Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Online multiplayer hra v Unity3D

2017

Tomáš Polák

Vedoucí práce:
Mgr. Martin Trnečka, Ph.D.

Studijní obor: Aplikovaná informatika,
prezenční forma

Bibliografické údaje

Autor:	Tomáš Polák
Název práce:	Online multiplayer hra v Unity3D
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2017
Studijní obor:	Aplikovaná informatika, prezenční forma
Vedoucí práce:	Mgr. Martin Trnečka, Ph.D.
Počet stran:	41
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Tomáš Polák
Title:	Online multiplayer game in Unity3D
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2017
Study field:	Applied Computer Science, full-time form
Supervisor:	Mgr. Martin Trnečka, Ph.D.
Page count:	41
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Práce pojednává o kompletním vytvoření akční hry pro více hráčů v herním enginu Unity3D. V práci jsou popsány využité technologie a postupy tvorby, od grafické stránky, přes zvuky ve hře, až po popsání jednotlivých naprogramovaných skriptů. Práce tak obsahuje uživatelskou i programátorskou dokumentaci.

Synopsis

This thesis describes complete creation process of an action multiplayer game in Unity3D engine. This means description of used technologies and work processes in range from graphical design, through in-game sounds, to details of individual programmed scripts. Thesis therefore contains whole user guide and programmer's documentation.

Klíčová slova: Unity3D; PhotonNetwork

Keywords: Unity3D; PhotonNetwork

Děkuji Martinu Trnečkovi za odborné vedení bakalářské práce.

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

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam zdrojových kódů

1 Úvod

Už zhruba od druhého ročníku střední školy jsem se ve svém volném čase zabýval tvorbou her, která mě nesmírně baví. Je to obor, ve kterém bych se chtěl dále rozvíjet, a proto tvorba hry, jakožto bakalářské práce, byla zcela jasná volba. Byla to pro mě příležitost dostat se této oblasti víc pod kůži a poprvé si vyzkoušet vytvořit větší projekt. Tolik k důvodu, proč jsem si toto téma vybral.

2 Použité technologie

2.1 Unity3D

Unity3D je herní engine, který byl zpočátku zaměřen na 3D hry pro Mac platformu, následně se ale velmi rychle rozvinul. Nyní lze za pomocí tohoto enginu tvořit hry jak 3D tak i 2D. V současné době můžete hru pomocí Unity vydat až 25 platforem kde nechybí ani v dnešní době velmi oblíbené a diskutované VR platformy nebo také platformy využívané v zařízeních Smart TV[1].

Unity je velmi komplexní vývojové prostředí. Ke tvoření her, a to na jakoukoliv z podporujících platforem, nepotřebujete jiný nástroj než samotné Unity3D prostředí.

S Unity se vám nainstaluje i vývojářské prostředí Mono Develop, které můžete využít pro psaní skriptů. Nic vám ale nebrání použít jakékoliv jiné vývojářské prostředí.

Unity obsahuje vlastní grafickou scénu, ve které můžete nastavovat objekty, zvuky, světla, pozadí a ostatní věci objevující se ve vaší hře. Je vám také k dispozici vývojářská konzole, do které si můžete nechat v průběhu testování vaší hry vypisovat různé hlášky ze skriptů a tím si jednoduše ověřit v jakém stavu se vaše hra nachází.

V Unity jsou veškeré výjimky odchytávány automaticky a jejich obsah je vám vypisován rovnou do konzole. Výjimky můžete také zachytávat klasickým způsobem pomocí kódu.

Pro testování hry není nutné vytvářet spustitelné soubory pro vaši platformu. Vaši hru si jednoduše spustíte přímo v Unity prostředí, kde si také můžete hru libovolně pozastavovat a následně zkoumat objekty ve scéně, měnit rozlišení obrazovky a libovolně upravovat nebo mazat vytvořené objekty.

