má hra ovládat a jak fungují základní mechaniky. Na začátku hry je k dispozici pouze první tutoriál popisující pohyb postavy a další nápověda se zpřístupní ve chvíli, kdy je relevantní.

3.5 Umělá inteligence

Hra obsahuje nepřátelskou jednotku s jednoduchou umělou inteligencí řízenou konečným stavovým automatem. Výchozí činností nepřítele je hlídkovat libovolný počet bodů na mapě. Jakmile nepřítel zahlédne postavu hráče, vydá se toto místo prozkoumat a popřípadě na hráče zaútočí. Nepřítel využívá algoritmu A*, aby věděl, jakou cestou se vydat ke svému aktuálnímu cíli.

Je třeba zmínit, že pojem „umělá inteligence“ v kontextu videoherní tvorby obvykle neznamená algoritmus řešící komplexní problémy na základě datové analýzy. Cílem je vytvořit chování nehráčských postav, které zlepšuje herní zážitek. Předvídatelnost a čitelnost může mít při návrhu takového algoritmu prioritu před simulací reálného lidského chování. [6]

3.5.1 Konečný stavový automat

Konečný stavový automat (finite state machine, FSM) je výpočetní model běžně používaný ve videohrách pro implementaci umělé inteligence nehráčských postav. Objekt implementující FSM má definovaný konečný počet stavů a vždy se musí nacházet právě v jednom z nich. V případě her jednotlivé stavy reprezentují akce a činnosti, které mohou nehráčské postavy vykonávat. K návrhu FSM lze použít graf, kde je pro každý stav vytvořen příslušný vrchol a orientované hrany mezi těmito vrcholy značí podmíněné přechody mezi stavy. Objekt se tedy přepne z jednoho stavu do jiného, pokud mezi těmito stavy existuje přechod a jsou splněny jeho podmínky. [7]

3.5.2 Nepřátelská jednotka – návrh

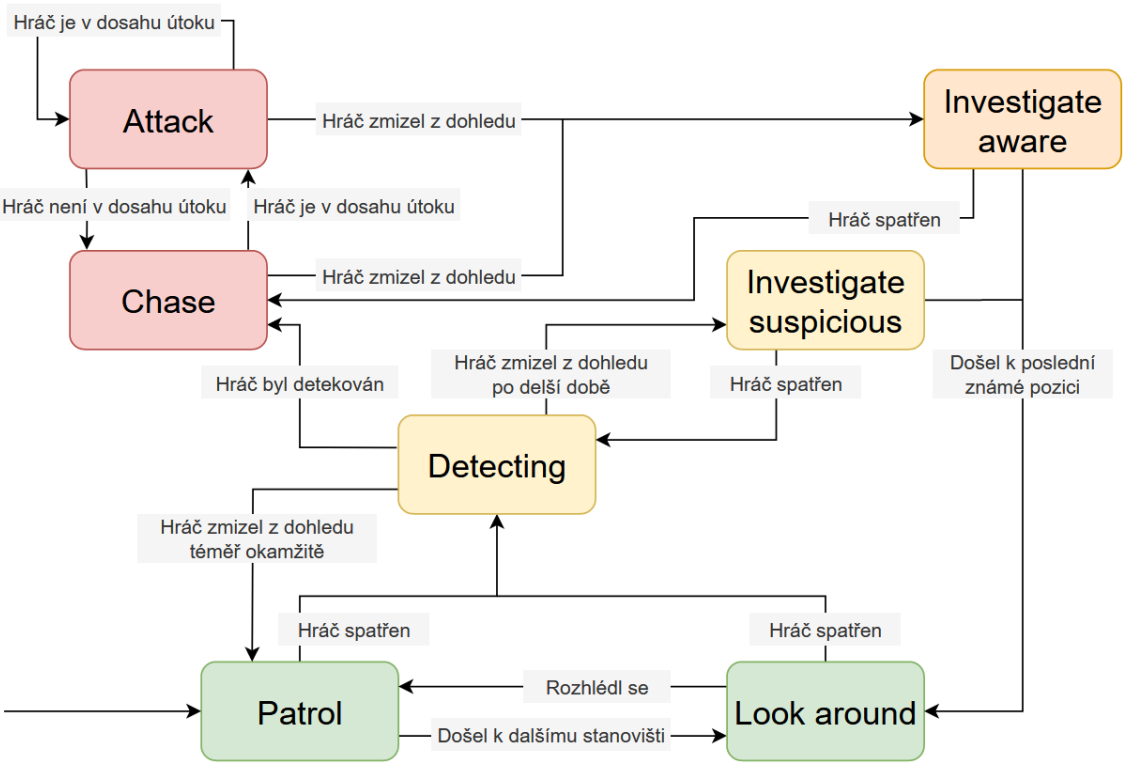
Výchozím stavem nepřátel v této hře je stav hlídky *PatrolState*. Nepřítel hlídkuje mezi jednotlivými body na mapě, které má v sobě uložené jako kolekci dvourozměrných souřadnic. Jakmile se dostane na souřadnice stanoviště, zastaví se a přepne do stavu rozhlížení *LookAroundState*, po kterém následuje opět stav hlídky. Tentokrát má ale tento stav jako stanoviště nastavené následující souřadnice z kolekce. Mezi těmito dvěma stavy takto nepřítel přepíná do té doby, než se do jeho zorného pole dostane postava hráče.

