Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Hra typu tower defense v Unity engine

2024

Pavel Doležel

Vedoucí práce:
Mgr. Radek Janoštík, Ph.D.

Studijní program: Informatika,
Specializace: Programování a vývoj software

Bibliografické údaje

Autor:	Pavel Doležel
Název práce:	Hra typu tower defense v Unity engine
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2024
Studijní program:	Informatika, Specializace: Programování a vývoj software
Vedoucí práce:	Mgr. Radek Janoštík, Ph.D.
Počet stran:	33
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliographic info

Author:	Pavel Doležel
Title:	Tower defense game in Unity engine
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2024
Study program:	Computer Science, Specialization: Programming and Software Development
Supervisor:	Mgr. Radek Janoštík, Ph.D.
Page count:	33
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Práce se věnuje tématu tvorby hry typu tower defense v Unity engine. Popisuje tvorbu hry „Defend it, Private!“ z období druhé světové války, se zaměřením na volbu užitého editoru, jeho detaily, zabývá se grafickou i zvukovou stránkou hry a zmíněny jsou i řešení problémů, které lze při implementaci hry očekávat. Práce je logicky členěna a představuje tak uceleného průvodce tvorbou hry typu tower defense.

Synopsis

This bachelor's thesis is dedicated to creating a tower defense game in the Unity engine. The thesis portrays the making of the game "Defend it, Private!" from the World War II period with a focus on choosing the right code editor and engine, which details and oversees graphics and audio of the game. Troubleshooting of the issues that were dealt with during the production of the game is mentioned as well. The thesis is coherently articulated and provides a complete guide throughout the tower defense game making.

Klíčová slova: 2D videohra; pixel art; bránění oblasti; druhá světová válka

Keywords: 2D videogame; pixel art; tower defense; World War II

Prostřednictvím těchto pár řádků bych rád poděkoval vedoucímu mé bakalářské práce Mgr. Radku Janoštíkovi, Ph.D. To s jeho pomocí se moje téma objevilo ve školním systému, a to díky němu jsem byl schopen pracovat na něčem, co mě skutečně naplňuje a rozvíjí. Věřím, že jsem jeho důvěru, ve mne vloženou, nezklamal.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

1 Úvod

V mé bakalářské práci se věnuji tématu Hra typu tower defense v Unity engine. Mým cílem je vyvinout a naprogramovat počítačovou hru žánru tower defense, kteroužto jsem zasadil do období druhé světové války.

Tower defense je videoherní žánr, který je zaměřen na obranu základny před útoky nepřátelských jednotek. Ty postupem času nabývají na síle, a tím se obrana stává složitější. Aby nebyl hráč znevýhodněn, dochází v průběhu hry i k zesílení jeho vlastních obranných jednotek. Toho bývá docíleno několika způsoby. A to ať přidáním nových (silnějších), či odemknutím možnosti vylepšení již dosavadních jednotek. Samotným vylepšením je pak myšleno většího poškození, zkrácení přebíjecí doby jednotky, či vyšší počet životů. Velmi často dochází ke kombinaci zmíněných způsobů, a to právě za účelem vybalancování hry.

Základ videohry bude tvořit postupně rozvíjející se kampaň, při níž se hráč seznámí jak s mechanikami hry, tak i s jednotkami, jež budou pro obranu základny potřebné. Pohyb nepřátelských sil, jež budou útočit na spojeneckou základnu, nemusí probíhat pouze po jedné, předem definované trase. Těch může být hned několik. Dalším klíčovým aspektem pro splnění dané úrovně bude správné nakládání s herní měnou. Ta se generuje automaticky, ale lze ji získat i při zničení nepřátelských jednotek. Pro splnění jednotlivých úrovní tak bude potřeba taktického myšlení.

Tato práce je zaměřena na samotný vývoj videohry, ať již po programátorské stránce, tak i po té vizuální. Opomenuty nebudou ani na různé chyby a příkoří, které se během vývoje vyskytly a jejich následné řešení. Závěr poté shrnuje mechaniky a principy, kterých bylo při vývoji využito.

2 Volba editoru

Vývoj kterékoliv videohry je možné pro jednoduchost rozdělit do dvou, na sebe nezávislých proudů. A to na vývoj vizuální a na programovací. Právě o druhé části bude pojednávat následující kapitola. Na začátku vývoje jakékoliv aplikace je důležité provést rozhodnutí, v jakém vývojovém prostředí/editoru bude aplikace vyvíjena. Zatímco správná volba jej posouvá vpřed, zlá jej může zpomalit. K dnešnímu dni existuje celá řada editorů, ať již těch bezplatných, či zpoplatněných. Mezi nejznámější prostředí patří: Unreal Engine, Unity a Defold. V následujících podkapitolách tak budou popsány jak jednotlivé plusy, tak i mínusy jednotlivých enginů. V poslední podkapitole pak rozhodnutí, který vývojový editor bude použit pro mou práci.

2.1 Unreal Engine

První verze Unreal Engine [1] byla vytvořena společností Epic Games v roce 1998 pro hru Unreal. Jedná se tak o jeden z nejstarších, nicméně stále populár-

ních, enginů. Díky jeho oblibě je v současnosti využíván nejen pro programování střílečeк z první osoby, ale také pro jiné typy her. [2]

Programovacím jazykem aplikace je C++[3], který je při práci náchylnější k únikům paměti oproti jazyku C#. Unreal engine současně podporuje tzv. *visual scripting* [4], v němž se programuje vizuálně, pomocí skládání bloků kódu. K dosažení nejlepšího výkonu aplikace je dobré zkombinovat jak klasické programování, tak i již zmíněný visual scripting.

Donedávna byl tento editor preferovanější než Unity, a to právě kvůli dosahování lepších grafických výsledků. Unity ale nezaostává a postupně Unreal Engine dohání. Nicméně, pokud nemá být výsledný produkt podobný AAA titulu (videohra s ohromným rozpočtem a masivní propagační kampaní [5]), rozdíl mezi enginy není žádným způsobem markantní. Posledním hlavním rozdílem, mezi Unity a Unreal Enginem jsou dostupnost tutoriálů a tzv. „plug-in“ komponenty. Unreal Engine velmi zaostává v počtu různých tutoriálů, kterých lze na internetu bezplatně dohledat. „Plug-in“ komponentu si čtenář může představit jako objekt či skupinu objektů, jež si vývojář do své aplikace vloží z repozitáře dostupného daným enginem. Z pravidla se jedná o komponenty, které zkrášlují samotný vzhled. Mnoho komponent, které bychom označili za tzv. „plug-in“, se již v tomto vývojovém prostředí nachází hned po vytvoření projektu.

2.2 Defold

K vytváření především dvourozměrných her se používá multiplatformní herní engine Defolt [6] vyvinutý společností King, později pak nadací Defold Foundation. V tomto enginu je sice možné plnohodnotně vytvářet i trojrozměrné hry, zaměřuje se však na projekty menší velikosti. Pro vývoj využívá Defold méně známý programovací jazyk Lua [7], který je pro začátečníky snazší na naučení, ale je odlišný od ostatních programovacích jazyků. Používá rozdílné typování (minimální počet klíčových slov), neobsahuje třídy ani dědičnost a prozatím nedisponuje vizuálním skriptováním. Nicméně principy OOP mohou být dodány pomocí funkcí a tabulek. To má za následek menší rozšířenost mezi vývojáři, a tím i nižší počet tutoriálů.

Na druhou stranu tuto nevýhodu zmírňuje fakt, že vývojářská komunita je aktivní a otevřená. Vývojáři vydávají nové verze v měsíčních intervalech, z nichž první dva týdny probíhá tzv. „public beta“ verze, po které následuje teprve verze oficiální. [8] I díky tomu jsou jeho velkou předností stabilita a zpětná kompatibilita.

2.3 Unity

Engine Unity [9] byl vyvinut společností Unity Technologies a první verze byla vydaná v roce 2005. Aktuální verze, s označením 2022.3 LTS [10] se využívá k tvorbě nejen počítačových, ale i konzolových her. [11] Unity se od Unreal Enginu a Defoldu liší tím, že se v něm veškeré skripty píší v programovacím jazyce

C#. Ten poskytuje jisté výhody vůči C++, jakou je ku příkladu snazší syntax. Další předností je pak automatická správa paměti. Výpočetní výkon, který je touto automatickou správou paměti ubírán, však ovlivnil mou bakalářskou práci jen zcela nepatrně. Neméně podstatným rozdílem mezi Unity a Unreal Engine je počet tzv. „plug-inů“ [12], které se v Unity, na rozdíl od Unreal Enginu, nachází. Ať již těch, které vylepšující grafickou stránku aplikace, tak i těch, upravující zvukový podklad videohry.

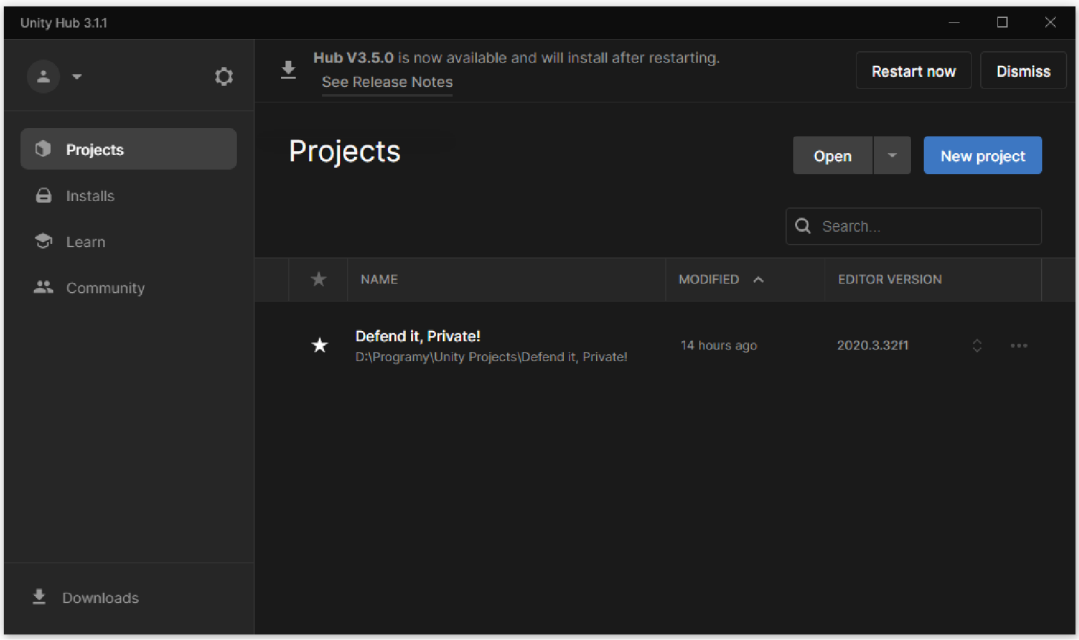
Unity je uzpůsobeno pro tvorbu méně náročných počítačových her. Touto náročností se pak myslí především detail textur, množství stínů vyskytujících se v jednotlivých scénách a podobné detaily. Jako příklad počítačové hry, vytvořené v tomto enginu, mohu uvést populární hru „Fall Guys: Ultimate Knockout“ [13], která vyšla na konci roku 2020. Oblíbenost tohoto enginu napříč programátorskou komunitou dokládá i fakt, že je možné nalézt na internetu velké množství nejrůznějších tutoriálů. Ty mohou jednotlivým programátorům pomoci nejen při jejich začátcích, ale také s řešením problémů, jež se mohou objevit v rámci vývoje.