Pro samotné vydání hry vám slouží okno, kde se vám po výběru platformy a nastavení vygeneruje potřebný obsah pro spuštění. Obecně to funguje velmi jednoduše, ale realita je trochu někde jinde.

Například než vydáte hru na platformu WebGL, musíte hru složitě optimalizovat, aby nezabírala moc paměti prohlížeče, ve kterém je pak spouštěná.

V tomto ohledu pak figuruje více aspektů, ale obecně pro většinu klasických platforem opravdu stačí jen kliknout a jednu hru si vydat třeba na tři různé platformy.

V Unity si můžete vybrat, jestli budete psát skripty v programovacím jazyku CSharp nebo Java Script, nebo je můžete klidně i střídat, avšak poté přichází problémy s kompatibilitou mezi jednotlivými skripty.

Já jsem si vybral jazyk CSharp, ve kterém programuji od střední školy, a tak je mi nejblíž. Většina uživatelů na fórech pak také doporučuje psát skripty v CSharp.

Dokumentace k enginu i prostředí samotnému je velmi dobrá a rozsáhlá. Velká výhoda je, že příkladné skripty, které naleznete v dokumentaci si můžete libovolně přepínat mezi CSharp a Java Scriptem.

Unity3D byl jako jeden z prvních enginů, který poskytoval většinu svých funkcí zcela zdarma. Proto už je na světě nespočet uživatelů, kteří s ním mají velké zkušenosti, a tak je komunita kolem Unity velmi rozsáhlá a není problém najít radu u někoho zkušenějšího.

2.2 PhotonEngine

Tento malý engine slouží výhradně pro potřeby networkingu. Jeho testovací používání je zdarma, což je obrovská výhoda – nemusíte platit za něco co se rozhodnete nepoužívat. Je stejně jako Unity3D multiplatformní, a tak jej můžete využívat jak v Unity, tak třeba i v Unreal Enginu 4, v Andorid aplikacích, Windows Store aplikacích, a tak dále [2].

Ve zdarma verzi pro Unity3D můžete hru vydat na všechny platformy mimo Android a IOS.

Obecný princip networkingu Photonu je jednoduchý. Klient je automaticky přihlášen k hlavnímu serveru, a poté se může připojit do jednotlivých lobby. Lobby jsou rozdělené primárně podle regionů (Evropa, USA, Čína a tak dále). Po připojení do lobby má klient přístup k jednotlivým pokojům. Pokoje samotné mají několik základních nastavení jako třeba maximální počet hráčů, heslo, jméno pokoje a další. První připojený klient do pokoje je automaticky označen jako Master Client. Master Client má oproti obyčejným klientům obecně více oprávnění. Může například mazat objekty patřící jinému klientovi a všechny důležité funkce by měl spouštět Master Client. Pokud se Master Client odpojí, tak je tato funkce automaticky předána jinému klientovi.

Abyste mohli Photon používat musíte si vytvořit účet na jejich portálu, kde si vygenerujete unikátní označení aplikace (UID), které potom vložíte do nastavení v prostředí Unity3D.

Až potom se vám hra bude připojovat na jejich master server a vy můžete začít obsluhovat jednotlivá lobby a pokoje. Když používáte zdarma licenci Photonu, tak je vám přidělen jeden ze slabších serverů, který můžete začít ihned používat.

Jak už jsem psal výše, tak na testovací účely tento server stačí, ale reálně je na něm stálá odezva kolem 80 – 120ms, a to je pro plynulé hraní nedostačující.

Pro ostrý provoz hry vám stačí dokoupit licenci nebo jako hlavní server použít svůj vlastní.

2.3 Grafická stránka hry

Všechny obrázky a animace, které jsou obsažené v mé hře byly vytvořeny přímo pro účely mé bakalářské práce mým kolegou. Žádný není ani koupený ani stažený z internetu. Obrázky jsou ručně kreslené grafickým tabletem v programu Adobe Flash Professional CC.