Zorné pole nepřátel je ve hře vizualizováno pomocí barevné kruhové výseče, která je vykreslena na zemi tak, že střed kružnice leží nepříteli u nohou a její poloměr je roven délce dohledu nepřítele. Tvar této výseče se dynamicky mění, aby odpovídal tomu, kde nepřítel může a nemůže spatřit hráče. Pokud se tedy např. hráč krčí za stolem, tak je zkrácena délka paprsků výseče, které se stolem kolidují.

Výchozí barvou výseče během stavu hlídky je zelená. Jakmile do zorného pole nepřítele vstoupí hráč, změní výseč svou barvu na žlutou a nepřítel se přepne do stavu detekce *DetectingState*. Pokud se hráč v určitém čase nestihne skrýt, pak nepřítel přejde do stavu pronásledování *ChaseState*. Tento čas, během kterého má hráč šanci se ukrýt a nebýt detekován, je také vizualizován. Nad žlutou výsečí je vykreslena druhá červená výseč stejného tvaru, jejíž poloměr se postupně zvětšuje a jakmile se její okraj dostane až k hráči, znamená to, že byl detekován. Během stavu hlídky se pak poloměr červené výseče zmenšuje zpět na nulu. Tento druh zobrazení zorného pole a hladiny podezření nepřátelských jednotek je inspirován hrou Shadow Tactics: Blades of the Shogun.

Při stavu pronásledování nepřítel stíhá hráče a když se dostane dostatečně blízko, tak na něj zaútočí (*AttackState*). Pokud hráč zmizí z jeho dohledu, přechází nepřítel do stavu průzkumu *InvestigateAwareState*, kde je cílem jeho cesty poslední pozice, na které byl hráč viděn. Spatření hráče během tohoto stavu znamená opětovné pronásledování, jinak, jakmile se dostane na konec své cesty, se nepřítel vrátí k rutině neutrálních stavů hlídka a rozhlížení. Podobně funguje stav *InvestigateSuspiciousState* s tím rozdílem, že do něj se lze dostat ze stavu detekce a spatření hráče tedy znamená opětovnou detekci.

Posledním stavem je *KnockbackState* (vynechán z obrázku 3.1 kvůli přehlednosti). Do něj se nepřítel dostane po obdržení zásahu od hráče. V tomto stavu je mírně odstrčen ve směru obdrženého útoku a následně se vrací do stavu útoku, pronásledování, nebo průzkumu podle toho, jak daleko se od něj nachází hráč a zdali je vůbec v jeho zorném poli.



Obrázek 3.1: Graf stavového automatu nepřátelské jednotky

4 Implementace

Po návrhu videohry následovala její implementace, která začala založením prázdného 2D projektu v Unity 2021. Byla zvolena verze 2021, protože se jedná o verzi s dlouhodobou podporou a samo Unity ji v tuto chvíli doporučuje.

Poté už se mohly začít tvořit jednotlivé assety. Prvním cílem bylo vytvořit postavu hráče a umožnit jí základní pohyb. To si vyžadovalo nakreslit sprity a z nich sestavit animace, napsat skripty pohybující s postavou hráče, ale také zprovoznit tilemap pro základní testovací mapu.