2.4 Rozhodnutí

Vzhledem k povaze tower defence hry, mým potřebám pro její naprogramování i samotný programovací jazyk C#, v němž jsou skripty psány, jsem se rozhodl zvolit právě Unity. Výsledná hra „Defend it, Private!“ nevyžaduje detailnější textury ani stíny, tím pádem plnohodnotně využiji potenciál tohoto enginu. Nemalou výhodou je i má dřívější zkušenost a praxe právě s Unity.

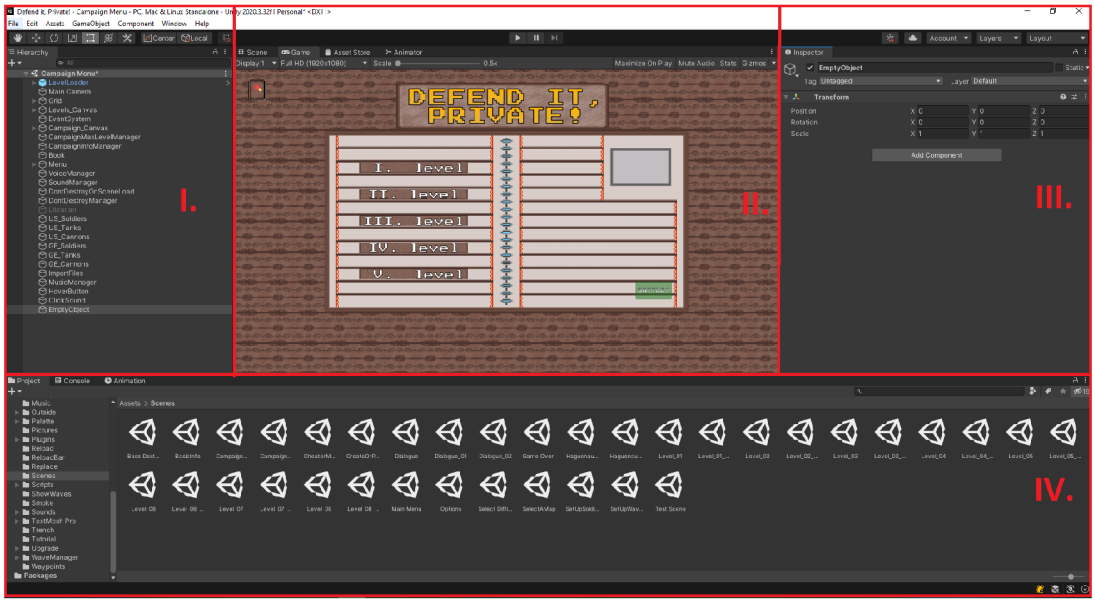
3 Seznámení s Unity

Prostřednictvím této kapitoly se čtenář seznámí s prostředím, v němž bude celá videohra vznikat. Tou je, pro rozsah mé bakalářské práce, bezplatná aplikace Unity Hub. Po jejím spuštění vývojáře přivítá úvodní menu. V něm se nachází veškeré odkazy na projekty, jež na daném zařízení doposud vznikly. Jak je ostatně možné spatřit na obrázku 1.



Obrázek 1: Unity Hub

Po spuštění projektu, jež byl v Unity vytvořen, se vývojář ocitne v samotném srdci dané aplikace. Aktuální prostředí se nazývá Editor. To v něm probíhá samotný vývoj. Ten je možno si rozdělit na několik nezávislých částí (zobrazených na obrázku 2), jejichž funkcionalita bude v následujících podkapitolách popsána.



Obrázek 2: Unity prostředí

3.1 Herní objekty

V první části jsou zobrazeny veškeré objekty, jež se v aktuální scéně nachází. Ty je poté možno rozdělit do skupin na tzv.: *Empty Object*, *2D Object*, *3D Object*, *Effects*, *Lights*, *Audio*, *Video*, *UI* a v poslední řadě *Camera*. *Empty Object* neobsahuje žádnou komponentu s výjimkou komponenty Transform, která je součástí každého objektu. Tu není možno odebrat. O tom, co to „komponenta“ je pojednává podkapitola 3.3. *2D Object* byl během vývoje videohry použit nejčastěji, a to z důvodu snadné modifikace. Zbylé skupiny, vyjma objektu *Camera*, jsou rozděleny do podskupin. Objektů, jež se v nich nachází nebylo během vývoje využito, a tak nebude ani jejich funkcionalita blíže upřesněna.

3.2 Scény

Ve druhém okně se nachází vizuální stránka scény. Navigační lištu tohoto okna je pak možné rozdělit dle záložek na: *Scene*, *Game*, *Asset Store* a *Animator*. Záložka *Scene* zobrazuje veškeré objekty, jež se ve scéně nachází. Jakým způsobem dochází k modifikaci vlastností objektů bude pojednávat podkapitola 3.3. *Game* poté zobrazuje samotný vzhled scény. Tato záložka pak obsahuje podkategorie, kterými jsou: *Display Option*,*Resolution Settings* a *Scale*. *Display Option* slouží k výběru kamery, jejíž vizuál se má zobrazit. A to pro případ, kdyby se v dané scéně nacházelo kamer více. Pomocí *Resolution Settings* si vývojář může nastavit rozlišení, v němž má být daná scéna vyobrazena. Poslední záložka slouží k přibližovaní/oddalování aktuální scény. *Asset Store* si může čtenář představit jako uložiště, v němž se nachází velké množství textur a modelů, jež byly vytvořeny komunitními fanoušky. Ty jsou poté vývojáři propůjčeny zadarmo, či za poplatek. Jelikož jsem grafickou stránku všech objektů vytvářel sám, hlubšího a detailnějšího popisu *Asset Storu* se vyvaruji. Poslední nepopsanou záložkou je *Animator*. Ten slouží k vytváření a úpravě animací herních objektů.

3.3 Komponenty

Ve třetí části pak dochází k modifikaci samotných herních objektů. K tomu je nejprve potřebné označit objekt, který se nachází v prvním okně. Po jeho označení dojde k zobrazení jednotlivých komponent, tedy vlastností, kterými objekt disponuje. Každý z objektů, jež se v Unity nachází je složen minimálně ze: *jména*, *tagu*, *vrstvy*, *identifikátoru typu objektu* a komponenty *Transform*. Komponenty je pak možné objektu přiřadit i odebrat, přičemž v této podkapitole budou vypsány pouze ty nejdůležitější. Popis všech komponent není možný z důvodu velkého množství, jimiž Unity disponuje. Přidání komponenty probíhá pomocí tlačítka *Add Component*, které je umístěno pod poslední komponentou daného objektu. Všechny objekty, které mají být zobrazeny musí disponovat komponentou *Sprite Renderer* či *Image*. Kromě obrazu je možno pomocí této komponenty nastavit *obarvení objektu*. Dále pak *průhlednost* či *materiál*, které mění vizuál objektu. Další komponentu, kterou je důležité zmínit nese název *Rigidbody 2D*. Ta slouží

k nastavení fyzikálních vlastností objektu. Pomocí ní je možné nastavit hmotnost, posunutí a další, pro mou práci nedůležité, vlastnosti. Poslední důležitou komponentou je *Collider 2D*. Přesněji se pak dělí dle oblasti, v němž má být kolize detekována. A to na kruhovou, obdélníkovou, či polygonovou. Pomocí tzv. *offsetu* je možné kolizní oblast posunout. V takovém případě se nemusí nacházet přímo v daném objektu. Poslední krok pak slouží k nastavení oblasti, v níž má objekt reagovat s ostatními.

3.4 Pracovní prostor

Čtvrté, a tedy poslední okno představuje pracovní prostor. V něm je uložena veškerá hierarchie ať již obrázků, *prefabů*, scén či modelů, kterými aplikace disponuje. Prefab je objekt, který může být použit i ve více scénách. Jeho hlavním rozdílem oproti ostatním objektům je ten, že modifikací jednoho prefabu stejného druhu dojde automaticky k modifikaci zbylých objektů téhož druhu. Po ustanovení hierarchie na začátku projektu je důležité ji neměnit. Opak poté vede k nekonzistencím, které nejsou v žádném vývoji žádoucí.

4 Vývoj hry

Vývoj jakéhokoliv videohry je možné rozdělit do dvou nezávislých, ale sebe navzájem navazujících proudů, a to na vizuální a programovací. O vizuální stránce videohry pojednává kapitola 6. Tato kapitola je zaměřena na vývoj a rozbor jednotlivých skriptů, jež se v mé práci nachází.

4.1 Attack

Prvním důležitým skriptem, který se v mé práci vyskytuje, je *Attack*. Ten je zodpovědný za útoky mezi nepřátelskými a spojeneckými jednotkami. Z důvodu snazší vizuální představy je pod touto podkapitolou umístěn obrázek 3, který zobrazuje diagram aktivit. Samotný skript je složen z atributů, kterými jsou: *počet životů jednotky*; *poškození*; *doba přebíjení útočného předmětu*; *grafické prvky*, které zobrazují, jak počet životů, tak dobu přebíjení. U spojeneckých jednotek dále pak *cena*, kterou bude její vytvoření stát. U nepřátelských pak *odměna*, jíž hráč obdrží po zničení jednotky a *míra poškození báze*, kterou jednotka způsobí, dostane-li se až ke spojenecké bázi. Respektive u obou typů jednotek je možné nastavit i to, co u jeho oponenta, nicméně v těle skriptu dochází k ignoraci těchto dat.