Každý snímek z každé animace je kreslen zvlášť, tím jsme se snažili docílit ne moc uhlazeného vzhledu. Zvolili jsme tento tzv. cartoon styl. Je trochu jednodušší

na návrh – grafik se nemusí zabývat realistickými detaily a celá hra pak vypadá velmi živá a dokáže hráče zaujmout.

Tímto bych mu ještě jednou rád poděkoval, protože práce, kterou vytvořil, je opravdu výborná a celkový vzhled bude určitě velkým lákadlem pro potencionální hráče.

Jak jsem psal výše tak jednotlivé snímky animací jsou kresleny zvlášť. Tímto stylem mi byly dodávány a já je pak v prostředí Unity3D, v nástroji, který je k tomu uzpůsoben, kompletoval a vytvářel z nich funkční animace.

3 Návrh řešení

V první fázi vývoje jsem žádný návrh nezpracoval a celý základ hry jsem naprogramoval postupně tak jak mě to napadlo. První verzi, kterou jsem zhotovil, byla verze pro jednoho hráče, kde jsem se ještě nezabýval networkingem.

Naprogramoval jsem pohyb hráče, střelbu, sbíraní nových zbraní a tak dále. Potom jsem začal tuto hru předělávat pro hru s více hráči. Ze začátku, když jsem neměl implementované animace, byl takový postup úspěšný. Později, když začalo věcí a funkcí přibývat jsem zjistil, že se mi hra bortí a další postup tímto stylem nebyl možný.

Začal jsem tedy znovu a jako první věc jsem navrhl řešení networkingu. Zhruba jsem si napsal například jak se budou obsluhovat projektily a smrti, jak bude řešeno skóre hráčů, jak se budou připojovat do pokojů, jak bude řešen čas kola a další.

Když jsem si naprogramoval tento networkingový základ a začal hru „skládat“ rovnou pro více hráčů, postup byl rychlejší a celý projekt stabilnější.

4 O hře

4.1 Žánr hry

Žánrově by se dala tato hra popsat jako akční střílečka pro více hráčů. V jednoduchosti jde totiž jen o zabíjení nepřátel, kteří jsou ovládaní jiným hráčem, pro získáni nejvíce bodů a tím si zajistit výhru kola.

4.2 Koncept hry

Hlavní koncept hry je inspirován starší hrou od českých vývojářů ze studia Sleep-Team. Hra se jmenuje Bulánci, a jako děti jsme ji znali a hrávali všichni. Bohužel, hra byla vydána v době, kdy ještě nefungovaly sociální sítě, a tak byl marketing o dost těžší, než je dnes a tím pádem se hra nedostala do širšího okruhu, třeba do zahraničí.

Chtěl jsem tak tuto starou hru oživit a dát ji zpět slávu, kterou v době vydání měla mezi námi.

Samotná hra je velmi jednoduchá. Základem jsou čtyři hráči. Hraje se každý proti každému. Na začátku kola se hráči náhodně objeví na mapě, neboli na scéně, se základní zbraní v ruce. Tato základní zbraň má pět výstřelů a po použití všech pěti se nějakou dobu nabíjí. Na scéně se postupně objevují náhodně zvolené speciální zbraně. Jsou čtyři druhy speciálních zbraní a každá má odlišnou funkci a odlišný počet projektilů, které jsou na jedno použití vystřeleny. V případě, že hráč použije všechny výstřely u speciální zbraně tak jí ztrácí a do rukou se mu vrací základní zbraň.

Každý hráč má jen jeden život, to znamená, že stačí jeden zásah od jakékoliv zbraně a hráč je na pár sekund vyřazen ze hry, kam se pak znovu vrací opět se základní zbraní v ruce.

Hra není omezena počtem smrtí nebo počtem zabití, ale hraje se dokud nevyprší časový limit. Po vypršení časového limitu hra končí a vybere se hráč, který měl na konci kola nejvíce bodů.

Bodování ve hře je jednoduché. Každý hráč začíná s nulou, pokud někoho trefí, jeden bod se mu přičte a trefenému hráči se jeden bod odečte. Body mohou jít i do záporných hodnot.