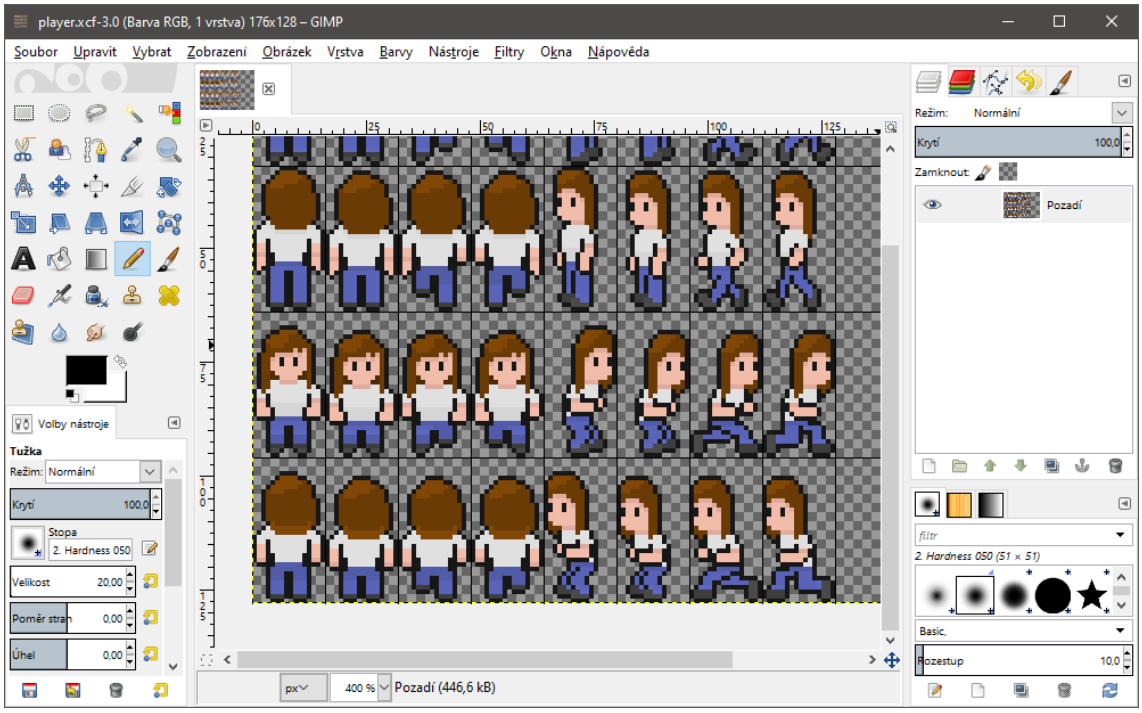
Implementace takto probíhala postupným nabalováním jednotlivých herních mechanik. Vždy se pracovalo na jedné konkrétní funkční části a dokud nebyla dotažena do uspokojivého stavu, nevěnoval se vývoj ničemu jinému. Tento postup vývoje byl validní, dokud se nedostal k tvorbě nepřátelské jednotky, která byla časově náročnější. Hra sice obsahovala nepřátelskou jednotku, která se postupně zlepšovala, ale čas určený pro praktickou část této práce se krátil a hra postrádala jakýkoliv příběh či mapu, na které by se tento příběh odehrával.

Proto byl zvolen přístup tvorby minimálního životaschopného produktu. Prioritou bylo zhotovit hru, kterou bude možné dohrát od začátku do konce, a až poté se věnovat případným detailům, chybám a nedostatkům. Vývoj se po této změně zaměřoval převážně na herní mapu, příběh a grafické uživatelské rozhraní. Hra byla po dokončení své první verze odeslána malé skupince testerů, na základě jejichž zpětné vazby byly doladěny některé herní mechaniky a opraveny nalezené chyby.

4.1 Grafika

Unity v sobě nemá zabudovaný žádný grafický editor, pro tvorbu spritů do 2D hry je ale možné použít jakýkoliv program s podporou průhlednosti. Vhodným nástrojem pro tvorbu herní grafiky ve stylu pixel art je program Aseprite, pro tento projekt byl ale zvolen GIMP kvůli své bezplatné licenci a předchozím zkušenostem s tímto programem.

V programu GIMP byla většina potřeb grafiky této hry pokryta nástrojem tužka. Vhod přišla také možnost nechat si v obrázku vykreslit mřížku oddělující jednotlivé snímky ve spritesheetu. GIMP nemá žádné podpůrné nástroje pro tvorbu animací. Ty proto vznikaly tak, že byly nakresleny jejich jednotlivé snímky, v Unity se z nich vytvořila animace a teprve v té chvíli si bylo možné animaci prohlédnout. Kvůli neintuitivnosti a časové náročnosti tohoto procesu se nakonec postava hráče stala jediným objektem ve hře, který disponuje animacemi složenými z více snímků.



Obrázek 4.1: Program GIMP

4.2 Postava hráče

Postava hráče využívá stejnou implementaci stavového automatu jako jednotka nepřítele (podrobněji popsáno v sekci 4.4). FSM je typičtější používat pro nehráčské postavy, i zde se dá ale využít jeho výhod. Kód je logicky členěn do souborů podle jednotlivých činností, které může hráč vykonávat. Přechody mezi jednotlivými stavy jsou v tomto případě většinou podmíněny stiskem nějaké klávesy.