Pod atributy se pak nachází jejich přístupové metody. O hlavní mechaniku skriptu se starají dvě primární metody. A to *Start()* a *Update()*. První z nich je volána před vykreslením prvního snímku dané scény a slouží tedy primárně k inicializačním účelům. Druhé metoda je volána při vykreslení každého snímku, a je tím tak srdcem samotného skriptu. Metoda *Update()* kontroluje, zda-li má

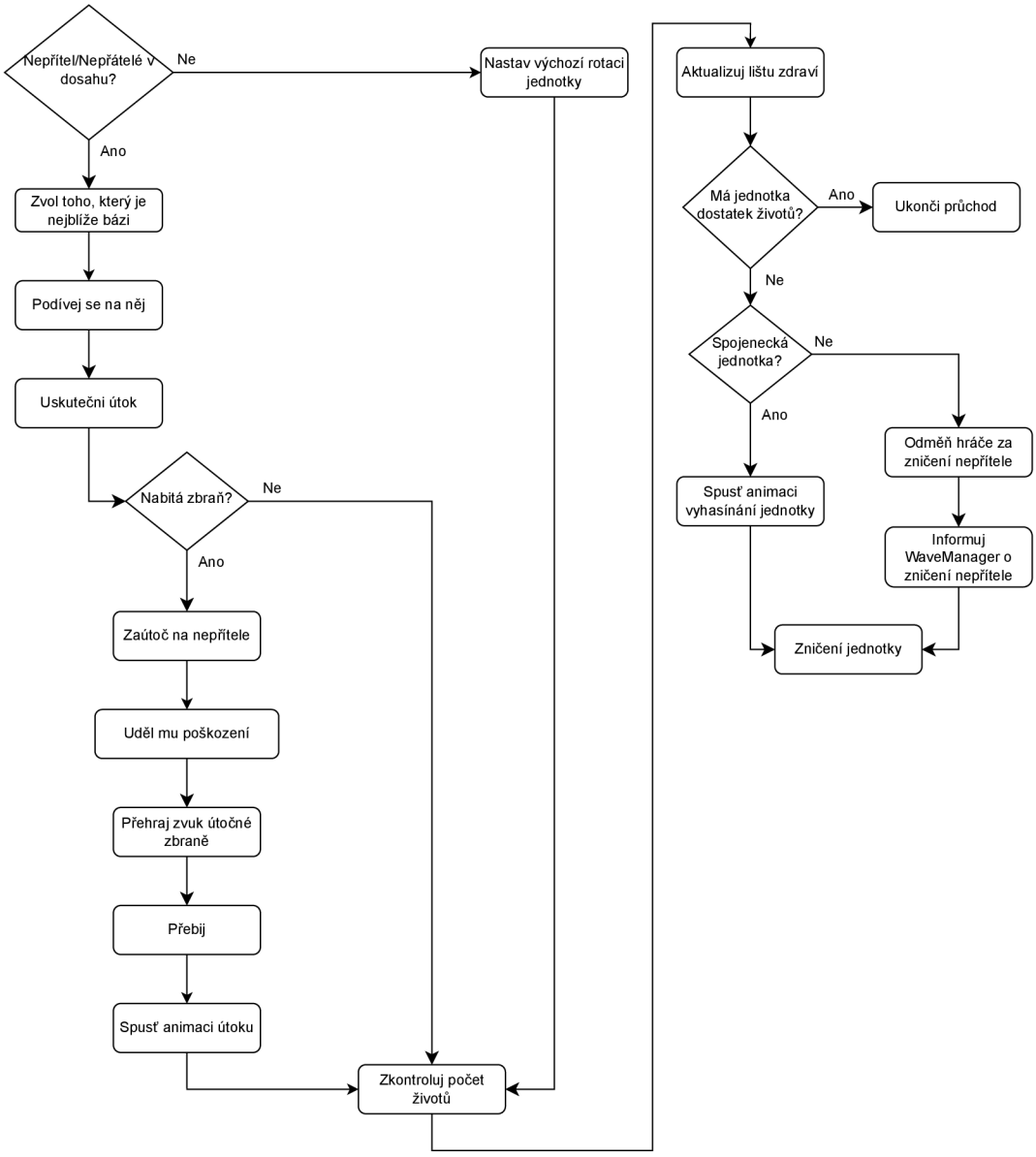
daná jednotka v dohledu cíl, na který by provedla útok. Jestliže ano, otočí se takovým způsobem, aby na svůj cíl hleděla čelem.

S touto mechanikou se však vyskytly komplikace, i přes to, že Unity disponuje metodou *LookAt()*. Ta jako parametr přijímá objekt, na který se má jiný objekt dívat čelem. V mém případě tato metoda bohužel nefungovala dle očekávání. Pro její správné fungování je za potřebí 3D prostředí. Implementace metody, jak se později ukázalo, může být poměrně přímočará a obsahovat pouze pět řádků, vyjma komentářů. Přičemž je důležité zmínit, že byla konkrétní implementace dohledána na internetu. [14]

Po natočení zavolá metodu, která je za realizaci útoku zodpovědná. Ta nejprve zkontroluje, zda má daná jednotka plně nabitou svou útočnou zbraň. Jestliže má, provede následující: zaútočí na nepřítele, přehraje zvuk útočného předmětu, začne přebíjet danou zbraň a přehraje animaci útoku. Tím je metoda, hledající svůj cíl v dohledné vzdálenosti dokončena. Jestliže se žádný cíl v dohledné vzdálenosti nenachází, je napozicování jednotky ponecháno na výchozí.

Po již proběhlém útoku dochází ke kontrole životů dané jednotky. Jestliže je počet životů kladný, dojde k aktualizaci *healthBaru*, jehož úkolem je vizualizace životů jednotky. V opačném případě nastane jedna z následujících událostí. Nejprve se zkontroluje národnost jednotky, jež má být zničena. Jestliže se jedná o spojeneckou, je spuštěna animace starající se o pomalé zhasínání jednotky. Po jejím dokončení je jednotka zničena. Jestliže se jednalo o nepřátelskou jednotku, nastane její okamžitě smazání. Rovněž dojde ke spuštění animace, jež hráče informuje o množství herní měny, kterou hráč obdrží jako odměnu za zničení jednotky. Současně kontaktuje skript *WaveManager* o tom, že byla právě jedna z jeho jednotek zničena.

K nalezení nepřátel pak slouží dvě vestavěné metody, které se v Unity nachází. A to metody *OnTriggerEnter2D()* a *OnTriggerStay2D()*. Ty jsou volány v okamžik, kdy dochází ke kolizi mezi dvěma různými objekty, jež obsahují komponentu Collider2D. Kterou, mimo jiné, obsahují všechny jednotky. První z metod je volána v případě, že došlo k první kolizi mezi danými objekty. Druhá je pak volána při přetrvávání dané kolize. V nich je pak ukryta logika výběru jednotky, na kterou bude v budoucnu zaútočeno.



Obrázek 3: Diagram aktivit skriptu Attack

4.2 WaveManager

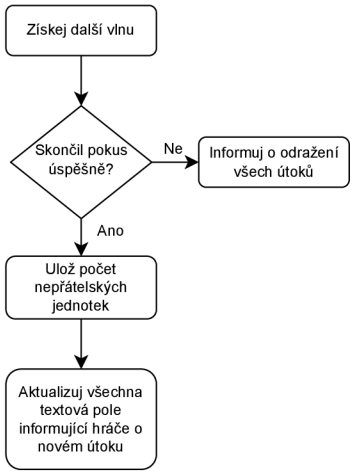
Druhým skriptem je *WaveManager*. Ten je zodpovědný za načítání nepřátelských vojsk v předpřipravených úrovních. Tím jsou myšleny pouze ty úrovně, se kterými se hráč setká v průběhu hlavní kampaně. V uživatelsky definovaných úrovních je pak použit jiný skript. Z důvodu snazší vizuální představy je pod touto podkapitolou umístěn obrázek 4, zobrazující diagram aktivit skriptu. *WaveManager* obsahuje následující atributy: *počáteční a koncový index cest*; *seznam obsahující startovací orientaci nepřátelských jednotek*; *název úrovně*; *textové pole aktuální vlny*, *počtu všech vln* a *doby*, za níž začne probíhat útok nepřátelských jedno-

tek na spojeneckou základnu; *tři celočíselné proměnné*, v nichž jsou uchovány údaje zmíněných textových polí; *třech seznamů obsahující nepřátelskou pěchotu*, *kanóny* a *tanky* a *počet nepřátelských sil*, které aktuálně provádí útok.

Pod atributy se pak nachází tři pomocné soukromé třídy (*Wave*, *Unit* a *Units*), které usnadnily manipulaci s nepřátelskými jednotkami vyskytujícími se ve scéně. První zmíněná třída je složena z atributů indikující: *dobu*, za níž nepřátelské jednotky zahájí útok, *časový rozestup* mezi jednotkami a *seznam daných jednotek*. V ní jsou tedy uchovány informace o aktuálním útoku. Další třída nese název Unit a obsahuje informace o jednotce. Její atributy jsou: *nepřátelská jednotka*, *metoda* zajišťující načtení jednotky ve správném čase a *proměnná* indikující dokončení načtení (jednotky). Poslední třída *Units* je složena z jediného atributu, kterým je *seznam Unit objektů*. Obsahuje tři pomocné metody jako *AddUnit()*, *FindUnit()* a *GetUnits()*. Jejich význam je pak patrný z názvu.

Pod těmito pomocnými třídami se nachází hlavní metody skriptu, kterými jsou *Start()* a *Update()*. V první zmíněné metodě dochází k inicializaci proměnné indikující index aktuální vlny. Následuje zavolání metody *SpawnWave()* s výše popsaným parametrem. V jejím těle poté dojde k zavolání metody *GetNextWave()* (s jistými parametry) která vrací buď hodnotu *null* (sloužící jako indikátor dokončení všech vln) nebo novou nepřátelskou vlnu. Jestliže je vrácena vlna, dojde k jejímu spuštění spolu s nastavením všech textových polí informujících hráče o novém útoku. Jestliže je vrácena hodnota *null* nastane automatické ukončení úrovně. V těle druhé zmíněné metody *Update()* dochází ke kontrole počtu nepřátelských jednotek útočících, v danou chvíli, na spojeneckou základnu. Jestliže jejich počet klesne na nulu je zavolána metoda *SpawnWave()*, jíž význam byl popsán dříve.