První implementací pohybu postavy bylo použití funkce *Transform.Translate*. Ta na vstupu dostává vektor, který přičte k současným souřadnicím herního objektu a následně ho na tyto souřadnice přemístí. Pokud do funkce dosadíme směrový vektor odpovídající aktuálně stisknutým směrovým klávesám vynásobený rychlostí a budeme ji volat v metodě *Update*, pak se bude objekt pohybovat daným směrem a danou rychlostí. Metoda *Update* je ale volána každý snímek a snímková frekvence hry nemusí být konstantní. Pro dosažení neměnné rychlosti je ještě třeba vynásobit směrový vektor hodnotou *Time.deltaTime*, která je rovna převrácené hodnotě snímkové frekvence.

Aby spolu dva herní objekty v Unity mohly kolidovat (např. aby hráč neprošel stolem, ale zastavil se o něj), musí oba objekty mít nějaký collider a jeden z nich navíc komponentu *RigidBody2D* (dále jen RB). Po přidání této komponenty začne herní objekt podléhat fyzice. Komponenta RB byla přidána na postavu hráče a díky tomu lze s postavou pohybovat nastavováním proměnné RB.*velocity*, u které už není třeba řešit snímkovou frekvenci. U komponenty RB bylo potřeba nastavit gravitační zrychlení na nulu (u top-down her se gravitace obvykle nevyužívá) a uzamknout

rotaci na ose z. Dále byl nastaven druh kontroly kolizí na *Continuous*, aby byly kolize detekovány i při vyšších rychlostech.

Posledním důležitým nastavením u RB je interpolace. Jelikož fyzické výpočty v Unity běží na konstantní frekvenci a grafika je vykreslována s proměnnou frekvencí, tak se pohyb herních objektů, které podléhají fyzice, může vykreslovat sekaně. Toto je zřetelné především u objektů následovaných kamerou, což je právě postava hráče, a zapnutím interpolace se tomuto jevu dá zabránit. [8]

Po implementaci pohybu byla hráči přidána možnost útočit s chladnou zbraní. Soubojový systém je něco, co může během vývoje videohry zabrat velké procento času. Dopad na jeho celkový dojem má mimo jiné i to, jak dobře jsou nakreslené a načasované animace. Implementace v tomto projektu se ale animacím opět vyhýbá. Místo toho se pro vizualizaci útoku v čase mění souřadnice a rotace herního objektu reprezentujícího zbraň.

Útok se zahájí kliknutím a pozice kurzoru určuje jeho směr. Po kliknutí se zbraň otáčí okolo hráče nejdříve pomalu (napřáhnutí) a pak rychle opačným směrem (švih), až se nakonec zastaví na místě odpovídajícímu pozici kurzoru (zásah). Nejprve se zjistí úhel mezi hráčem a zbraní ve své konečné pozici, k jeho hodnotě pak v čase dle potřeby přičítá nebo odčítá a nakonec je převeden zpět na souřadnice. Úhel je vypočten jako $\alpha = \arctan\left(\frac{v.y}{v.x}\right)$, kde v je směrový vektor od hráče ke kurzoru. Pro převod úhlu na souřadnice se pak používá funkce sinus pro složku y a cosinus pro složku x.

Dále byla přidána hráči možnost úskoku (dash). Zde opět hraje roli komponenta RB, konkrétně její funkce *AddForce* aplikující na objekt sílu, která ho posouvá v čase. Právě kvůli vysoké rychlosti úskoku je třeba u postavy hráče nastavit detekci kolizí na *Continuous*, jinak by hráč mohl pomocí této mechaniky procházet např. skrz stěny. Pro pocit rychlosti byl také přidán vizuální efekt, který za postavou během pohybu zobrazuje její částečně průhledné kopie, kterým se jejich průhlednost postupně zvyšuje, až úplně zmizí. Jednotlivé snímky pak mohou připomínat fotografii zachycující pohybující se objekt.

Kód těchto vizuálních kopií hráčské postavy si získá referenci na barvu jejího spritu, postupně odečítá od hodnoty alfa kanálu a při dosažení nulové hodnoty objekt smaže. Pokud by se toto odečítání provádělo každý snímek v metodě *Update*, muselo by být odečítané číslo velmi malé (aby objekt nezmizel okamžitě) a opět by se musel řešit problém variabilní snímkové frekvence. Pro implementaci logiky, která se má odehrávat napříč několika snímky, je v Unity vhodné použít tzv. *Coroutine*. Jedná se o koprogram, ve kterém lze použít např. funkci *WaitForSeconds*. Tato funkce zabalená v cyklu s nějakou jinou činností znamená, že daná činnost se provede každých n vteřin.