Za zmínku stojí ale i ty metody, které v rozboru skriptu chybí. Jejich vyjmenování by totiž vyžadovalo podrobnější rozbor. Z tohoto důvodu budou uvedeny jen ty nejzajímavější, kterými jsou *CallNextWave()*, která vrací logickou pravdivostní hodnotu informující o tom, zdali má dojít k načtení nové nepřátelské vlny. Dále pak *FullCopy()*, jež přijímá jako parametr herní objekt. Ten zkopíruje, nastaví orientaci jednotky, startovací pozici, zobrazení životů a doby přebíjení. Následně zkopírovaný objekt vrací. Poslední metodou, která byla a bude zmíněna je *GetNextWave()*. Ta totiž obsahuje všechny útoky, na které hráč v průchodu kampaně narazí a je tím tak srdcem samotného skriptu.



Obrázek 4: Diagram aktivit skriptu WaveManager

4.3 UpgradeManager

Další rozebíraná třída nese název *UpgradeManager* a je zodpovědná za správné vylepšení schopností a dovedností spojeneckých jednotek. U ní došlo v průběhu vývoje k přepracování. V dřívější implementaci docházelo ke změně typu jednotky po jejím vylepšení. To bylo na základě zpětné vazby opraveno. V tuto chvíli již ke změně typu nedochází. Typ zůstává stejný, ačkoliv nastane vylepšení schopností jednotky.

Třída obsahuje tři soukromé atributy, ve kterých jsou uloženy odkazy na patřičné objekty. Těmi atributy jsou: *_soundManager*, *_moneyBalance* a *_allUnits*. Ty jsou následně v metodě *Start()* inicializovány. Jinými instrukcemi metoda *Start()* nedisponuje. Tato třída se od předchozích, dříve vyjmenovaných, liší tím, že druhou typickou metodu *Update()* neobsahuje. Hlavní metodou celé třídy je metoda *Upgrade()*. Ta je volána s parametry identifikující: *jednotku*, která bude vylepšena, dále pak její *aktuální úroveň*, *fázi vylepšení* a *typ*. V těle metody pak dochází k přepočítání aktuální úrovně na její index, a k vypočítání nové fáze jednotky. Následuje zavolání metody *GetUpgradeList()*, která vrací seznam objektů, ve kterých jsou uloženy potřebné informace k vylepšení jednotky daného typu. Tato metoda přijímá jako parametr *typ* jednotky a *index aktuální úrovně vylepšení*. Dále pak dojde ke kontrole, zdali nebylo dosaženo maximálního vylepšení. Jestliže tomu tak bylo, je metoda *Upgrade()* ukončena. V opačném případě dojde k provedení pomocných operací a instrukcí, na které navazují metody *UpgradeUnit()* a *TakeMoneyForUpgrade()*. V těle prvně zmíněné metody dochází k vylepšení jednotky. To zahrnuje přenastavení počtu životů, síly útoku a doby přebití. Úkolem druhé metody je zpoplatnit hráče za provedené vylepšení.

4.4 ReplaceManager

Třída *ReplaceManager* je zodpovědná za přemisťování již postavených jednotek a tvoří tak základ pro potřeby dynamické obrany. Na začátku této třídy jsou umístěny soukromé atributy, kterými jsou: *isMoving* a *isReplacing*. V prvním jsou uchovávány informace o tom, zdali bylo možné vytvořit kopii přesouvaného objektu, v druhém pak zdali právě probíhá přesouvání jednotky. Dále pak atributy *steady* a *moving* ve kterých jsou uloženy aktivní a pasivní kopie přesouvaného objektu. Níže se nachází pomocný atribut *resetReload*, který indikuje potřebu resetování doby přebití u přesunuté jednotky. Posledním atributem je pak *_buildManager* ve kterém je uložen odkaz na objekt zodpovědný za vznik jednotek.

Pod posledním atributem se nachází hlavní metoda celé třídy, *MainAttendant()* a je volána se dvěma parametry. Těmi jsou objekty reprezentující prostor, na který je možno jednotku umístit a přesouvaný objekt. Tělo metody je poté tvořeno čtyřmi „if“ bloky. První se kontroluje ve chvíli, když uživatel klikne na volný prostor, ve kterém se žádná jednotka nenachází. Současně ale musí platit, že žádnou jednotku nepřemisťoval. V tom případě dojde k ukončení celé metody. Obdobná situace nastane, když se uživatel pokusil přemístit jednotku do prostoru, ve kterém se již jedna jednotka nachází. Zajímavější je ovšem rozbor druhého a čtvrtého bloku. Druhý je aktivován v okamžik, když uživatel klikl do prostoru, ve kterém se jednotka nachází. Zároveň ale musí opět platit, že žádnou jednotku nepřemisťoval. V těle tohoto bloku pak dochází k inicializaci atributů *moving* a *steady*. Dále k ukončení útočné animace přesouvané jednotky, jestli doposud probíhala, a k ukončení přebíjení zbraně. Níže dojde k aktivování objektu *moving*, ukončení možnosti koupě jednotky, deaktivování objektu *steady* a přenastavení hodnot proměnných *isMoving* a *isReplacing*. Dále dojde k aktualizaci hodnot objektu reprezentující prostor. Poslední větev je pak aktivována ve chvíli, když uživatel klikne do volného prostoru, zatím co přesouval jednotku. V těle této větve se nejprve zkontroluje, zdali uživatel klikl do prostoru kompatibilního s danou jednotkou. Každá druh jednotky má totiž svůj vlastní prostor, do kterého není možné umístit jednotku odlišného druhu. Jestliže prostor s jednotkou kompatibilní není, dojde k ukončení celé metody. V opačném případě dojde k deaktivaci objektu *moving*, aktivaci nákupního menu, aktivaci objektu *steady*, aktualizaci hodnot objektu reprezentující prostor a k aktualizaci souřadnic přesunuté jednotky. Dále dojde k přesunutí indikátorů počtu zdraví a doby přebití jednotky. Poté je nastavena proměnné *resetReload*.

Pod metodou *MainAttendant()* se nachází další dvě metody, kterými jsou *Update()* a *LateUpdate()*. V první z nich je realizován samotný přesun jednotky. Tělo druhé metody pak ve velké míře slouží k resetování atributů po přesunu jednotky. V metodě *LateUpdate()* nejprve dochází ke kontrole proměnné *resetReload*. Jestliže je v ní uložena pozitivní pravdivostní hodnota, dojde k ukončení možnosti opakovaného přesunu jednotky, v případě, že byla daná jednotka typu tank, dále pak k resetování přebití přesunuté jednotky, k resetování atributů *isMoving*, *isReplacing*, *steady* a *moving*. V posledním řádku této podmínky pak dochází k resetování atributu *resetReload*. S touto třídou dále souvisí třídy *ReplaceSpace*

a *ReplaceObject*.

4.4.1 ReplaceSpace

Třída ReplaceSpace je aplikována na všechny objekty, na které je možno umístit obrannou jednotku. Ve videohře jsou těmito objekty myšleny tzv. zákopy. Tato třída disponuje soukromými atributy, kterými jsou: *defendType*; *defendTower*; *callLock* a *_buildManager*. Atribut *defendType* slouží k nastavení typu zákopu. Tedy k identifikaci typu jednotek, kterým bude umožněn úkryt. V atributu *defendTower* je poté uložena právě ta jednotka, která se v daném zákopu skrývá. Následuje pomocný atribut *callLock*, jehož hodnota se nastavuje v závislosti na situaci hry. Ve své podstatě tak uzamyká možnost přesouvání jednotek v okamžiku, kdy hráč pozastaví boj. Posledním atributem je pak *_buildManager*, v němž je uložen odkaz na objekt zodpovědný za vznik jednotek.

Pod zmíněnými atributy se pak nachází metoda *Start()* v níž dochází k inicializaci posledně zmíněné proměnné. Následuje řada pomocných metod, ze kterých stojí za zmínku *CallManager()* a *RealizeCall()*. První metoda je volána v situaci stisknutí tzv. zákupu. V těle metody pak dochází k zavolání metody *RealizeCall()*. Ta je volána pouze tehdy, když nedošlo k pozastavení hry. V metodě *RealizeCall()* jsou nejprve inicializovány pomocné proměnné *replaceSpace* a *replaceObject*. Ty slouží k identifikaci aktuálního objektu a objektu, který chce hráč přemístit. Dále je zkontrolováno, zda uživatel nestaví obrannou jednotku. Pokud ano, dojde k provedení pomocných operací a následnému ukončení metody. Jestliže uživatel nestavěl, dojde k pokusu získání pomocných informací o přesouvané jednotce. V případě selhání je metoda ukončena. Dále dojde ke kontrole proměnné *replaceObject*. Jestliže neobsahuje výchozí hodnotu, a je u ní povolen přesun, nastane zavolání metody *MainAttendant()* s parametry *replaceSpace* a *replaceObject*.

4.4.2 ReplaceObject

Třída *ReplaceObject* je aplikována na všechny objekty, které je možno přesouvat. Ve videohře jsou těmi objekty myšleny převážně obranné jednotky. Výjimku však tvoří kouřový granát a letecká podpora. Ty tuto třídu neobsahují. Na vrcholu třídy se nachází soukromé atributy, kterými jsou: *moving*, *place* a *disableObject*. V první proměnné je uložen odkaz na objekt, na který je třída aplikována. Samotná inicializace proměnné pak probíhá v metodě *Start()*, která je volána při prvním vykresleném snímku. V proměnné *place* je uložen odkaz na objekt zákopu, ve kterém se jednotka skrývá. V poslední proměnné je pak uložena informace o tom, zdali nebylo zakázáno přesouvání již přesunuté jednotky. To se týká převážně tanků, protože ty je možno přemístit pouze jednou. O tom se hráč dozví v průběhu kampaně. Samotným důvodem je pak pomalé zásobování čet pohonnými hmotami. Nejzajímavější metodou celé třídy je pak *GetWasBuild()*, která vrací informaci vypovídají o tom, zdali byla již daná jednotka postavena.