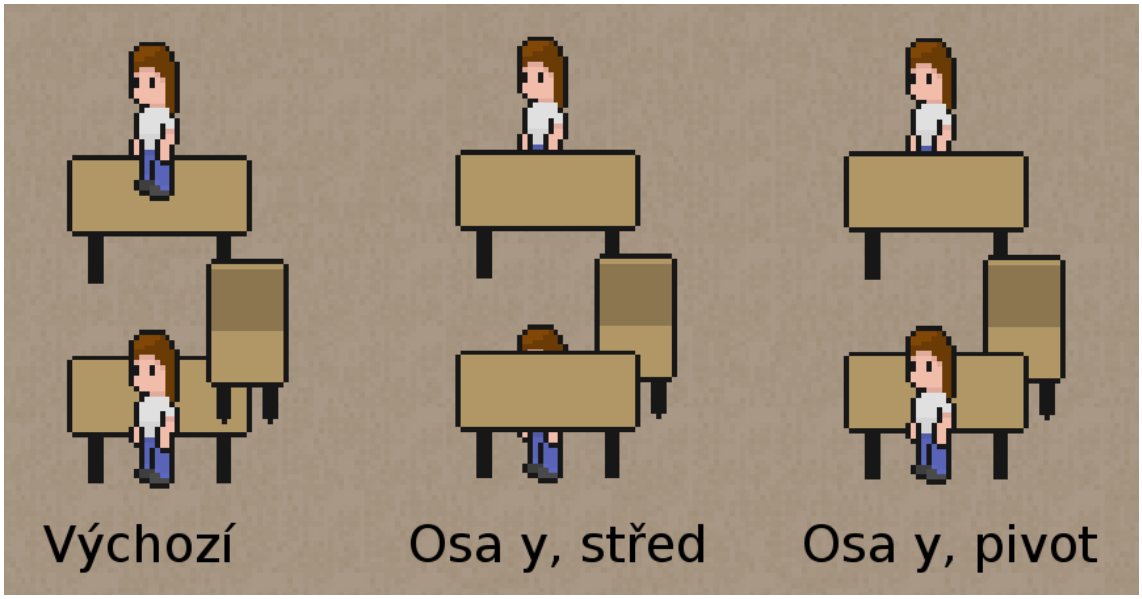
4.3 Herní mapa

Pro vytvoření mapy byla použita tilemap se čtvercovými dlaždicemi o velikosti 16 krát 16 pixelů. Tímto nástrojem byla do mapy položena podlaha a stěny budovy. Objekty rozmístěné po mapě (stůl, koš, dveře atd.) jsou každý herním objektem,

který má obvykle nějaký collider, C# skript a popřípadě RB. Pro povrchy podlahy byl často používán nástroj *Rule Tile*, aby dlaždice na krajích a v rozích místností měly odlišný vzhled oproti těm ostatním.

Nástroj tilemap umožnil rychlou tvorbu a případnou editaci rozložení místností na mapě, nesl s sebou ale jeden problém. Při pohybu hráče a kamery, která ho následuje, mezi jednotlivými dlaždicemi problikávaly šedé čáry. Jedná se o mezery vzniklé nedokonalým vykreslováním, kdy jsou některé dlaždice vykresleny na jiném místě, než by měly. Vzniklé mezery jsou sice velmi malé, ale je skrz ně vidět pod povrch podlahy na pozadí scény (které je nastaveno na šedou barvu) a jedná se o nepříjemný vizuální bug. Tento problém se řeší přidáním tzv. bleeding edge na postižené sprity. Jedná se o dodatečný několikapixelový okraj, jehož pixely barvou kopírují původní okraj spritu. Unity pro toto řešení nabízí tzv. sprite atlas, kde stačí vybrat, jaké sprity do něj mají být zabaleny, a o zbytek se engine postará.

Dalším problémem v této fázi implementace byla zvolená perspektiva. Pokud totiž hráč stojí před stolem, musí být vykreslen blíže ke kameře než stůl. Pokud ale stojí za stolem, musí být blíže vykreslen zase stůl. To znamená, že pořadí vykreslování jednotlivých spritů se mění za běhu. Správné řazení herních objektů v této perspektivě odpovídá jejich pozici na ose y. Čím níže na mapě se objekt nachází, tím blíže ke kameře by měl být vykreslen. Toto chování lze zapnout v nastavení projektu a ještě je nutné u všech herních objektů nastavit, aby se řadily podle svých pivotů (nikoliv podle svých středů). Tyto pivoty musí mít všechny sprity umístěny ve své spodní části, protože na tomto místě by se v reálném prostředí dotýkaly země.