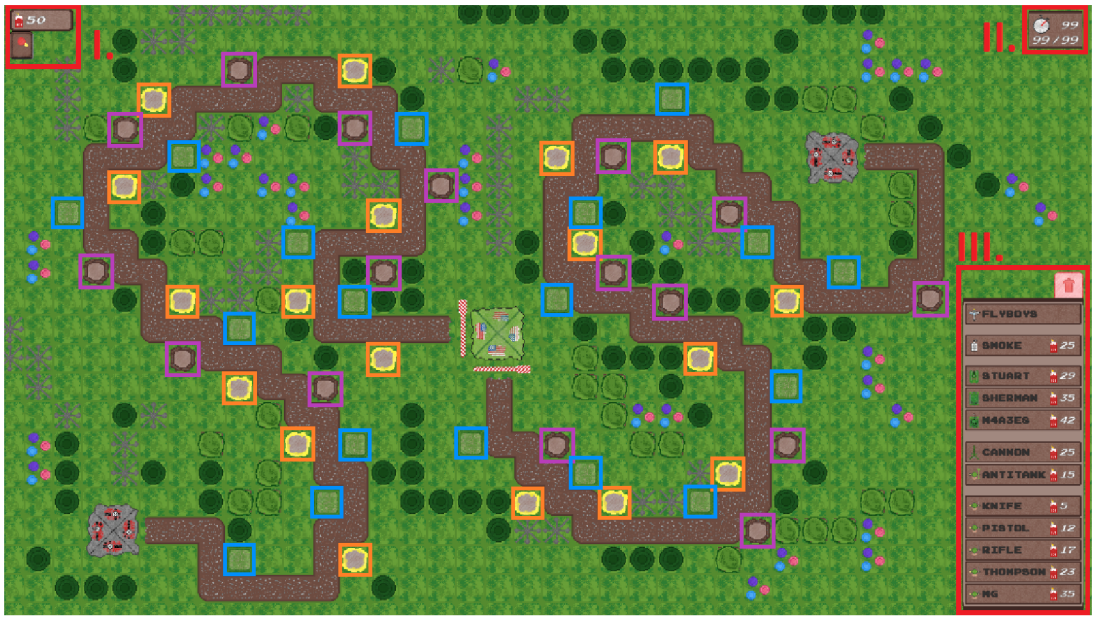
5 Hra z pohledu uživatele

Po spuštění videohry se hráč ocitá v hlavním menu. To je vybaveno čtyřmi tlačítky: *Play*, *Creative*, *Options* a *Quit*. Pomocí tlačítka *Play* se hráč přesune do nové scény, která slouží k volbě misí. Druhé z nich slouží k tvorbě uživatelsky definovaných úrovní, třetí k nastavení hudebního podkladu a čtvrté k ukončení celé videohry.

Zpočátku je odemknuta pouze první úroveň, přičemž k odemknutí nových je zapotřebí splnění všech předchozích. Po stisknutí tlačítka indikující úroveň je hráč obeznámen se základními informacemi o misi, jako jsou název, počasí, datum a bojeschopnost spojeneckých a nepřátelských jednotek. Potvrzením zeleného tlačítka *Battle!* nacházejícího se v pravém spodním rohu obrazovky je hráč přenesen do nové scény.

V ní vede před bitvou dialog se svým nadřízeným velitelem Ltg. Johnsonem. Od něj je hráč informován o detailech nadcházející mise. Při prvním průchodu je zapotřebí si vyslechnout celý dialog, přičemž po opakování mise je možné jej přeskočit stisknutím tlačítka *Skip dialogue*. Po dokončení dialogu je hráč přesunut do samotné bitvy. V ní se nachází aspekty, které je za potřebí rozebrat. Prvními jsou spojenecká a nepřátelská základna. Ty jsou propojeny trasou, po níž nepřátelské jednotky vedou svůj útok. Přiblížením se těchto vojsk až ke spojenecké bázi má za následek částečnou destrukci budovy, přičemž její úplné zničení vede k neúspěšnému ukončení úrovně. K její obraně jsou poblíž trasy umístěny zákopy, do kterých hráč umisťuje svoje obranné jednotky.

Herní prostor může být pro jednoduchost rozdělen do třech oblastí (obrázek 5). První oblast disponuje dvěma ikonami. Jedna ikona zobrazuje množství herní měny, které má hráč v danou chvíli k dispozici. Tu lze získat automatickým generováním či zneškodněním nepřátelských jednotek. Stisknutím druhé ikony se hráči zobrazí herní menu. V něm se muže vrátit do úvodního menu, kampaně nebo zpět do bitvy. Ve druhé oblasti je pak umístěn indikátor vln. Ten zobrazuje dobu, po jejímž uplynutí zahájí nepřátelské vojsko útok. Dále pak aktuální fázi útoku z jejího celkového počtu. Samotné nákupní menu se pak nachází ve třetí oblasti. To slouží ke koupi defensivních jednotek potřebných k obraně báze. Mezi ně patří pěchota, protitanková obrana, tanková brigáda a letectvo. Přičemž je každá jednotka speciální a záleží jen na zkušenostech hráče, aby přišel na její správné využití. Rovněž má každá jednotka svůj typ zákopu, do něhož je možno tuto jednotku umístit, a v pozdějších fázích i přemístit. Na následujícím obrázku (obrázek 5) jsou oranžovou barvou ohraničeny zákopy pro pěchotu, fialovou pak prostor pro protitankovou obranu a světle modrou barvou je ohraničena oblast určena pro tankovou brigádu.



Obrázek 5: Rozložení prostoru v úrovni

Po umístění jednotky do příslušného zákopu jsou nad ní zobrazeny podrobnější informace. Těmi jsou počet životů jednotky spolu s její přebíjecí dobou. Dále pak ikony šipky a odpadkového koše. Stisknutím šipky dochází k vylepšení jednotky. Samozřejmě za předpokladu, že má hráč k dispozici dostatečné množství herní měny. Cena poplatku vylepšení jednotky je umístěna vpravo od šipky. Stisknutím ikony odpadkového koše dochází k okamžitému zničení jednotky.

Po stisknutí tlačítka *Creative* nacházející se v hlavním menu se hráč přemístí do scény, v níž je mu umožněna tvorba vlastní úrovně. Zde jsou mu zobrazeny veškeré mapy, se kterými se setkal nebo v průběhu kampaně setká. K vytvoření požadované úrovně pak slouží tlačítko *Recreate* nacházející se v blízkosti mapy indikující danou úroveň.

Po jeho stisknutí je načtena scéna, ve které si tvůrce nastaví základní informace o dané úrovni. Tím je myšlen: *počet vln*; *množství herní měny* se kterým bude hráč začínat; *časové rozmezí*, po jehož vypršení dojde ke generování další herní měny a *počet životů báze*. Počet vln je omezen na patnáct, a to z důvodu grafického rozložení tlačítek nacházejících se v poslední scéně tohoto nastavení. Jestliže je hráč se svými rozhodnutími spokojen, stisknutím tlačítka umístěného v pravém dolním rohu, dojde k načtení nové scény. V ní se nastavují informace o jednotlivých jednotkách. Ty mohou být obecné, týkající se všech jednotek nebo specifické, vztahující se pouze k určitému druhu jednotky. Mezi obecné informace pak patří: *počet životů jednotky*; *poškození*, které způsobuje nepřátelům a *čas* potřebný k přebití útočného předmětu. Spojenecká jednotka obsahuje pouze jednu specifickou informaci, kterou je *cena* potřebná ke koupi dané defenzivní jednotky. U nepřátelských jednotek je situace velmi podobná. Specifickými jsou dvě informace: *odměna* za zničení a *poškození*, které jednotka způsobí, dostane-li se až ke spojenecké bázi. Samostatnou výjimku pak tvoří dva defenzivní prvky. Těmi jsou

kouřový granát a letecká podpora. U kouřového granátu je možné nastavit pouze pořizovací *cenu*. U letecké podpory pak *četnost* jejího zavolání. Ve chvíli, kdy je hráč spokojen s nastavením všech jednotek, přejde pomocí šipky, nacházející se opět v pravém dolním rohu, do poslední scény sloužící k vytváření útoku.

Ten se může skládat z jedné či více vln. U každé vlny je pak možné nastavit: *časovou prodlevu*, po jejímž uplynutí započne útok vlny a *rozestup jednotek*, který je rovněž definován pomocí času. Následně dochází ke specifikaci samotné útočné formace. Ta může obsahovat maximálně šestnáct jednotek. Na pravé straně obrazovky jsou poté umístěny specifické informace o nepřátelských jednotkách. Ty jsou zde umístěny staticky a není možné je žádným způsobem skrýt. Jestliže je uživatel s nastavením vln spokojen, stisknutím potvrzujícího tlačítka nacházejícího se v pravém dolním rohu dojde k uložení potřebných informací. Tím se daná úroveň stává hratelnou.

Následně je hráč přesunut do výchozí scény, ze které mu byl umožněn výběr uživatelsky definovaných úrovní. Vytvořenou úroveň může hráč spustit stisknutím tlačítka *Play*. Platí ovšem, že v danou chvíli může existovat nejvýše jedna vlastní úroveň daného bojiště. To jest, že vytvořením druhé úrovně stejného bojiště dochází k přepsání toho již existujícího. Mezi třemi zmíněnými scénami (sloužící k nastavení vlastní úrovně), lze přecházet prakticky libovolně. Hráč se při přecházení mezi scénami nemusí obávat ztráty již nastavených údajů. Ty jsou totiž při přechodu mezi scénami ukládány automaticky.

6 Grafická stánka

Tato kapitola je zaměřena na samotný vývoj grafických prvků, které je možné ve videohře spatřit. Na začátku vývoje je důležité zvážit, jaký grafický editor bude pro danou práci nejvhodnější. K dispozici se nabízí editory jako například GIMP [15], Adobe Photoshop [16] či PixilArt [17]. První dva jsou určeny pro náročnější úpravu multimediálního obsahu, zatímco PixilArt, jak již název napovídá, je převážně navržen pro tvorbu pixel-art modelů. GIMP a Adobe Photoshop je zapotřebí před samotným používáním nainstalovat, zatímco PixilArt je k dispozici v online podobě. Vzájemně se liší i v licenčních podmínkách. Zatímco je GIMP a PixilArt poskytován bezplatně, Adobe Photoshop poskytuje pouze 7denní bezplatnou verzi pro jednoho uživatele. Po jejím uplynutí je daný software zpoplatněný. V samotném odstavci jsou pak zmíněny informace, které je možné dohledat na stránkách editorů.

6.1 Volba editoru

Na základě již zmíněných licenčních podmínek programu Adobe Photoshop, které pro mne nebyly výhodné, jsem se rozhodl od zmíněného programu upustit. Výběr probíhal tedy mezi programy GIMP a PixilArt. Jelikož jsem potřeboval vytvářet modely i na zařízeních, na kterých mi byla odepřena instalace libovolného softwaru, rozhodl jsem se pro online editor PixilArt. Výhodou je, že s ním

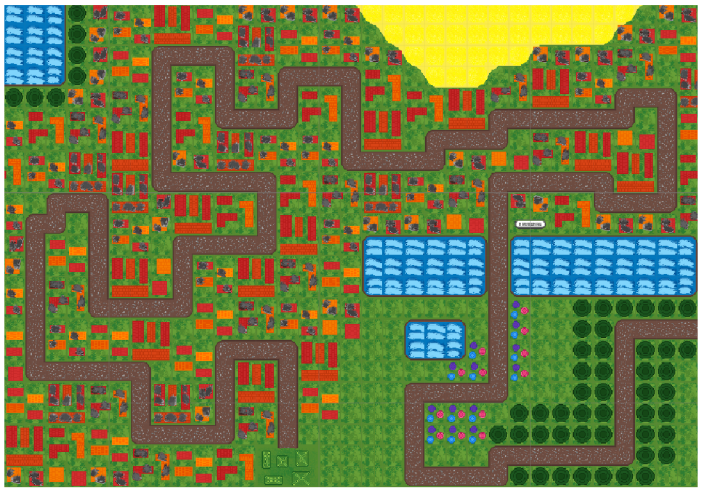
mám již určité zkušenosti. Samotný PixilArt je možné nalézt na webové stránce: https://www.pixilart.com/draw [18].

Editor je vhodný jak pro začátečníky, tak i pro ty středně pokročilé, a to z důvodu snadného a intuitivního používání. Při spuštění editoru zvolí uživatel pouze velikost kreslícího plátna v pixelech. Poté je umožněno bez jakýchkoliv problémů vytvářet dané modely. Součástí editoru je i krátký tutoriál, ve kterém se začínající uživatel dozví, jaké funkcionality editor poskytuje. Jelikož jsem doposud nenarazil, byť jen na sebemenší problém, vřele bych tento editor doporučil každému, kdo potřebuje vytvořit ve velmi krátkém časovém intervalu jakýkoliv 2D pixel-art model.

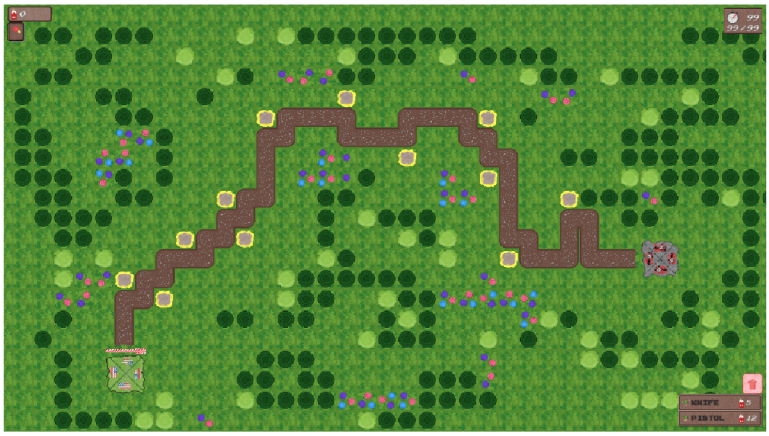
6.2 Tvorba úrovní a modelů

Po výběru aplikace, v níž bude grafická stránka mé práce vyvíjena, začal proces vytváření jednotlivých modelů. Zcela logicky mi přišlo nejvhodnější dále postupovat od jednodušších ke složitějším modelům. Právě proto mezi prvně vytvořené objekty patřily ty statické, mezi něž se řadí například tráva, cesta, květiny, stromy, křoví, ukazatele cesty a v poslední řadě jak nepřátelská, tak spojenecká báze. Když byly základní grafické prvky hotovy, začal návrh první úrovně.

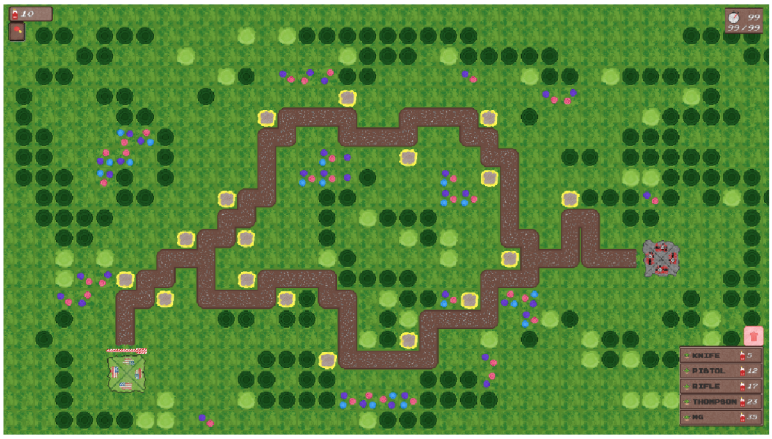
Jelikož bylo původním záměrem zrekonstruování bojišť, ve kterých v období druhé světové války sváděla boje 506. parašutistická 101. výsadková divize Spojených států Amerických, vyhledával jsem na internetu co nejvíce dobových fotografií. Mým cílem byla města jako Aachen, též někdy přezdíván jako „Malý Stalingrad“, Haguenau a Carentan. Dále pak oblasti jako lesy okolo Bastogne či tzv. Hürtgenský les, jež byl americkými vojáky přezdívaný jako „mlýnek na maso“. Každá úroveň měla nést jméno daného bojiště. Jelikož z pochopitelných důvodů nebylo možné dané fotografie dohledat, nebo je dohledat v dostatečném množství, nezbylo mi nic jiného než od původní myšlenky opustit a vytvořit úrovně, které přímou kopií bojišť nebudou. Na následujících obrázcích pak lze spatřit samotný vývoj jednotlivých úrovní.



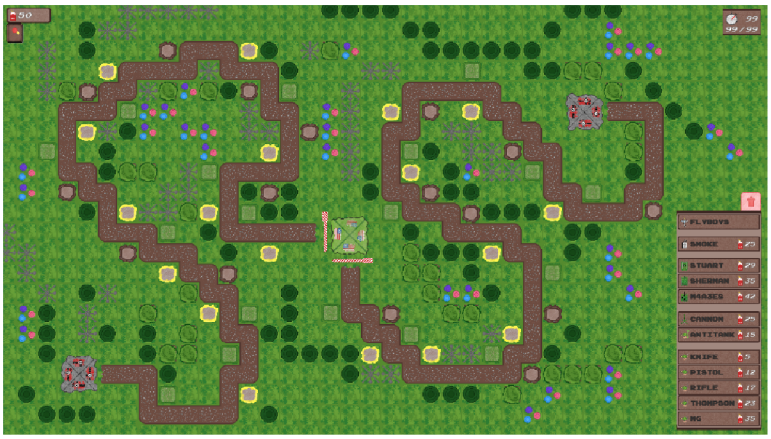
Obrázek 6: Úroveň, která nakonec posloužila pouze jako testovací



Obrázek 7: Úroveň, kde se nachází pouze jedna cesta, po které se mohou nepřátelé pohybovat



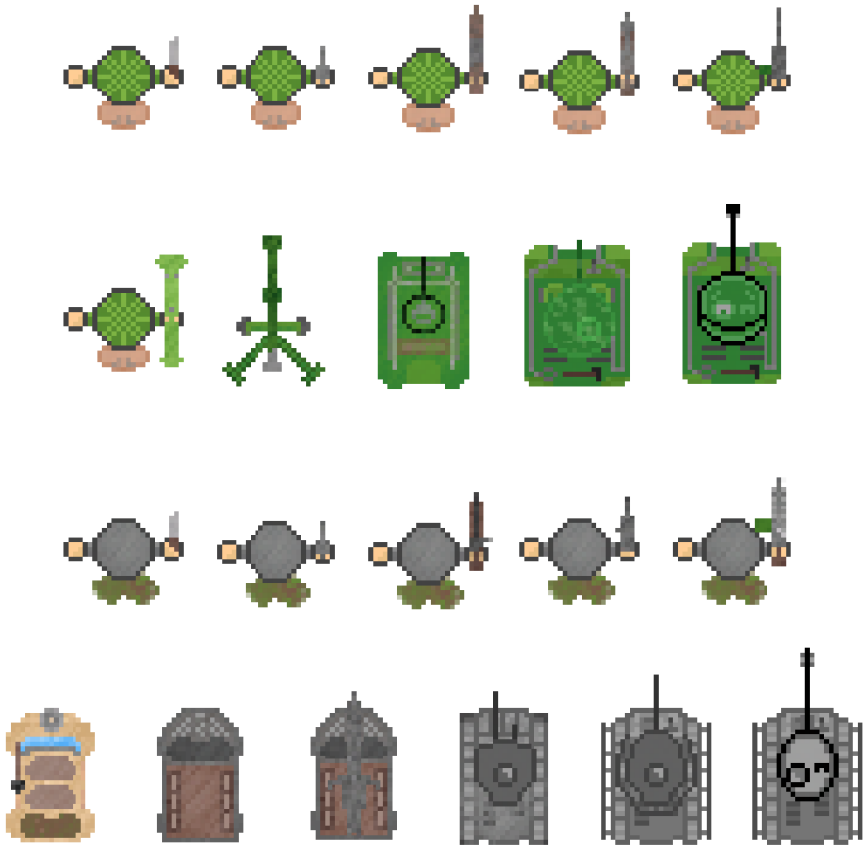
Obrázek 8: Rozdvojení trasy



Obrázek 9: V kleštích

Obrázek 7 zobrazuje typickou úroveň žánru tower defense. V podstatě se jedná pouze o trasu spojující nepřátelskou a spojeneckou základnu s možností umístění obranných prvků k ničení nepřátelských jednotek. Na obrázku 8 a 9 lze spatřit první originální aspekt, který jsem v žádné videohře tohoto žánru nezahlédl. Jedná se o možnost útoku nepřátelských vojsk z více směrů. Ačkoliv by se na první pohled mohlo zdát, že nedochází ke zkomplikování úrovně, opak je pravdou.