Obrázek 4.2: Řazení spritů ve zvolené perspektivě

4.4 Nepřátelská jednotka – implementace

4.4.1 Stavový automat nepřítele

Implementace jednotky nepřítele byla časově nejnáročnějším bodem tohoto projektu. Do hry byl přidán pouze jeden druh nepřítele, struktura implementace ale počítá s přidáváním nových druhů nepřátel a snaží se tento proces usnadnit. Každý druh nepřítele obsahuje skript (v případě implementované nepřátelské jednotky se jedná o skript *Enemy1*), který je jeho komponentou a dědí z obecného skriptu *Enemy*. Tento obecný skript obsahuje atributy a logiku, kterou by měl mít každý druh nepřítele, jako zdraví, zranění, pathfidning logika (hledání cesty), nebo detekce hráče. Důležitá je proměnná *currentState*, která je typu *EnemyState*.

EnemyState představuje obecný stav pro nepřátelský FSM. Obsahuje referenci na nepřítele, který ho využívá, a virtuální metody, které jsou přepisovány v potomcích. Takovým potomkem je např. *EnemyPatrolState* neboli stav hlídky. Každý stav obsahuje metodu *Enter*, která se stará o inicializaci, a *Update*, která popisuje chování nepřítele v daném stavu. Pokud nepřítel přejde do nového stavu, tak ho nastaví jako svůj *currentState* a zavolá jeho metodu *Enter*. Poté je každý snímek volána stavová metoda *Update*, dokud nepřítel nepřejde do stavu jiného.

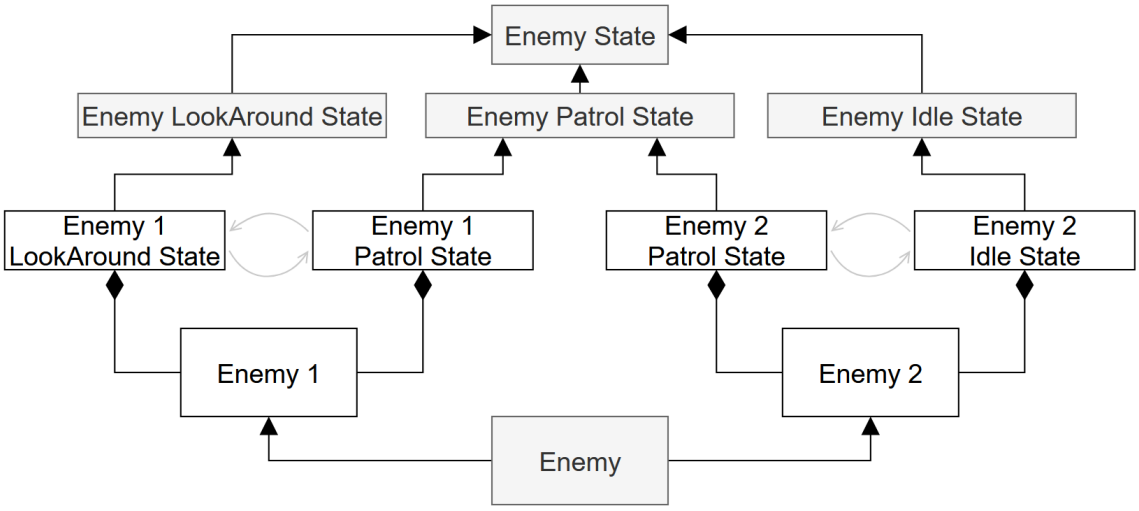
Stav má také vlastnosti typu boolean s názvem začínajícím prefixem „*End_*“, které jsou v metodě *Enter* nastaveny na false, a při splnění určitých podmínek mohou být v metodě *Update* nastaveny na true. Změna hodnoty těchto vlastností reprezentuje přechod do jiného stavu, např. ve stavu hlídky znamená nastavení vlastnosti *End_PatrolPointReached* na true, že se nepřátelská jednotka dostala na aktuálně cílené stanoviště a měla by v tomto případě přepnout do stavu rozhlížení. Popisované vlastnosti mají protected getter, což naznačuje další vrstvu dědičnosti.

Touto vrstvou jsou už přímo implementace stavů pro jednotlivé druhy nepřátel, kteří je obsahují v podobě kompozice. *Enemy1* má v sobě tedy *Enemy1PatrolState*, který dědí z *EnemyPatrolState*. Tento konkrétní stav ve své metodě *Update* volá nejdříve stejnojmennou metodu svého předka a následně kontroluje, zdali vlastnost *End_PatrolPointReached* není rovna true. Pokud ano, tak přepne nepřítele (na kterého má referenci) do stavu rozhlížení (*Enemy1LookAroundState*).