Po dokončení několika úrovní bylo důležité vytvořit jednotky, které se budou ve scénách vyskytovat. A to jak ty nepřátelské, tak spojenecké. S tímto úkolem byly spojeny dva hlavní problémy. Prvním z nich bylo, jakým způsobem odlišit jednotky od sebe, a druhým pak, jak rozlišit různé fáze stejné jednotky. Mnohem snazší bylo vyřešit první z nich. Nepřátelské jednotky budou zbarveny do šeda, zatímco ty spojenecké do světle zelena. Vymyšlení správného řešení druhého problému bylo náročnější. Fáze stejných jednotek se týkají jmenovitě pěchoty, kanónu a tanků. Rozlišení kanónů a tanků bylo snadné, neboť každý z těchto modelů je odlišný. Ke komplikacím dochází právě u pěchoty, jelikož je videohra laděna do pohledu z ptačí perspektivy, je helma primární objekt, který hráč spatří. Po několika konzultacích a ve finále nevhodných řešeních jsem se rozhodl rozlišit pěchotu pouze zbraní, kterou jsou vybavení, jak je možné spatřit na obrázku 10.



Obrázek 10: Všechny jednotky, které se zúčastní bojů

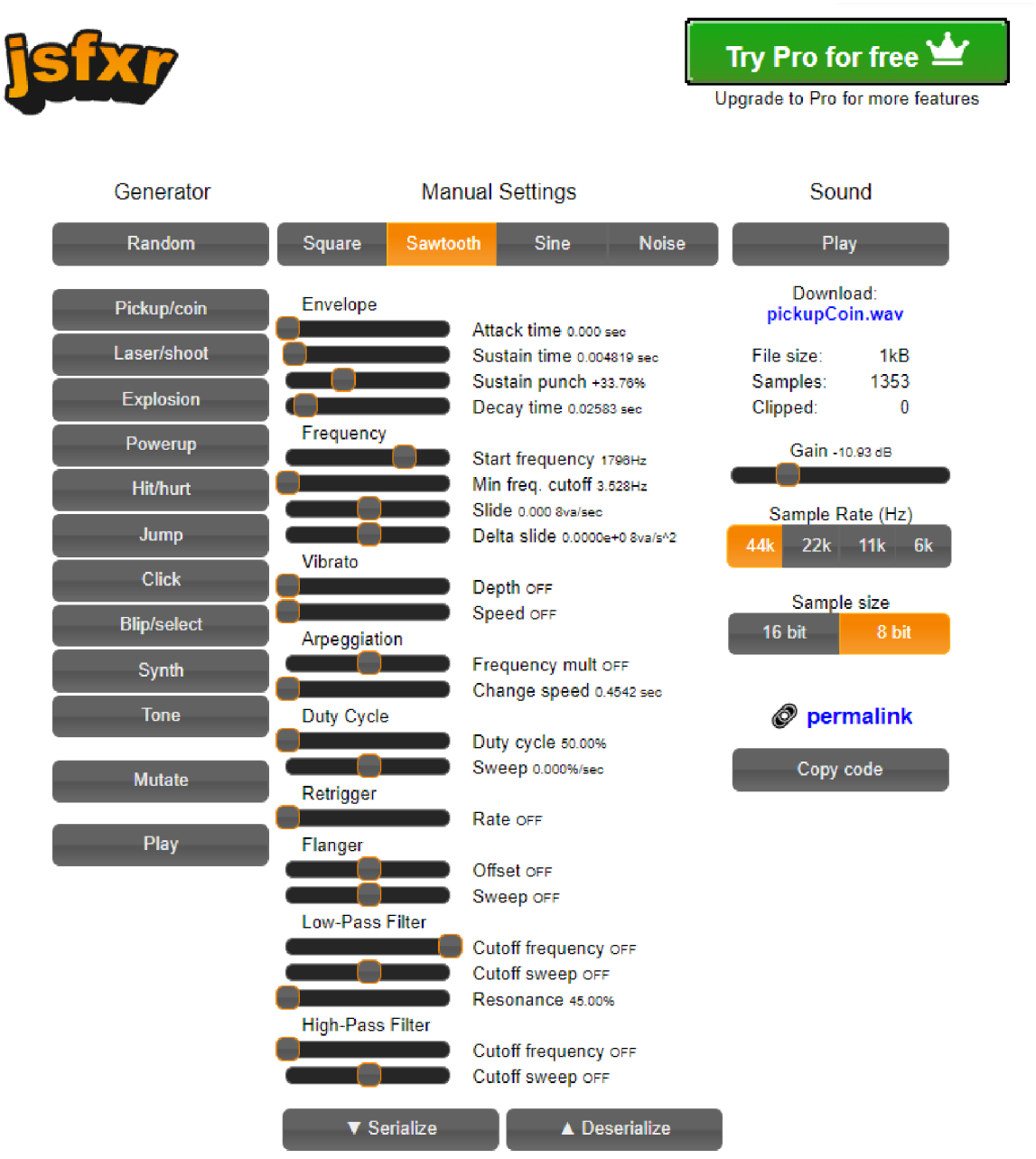
7 Zvukový doprovod

S každou hrou či videohrou je neodmyslitelně spjat i její hudební doprovod. V mém případě patřila jeho tvorba mezi ty nejnáročnější části celého vývoje videohry. I přes veškerou snahu se mi nepodařilo vytvořit hudební podkres odpovídající mým představám. Z toho důvodu jsem oslovil svého kolegu Jonáše Vyhnálka, který se skládání hudebních znělek věnuje. Samozřejmostí bylo, že veškerý zvukový doprovod podléhá jeho autorským právům, ale mě byl propůjčen pod podmínkou nekomerčního využití.

Zvukový podklad tvoří melodie *Way and Glory*, samotné bitvy pak doprovází *The Ides of March*. Jonáš Vyhnálek je také autorem zvuků palby útočných zbraní, jelikož se mi ani tento zvuk nepodařilo správně vykreslit v programu Bosca Ceoil [19], který jsem si pro svoji jednoduchost zvolil.

Avšak ve finální verzi videohry se nachází jeden mnou vytvořený zvuk, který je možné zaslechnout při stisknutí tlačítka či najetí myši na tlačítko. K němu jsem využil internetovou stránku https://sfxr.me [20]. Ta disponuje celou řadou nastavení, pomocí jichž lze docílit originální zvukové stopy. Součástí nastavení

je rovněž sada předpřipravených zvuků, kterých lze kdykoliv využít. Mezi ně patří například sebrání předmětu; střelby z laserové zbraně; exploze; vylepšení jednotky; výskoku či stisku tlačítka. Veškeré nastavení je poté patrno na obrázku 11.



Obrázek 11: Webová aplikace jsfxr

8 Řešení problémů

Během vývoje jakékoliv videohry se každý vývojář setká s obtížemi, které musí překonat. Zmíněnými obtížemi mohou být krom těch implementačních i ty, při níž dochází k hledání chyb ve zdrojovém kódu aplikace. K jejich odhalení je zapotřebí vysledovat, kdy a za jakých okolností aplikace končí v nekonzistentním stavu.

Pro jednoduchost může být tato problematika rozdělena do dvou částí. První vzniká vždy na základě posloupnosti kroků provedených uživatelem. Druhá se pak objevuje ojediněle či zřídka. Odhalení takového problému bývá zpravidla časově náročnější. Jelikož jsem se v průběhu vývoje setkal s prvním druhem chyby, bude tato kapitola zaměřena právě na jejich odhalení a opravy.

8.1 Pohyb nepřátelských jednotek

Prvním problémem, s nímž jsem se při vývoji hry setkal, bylo napsání jednoduchého skriptu, jehož účel spočíval v pohybu nepřátelských jednotek po předpřipravené trase. Řešení se mi podařilo nalézt na platformě YouTube, konkrétně na kanále Brackeys [21]. Náplní videa není pouhé zobrazení skriptu. Nejdříve je vysvětlen jeho cíl a způsob, jakým jej bude dosaženo a až poté autor přechází na samotnou ukázku zdrojového kódu. Stejným způsobem byla vyřešena mechanika dialogů [22], práce se soubory [23], či přechodu scén [24].

8.2 Odměna za zničení nepřítele

V pokročilejší fázi vývoje probíhalo testování samotné videohry mnou, jakožto vývojářem. Jelikož vím, jakým způsobem je má videohra naprogramovaná, bylo její uvedení do nekonzistentního stavu prakticky nemožné. A tak jsem se rozhodl využít testery, aby mi případné nedostatky pomohli odhalit. Jeden z nich nalezl následující chybu. Po zničení nepřátelské jednotky nedochází k přičtení herní měny zobrazující se v levém horním rohu obrazovky. Respektive dochází, ale zhruba se sekundovým zpožděním. Jelikož jsem nevěděl příčinu chyby, rozhodl jsem se znovu prostudovat skript, který je za danou funkcionalitu zodpovědný. Zjistil jsem, že při zničení jednotky nedochází k aktualizaci textového pole, které množství herní měny zobrazuje. Po jeho aktualizaci při zničení jednotky byla chyba odstraněna

8.3 Útoky jednotek stejné národnosti

Po dokončení skriptu zodpovědného za útoky mezi nepřátelskými jednotkami proběhlo jeho otestování v testovací scéně. Skript byl nejprve přidán na objekty představující jednotky. Poté nastalo sestavení celého projektu a spuštění testovací scény. Jenže nastala zvláštní situace. Útok jednotek byl chaotický, dokonce docházelo k bojům mezi jednotkami stejné strany. Zpočátku jsem si myslel, že došlo k jisté chybě během sestavování. A tak sestavení projektu proběhlo znovu,

bohužel se výsledek opakoval. Po rozebírání daného skriptu jsem zjistil, že jednotkám chybí identifikátor národnosti. Z tohoto důvodu dochází k útoku napříč všemi jednotkami. Přidáním identifikátoru následně došlo k opravení zmíněného problému.