Implementace tímto způsobem odděluje univerzální logiku jednotlivých stavů od nastavení přechodů u konkrétních druhů nepřátel. Pokud by se měl např. přidat nový druh nepřátelské jednotky, která se po dosažení stanoviště ve stavu hlídky namísto rozhlížení na pár vteřin zastaví a bude nečinná, není třeba znovu implementovat logiku hlídky. Vytvořil by se stav *Enemy2PatrolState*, který by dědil z již naprogramovaného stavu *EnemyPatrolState*, a nově by se musel vytvořit pouze stav *EnemyIdleState* a k němu následně *Enemy2IdleState*.

4.4.2 Zorné pole

Jak již bylo zmíněno v sekci návrhu, nepřátelská jednotka může detekovat postavu hráče, pokud jsou splněny dvě podmínky. První podmínkou je, že se hráč musí nacházet v nepřítelově zorném poli, které má tvar kruhové výseče. Druhá podmínka



Obrázek 4.3: Implementace stavového automatu

tvrdí, že hráč musí být viděn přímo. Mezi hráčem a nepřítelem tedy nesmí být žádná stěna a pokud je navíc hráč skrčený, nemůže mezi nimi být ani žádný objekt jako stůl, židle, koš apod. Postavy hráče a nepřítele jsou spojeny pomyslnou úsečkou, jejíž vlastnosti určují splnění těchto podmínek.

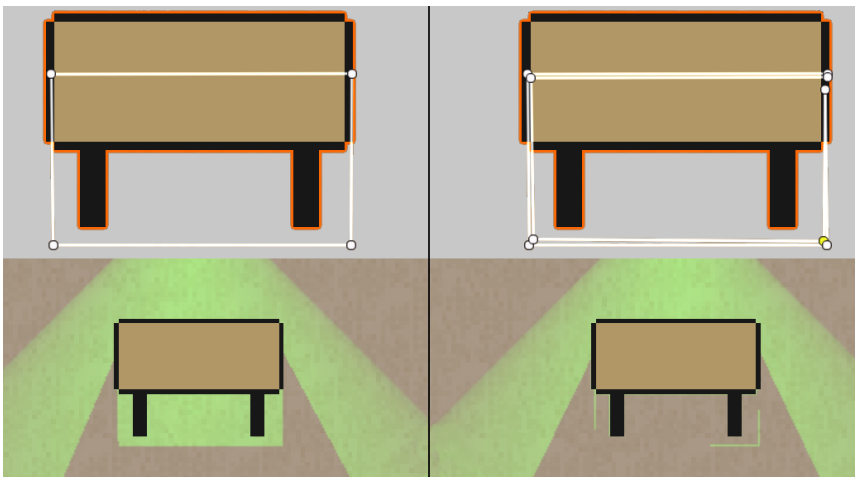
První je kontrolována délka úsečky. Pokud je delší než poloměr výseče *viewDistance*, znamená to, že se hráč nachází mimo zorné pole. Nepřítel se kouká směrem na hráče, pokud platí $|facingAngle - enemyToPlayerAngle| \leq \frac{FOV}{2}$, kde FOV značí velikost středového úhlu kruhové výseče. Druhá podmínka je kontrolována voláním Unity funkce *Physics2D.Linecast*.

Aby hráč věděl, kde může a nemůže být spatřen, je zorné pole nepřátel ve hře vizualizováno. Logika vizualizace je nezávislá na skutečné implementaci logiky spatření hráče a není stoprocentně přesná. Na herní mapě je pro každého nepřítele vykreslena kruhová výseč, která odpovídá danému zornému poli a jejíž tvar je následně upraven, pokud se v něm nachází nějaké neprůhledné objekty.

Pro vykreslení výseče je použit balík URP a jeho komponenta *Light 2D*. Jak název napovídá, tato komponenta vrhá na scénu světlo s nastavitelným tvarem, barvou a intenzitou. Konkrétně každá nepřátelská jednotka využívá dvou bodových světel. To hlavní je buď zelené nebo žluté a jeho hodnoty odpovídají tvaru kruhové výseče. Sekundární světlo je červené s proměnlivým poloměrem a informuje o stavu detekce.

Každý neprůhledný objekt pak na sobě musí mít komponentu *Shadow Caster 2D*, přes kterou neprochází světlo. Tvar neprůsvitné oblasti je nutné nastavit ručně v editoru a pokud se pracuje s průhlednými sprity, situace se ještě více komplikuje (viz obrázek 4.4). Tyto komponenty je nutné zapínat a vypínat za běhu podle toho, zdali je hráč v režimu plížení. Pokud se toto ve hře provedlo opakovaně (hráč se několikrát skrčil a zase postavil), nastaly velké problémy s výkonem, kdy snímková frekvence klesla z řádu stovek na jednotky FPS. Ukázalo se, že se jedná o chybu,

která byla zatím opravena pouze v Unity 2022[1]. Projekt tedy musel být upgradován na novější verzi Unity.



Obrázek 4.4: Shadow Caster 2D

4.4.3 Hledání cesty

Nepřátelská jednotka má v sobě implementovaný algoritmus A* pro nalezení cesty k jejímu aktuálnímu cíli. A* vychází z Dijkstrova algoritmu, ke kterému přidává heuristickou funkci. Hodnota heuristické funkce je u každého vrcholu grafu přičítána k jeho ceně a tím vzniká kombinovaná cena vrcholu, která určuje, jaký další dočasný vrchol bude vybrán při hledání cesty. Charakteristickým rysem tohoto algoritmu je, že byl navržen tak, aby využil znalosti cílových souřadnic a tím minimalizoval čas potřebný k nalezení řešení [9].

Jako heuristická funkce byla použita euklidovská vzdálenost pomocí *Vector2Int.Distance*. Vrcholy pomyslného grafu jsou jednotlivé dlaždice z tilemap. Pokud je pomocí *Physics2D.OverlapArea* zjištěno, že se na nějaké dlaždici nachází objekt, je dlaždice (vrchol) prohlášena za neprůchozí a ignorována při hledání cesty.

[1] Nahlášené chyby a reakce vývojářů lze sledovat na webu Unity Issue Tracker: https://web.archive.org/web/20230425215912/https://issuetracker.unity3d.com/issues/performance-loss-when-enabling-and-disabling-shadowcaster2d

5 Kritické zhodnocení

V rámci projektu byla vytvořena demonstrativní videohra pro platformy Windows a Linux. Hra obsahuje kompletní příběhový mód a během testování se nevyskytly žádné závažné technické problémy.

5.1 Nedostatky

5.2 Návrh na zlepšení

6 Závěr

Seznam použité literatury

1. UNITY TECHNOLOGIES. *Unity - Manual: Asset workflow* [online]. 2023-04. [cit. 2023-04-26]. Dostupné z: https://docs.unity3d.com/2022.2/Documentation/Manual/AssetWorkflow.html.
2. UNITY TECHNOLOGIES. *Unity - Manual: Important Classes - GameObject* [online]. 2023-04. [cit. 2023-04-26]. Dostupné z: https://docs.unity3d.com/2022.2/Documentation/Manual/AssetWorkflow.html.
3. GOLDSTONE, Will. *Unity 4.3: 2D Game Development Overview* [online]. 2013-11. [cit. 2023-04-26]. Dostupné z: https://blog.unity.com/technology/unity-4-3-2d-game-development-overview.
4. JAN, Matej. *Game developer's guide to graphical projections (with video game examples)* [online]. 2017-11. [cit. 2023-05-05]. Dostupné z: https://medium.com/retronator-magazine/game-developers-guide-to-graphical-projections-with-video-game-examples-part-1-introduction-aa3d051c137d.
5. SAMUELSON, Gustav. *Pixel art - The Medium of Limitation* [online]. 2020. [cit. 2023-05-05]. Dostupné z: http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1518210&dswid=-4613.
6. DILL, Kevin. *What Is Game AI?* [online]. 2013-09. [cit. 2023-05-08]. Dostupné z: http://www.gameaipro.com/GameAIPro/GameAIPro_Chapter01_What_is_Game_AI.pdf.
7. JAGDALE, Devang. Finite State Machine in Game Development. 2021, s. 384–390. Dostupné z DOI: 10.48175/IJARSCT-2062.
8. UNITY TECHNOLOGIES. *Unity - Scripting API: Rigidbody.interpolation* [online]. 2023-05. [cit. 2023-05-11]. Dostupné z: https://docs.unity3d.com/2022.2/Documentation/ScriptReference/Rigidbody-interpolation.html.
9. ADEGUN, Adekanmi; OGUNDOKUN, Roseline; OGBONYOMI, Samuel; SADIKU, Peter. Design and Implementation of an Intelligent Gaming Agent Using A* Algorithm and Finite State Machines. *International Journal of Engineering Research and Technology*. 2020, roč. 13, s. 191. Dostupné z DOI: 10.37624/IJERT/13.2.2020.191-206.