8.4 Rozšiřující informace jednotky

Během útoku na spojeneckou základnu je důležité neopomenout zobrazit počet životů i dobu přebíjení jednotek. U nepřátelských jednotek bylo umístění těchto informací poměrně intuitivní, a to přímo nad ně. U spojeneckých jednotek byla situace komplikovanější. Je to z toho důvodu, že se u nich zobrazuje daleko více informací. Těmi jsou dvě tlačítka a cena potřebná k vylepšení jednotky. Pomocí prvního tlačítka dochází k vylepšení jednotky, ovšem za předpokladu, že má hráč k dispozici dostatečný obnos herní měny. Druhé pak slouží k odstranění jednotky jako takové. Bylo tedy důležité vymyslet, kam tyto informace umístit.

Nabízelo se hned několik možností řešení. První z nich spočívalo v zobrazení informací přímo nad danými jednotkami, tedy stejně tak jako u nepřátelských jednotek. Hlavní výhoda spočívá v tom, že hráč okamžitě vidí počet životů a přebíjecí dobu každé jednotky. Druhou, ze zmíněných možností, bylo zobrazení těchto informací až v okamžik, kdy na danou jednotku hráč najede myší. Toto řešení vypadá po vizuální stránce lépe než první, nicméně žádným způsobem neusnadňuje hratelnost. Třetí způsob spočíval ve vytvoření speciální tabulky, v níž by byly potřebné informace zobrazeny. To bylo pro mou práci nevhodné, a tak nebylo použito. Využití druhé varianty by vedlo hráče k neefektivní koordinaci obranných jednotek. Další nevýhodou bylo automatické nezobrazování přebíjecí doby jednotky. Od tohoto řešení bylo rovněž upuštěno. Rozhodnul jsem se nakonec pro první řešení, a to z důvodu převažujících pozitiv.

8.5 Menu

Nedílnou součástí jakékoliv videohry je menu, které hráči poskytuje základní funkcionality, kterými jsou ztlumení hudebního doprovodu či přesměrování do jiných úrovní. Rovněž má při jeho načtení v předpřipravené, i uživatelsky definované, úrovni dojít k pozastavení veškerých aktivit odehrávajících se ve scéně. Těmi jsou myšleny pohyb nepřátelských jednotek, jejich útok, animace útoku, automatické generování herní měny a tak dále.

Samotné pozastavení, a jejich následné spuštění, bylo prováděno pomocí dvou pomocných metod vyskytujících se v těle každého skriptu. Tento problém jsem konzultoval s kolegou Ondřejem Fremuthem, který v Unity již delší dobu programuje. Upozornil mne na to, že pozastavení veškerých aktivit v Unity může být provedeno pomocí pouze jednoho řádku. Navzdory správnému fungování aplikace mi přijde vhodné na daný fakt upozornit. Z jistého úhlu pohledu se totiž o chybu skutečně jedná, je to výrazné zkomplikování mechaniky, kterou lze zjednodušit.

Závěr

V této bakalářské práci jsem vyvinul a naimplementoval 2D videohru situovanou do období druhé světové války. V ní se hráč zhostil role velitele čety, jehož úkolem byla organizace spojeneckých jednotek do obranných pozic. Zpočátku hráč obdržel minimální počet jednotek, které postupem času narůstaly spolu s rozvíjejícím se příběhem. Mezi ně patřila pěchota, protitanková obrana, tanková brigáda a letectvo, přičemž každá jednotka plnila svou specifickou úlohu a byla určena pro odlišný styl boje. Kromě předpřipraveného příběhu se ve hře nachází i možnost tvorby uživatelsky definovaných úrovní. U těch není v tuto chvíli možné vytvořit unikátní bojiště, nicméně je možné nastavit zbylé aspekty úrovně, jako je počet nepřátelských vln, počáteční množství herní měny či poškození, které udělují jednotky.

I když jsem s dosavadním výsledkem projektu spokojen, rád bych na něm v budoucnu pracoval i nadále, neboť si myslím, že s přibývajícími zkušenostmi dokážu lépe vytěžit potenciál hry, a to například rozšířením kampaně o úrovně odehrávající se v poušti či v zimním období. Dále je tu pak možnost tvorby unikátních map, v nichž se uskuteční boje definované uživatelem. Po grafické stránce by mohla být poupravena či rozšířena některá bojiště.

Samotná videohra byla vytvořena v herním enginu Unity, přičemž veškeré zdrojové kódy byly napsány v programovacím jazyce C#. Ve videohře jsem zužitkoval své znalosti s enginem Unity spolu s vědomostmi jazyka C#, které jsem v průběhu studia nasbíral. Práce na projektu byla inspirativní a předala mi cenné rady a zkušenosti, jakým směrem by se mohly ubírat budoucí projekty, ať už podobného, či většího rozsahu.

Conclusions

In this thesis, I have developed a 2D video game situated in the World War II period. The player takes on the role of a commander whose goal is to organize allied units into defense positions. At the start of the game, the player manages only a small number of units, which increase as the storyline progresses. There are units with different skill sets, each with a different combat style. The units are as follows: anti-tank gun, tank brigade, and last but not least, air force. The player can also create levels where all the parameters except the map layout can be personalized (amount of enemy waves, amount of in-game currency and damage, which enemy and allied units deal).

Even though I'm satisfied with the result, I would like to continue with the development of this game because I feel like with increasing aptitude, I can refine the game further with new levels in desert or winter environments. There is also the possibility of creating unique maps customized by the player. From a graphical perspective, there is the possibility of upgrading or advancing further some battlefields.

The game itself was created in the Unity game engine. All the source code was written in C#. I have utilized all my knowledge of the C#programming language and Unity engine, which I have gained during the study of this problem. The development of this game was inspiring for me and gave me valuable knowledge and direction towards where I would personally like to steer my future projects, whether they are more challenging or simpler in scope.

A Obsah elektronických dat

`bin/`
: Adresář obsahující spustitelnou videohru

`doc/`
: Adresář obsahující dokumentaci bakalářské práce

`src/`
: Adresář obsahující projekt spustitelný v Unity engine

`README.txt`
: Textový soubor obsahující instrukce ke spuštění videohry i projektu v Unity engine

Literatura

[1] *Unreal Engine*. [online]. 2024 [cit. 2024-2-16]. Dostupný z: ⟨https://www.unrealengine.com/en-US/⟩.

[2] *25 Years Later: The History of Unreal and an Epic Dynasty*. [online]. 2024 [cit. 2024-4-20]. Dostupný z: ⟨https://www.pcmag.com/news/25-years-later-the-history-of-unreal-and-an-epic-dynasty⟩.

[3] *Programming with C++*. [online]. 2024 [cit. 2024-4-20]. Dostupný z: ⟨https://docs.unrealengine.com/4.26/en-US/ProgrammingAndScripting/ProgrammingWithCPP/⟩.

[4] *Blueprint Visual Scripting*. [online]. 2024 [cit. 2024-4-20]. Dostupný z: ⟨https://docs.unrealengine.com/4.26/en-US/ProgrammingAndScripting/Blueprints/⟩.

[5] *AAA Titul*. [online]. 2024 [cit. 2024-3-31]. Dostupný z: ⟨https://cs.wikipedia.org/wiki/AAA⟩.

[6] *Defold*. [online]. 2024 [cit. 2024-2-16]. Dostupný z: ⟨https://defold.com/⟩.

[7] *The programming language Lua*. [online]. 2024 [cit. 2024-4-20]. Dostupný z: ⟨https://www.lua.org/⟩.

[8] *Defold versioning*. [online]. 2024 [cit. 2024-4-20]. Dostupný z: ⟨https://forum.defold.com/t/defold-1-6-2-beta/74910⟩.

[9] *Unity*. [online]. 2024 [cit. 2024-2-16]. Dostupný z: ⟨https://unity.com/⟩.

[10] *Unity Long Term Support*. [online]. 2024 [cit. 2024-3-31]. Dostupný z: ⟨https://unity.com/releases/editor/qa/lts-releases?major_version=&minor_version=&version=&page=1⟩.

[11] *The history of Unity*. [online]. 2024 [cit. 2024-4-20]. Dostupný z: ⟨https://medium.com/@wota_mmorpg/unity-development-history-and-the-influence-of-this-game-engine-on-the-game-development-36dc7a7a3b9d⟩.

[12] *Unity Plug-ins*. [online]. 2024 [cit. 2024-4-20]. Dostupný z: ⟨https://docs.unity3d.com/2020.1/Documentation/Manual/Plugins.html⟩.

[13] *Fall Guys: Ultimate Knockout*. [online]. 2024 [cit. 2024-2-16]. Dostupný z: ⟨https://www.fallguys.com/en-US/⟩.

[14] *2D LookAt Method*. [online]. 2024 [cit. 2024-4-22]. Dostupný z: ⟨https://discussions.unity.com/t/lookat-2d-equivalent/88118/4⟩.

[15] *GIMP - GNU Image Manipulation Program*. [online]. 2024 [cit. 2024-4-7]. Dostupný z: ⟨https://www.gimp.org/⟩.

[16] *Adobe Photoshop*. [online]. 2024 [cit. 2024-2-16]. Dostupný z: ⟨https://www.adobe.com/cz/products/photoshop.html/⟩.

[17] *Pixilart – Free Online Art Community and Pixel Art Tool.* [online]. 2024 [cit. 2024-4-7]. Dostupný z: ⟨https://www.pixilart.com/⟩.

[18] *Pixilart – Free Online Art Community and Pixel Art Tool.* [online]. 2024 [cit. 2024-4-7]. Dostupný z: ⟨https://www.pixilart.com/draw/⟩.

[19] *Bosca Ceoil.* [online]. 2024 [cit. 2024-2-16]. Dostupný z: ⟨https://boscaceoil.net/⟩.

[20] *Jsfxr – 8 bit sound maker and online sfx generator.* [online]. 2024 [cit. 2024-2-16]. Dostupný z: ⟨https://sfxr.me/⟩.

[21] *Brackeys.* [online]. 2024 [cit. 2024-3-30]. Dostupný z: ⟨https://www.youtube.com/@Brackeys⟩.

[22] *How to make a Dialogue System in Unity.* [online]. 2024 [cit. 2024-3-30]. Dostupný z: ⟨https://www.youtube.com/watch?v=_nRzoTzeyxU⟩.

[23] *SAVE & LOAD SYSTEM in Unity.* [online]. 2024 [cit. 2024-3-30]. Dostupný z: ⟨https://www.youtube.com/watch?v=XOjd_qU2Ido⟩.

[24] *How to make AWESOME Scene Transitions in Unity!* [online]. 2024 [cit. 2024-3-30]. Dostupný z: ⟨https://www.youtube.com/watch?v=CE9VOZivb3I⟩.