4.3 Téma hry

Tato hra, kterou jsem pojmenoval Wizards Of Azear je zasazena do fiktivního světa malých kouzelníků. Ve hře tak každý hráč hraje za jednoho z kouzelníků, kteří jsou odlišováni barvami.

Bojují prozatím na jedné mapě, která by měla znázorňovat posvátnou půdu, kde se konávaly rituály. Jako zbraně, se hráčům na mapě objevují různá kouzla.

Originální hra Bulánci je zasazena do fiktivního světa, kde proti sobě bojují polštáře.

V tomto směru jsem se tématem hry nechtěl podobat originálu, a tak jsem přišel s tématem malých kouzelníků hlavně z důvodu jednoduchosti kreseb a nepřeberného množství speciálních zbraní, tedy kouzel.

Veškeré uživatelské rozhraní je graficky zasazeno do tohoto magického tématu.

5 Uživatelské rozhraní

Celé uživatelské rozhraní je rozděleno do tří obrazovek. *Hlavní menu*, *Lobby Menu* a *Upcoming Stuff* obrazovka.

Graficky jsou tyto obrazovky znázorněny magickým svitkem, na kterém jsou všechny komponenty (tlačítka, nadpisy a tak dále) jen ve formě textů. V samotné hře, kde hráči proti sobě bojují, je uživatelské rozhraní odlišné.

Všechny texty, které tato hra obsahuje jsou napsány fontem *WoA* což je font, který je vytvořen speciálně pro tuto hru.

5.1 Hlavní menu

První obrazovkou je *Hlavní menu*. Tato obrazovka se uživateli zobrazí hned po spuštění hry. Ve vrchní části se nachází jen název hry, ve střední části pak jsou nejdůležitější tlačítka.

Tlačítko *Play* uživatele přesune do obrazovky *Lobby menu*. Následující dvě tlačítka jsou prozatím vypnutá, protože nejsou důležitá pro chod hry v současném stavu a zatím nemají žádnou funkci.

Poslední tlačítko uživatele přesune na obrazovku *Upcoming Stuff*. Ve spodní části je pak tlačítko, které hráče přesune na Facebookovou stránku hry, nyní je však také vypnuté.

Obrázek 1: Obrazovka Hlavní Menu

5.2 Lobby menu

Druhá obrazovka, na kterou se uživatel dostane po kliknutí na tlačítko *Play* v *Hlavním menu* nebo pokud opustí rozehranou hru. Tato obrazovka už je složitější a nachází se v ní poměrně dost komponent.

Od začátku jsem se snažil, aby celé uživatelské rozhraní bylo pro uživatele přehledné a nepotřeboval tak žádný návod k ovládání hry. To znamená i co nejméně komponent na jedné obrazovce, aby některou komponentu uživatel nepřehlédl.

Na druhou stranu bylo mým cílem, aby se uživatel co nejrychleji dostal do samotné hry, aby ho nezdržovalo přepínání více obrazovek pro tvorbu nebo připojení do hry. A tak jsem chtěl, aby bylo možné vytvořit novou hru a připojit do existující hry na jedné obrazovce. Vytvořil jsem tedy obrazovku, na které se nachází seznam existujících her a také možnosti pro vytvoření nové hry. Ve vrchní části, stejně jako u *Hlavního menu*, se nachází název hry.

Střední část je pak rozdělena do více částí. Ve vrchní části jsou 3 textová pole.

První textové pole slouží pro zvolení přezdívky, která se bude ukazovat ostatním hráčům ve hře. Další dvě jsou prozatím vypnutá – ty budou sloužit pro vyhledávání a zaheslovaní her. V levé částí se nacházejí tlačítka, kterými si uživatel vybírá do jakého regionu se připojí a kde bude pak následně hrát s ostatními hráči.

Ve střední části se nachází okno, ve kterém se zobrazuje seznam vytvořených pokojů ve vybraném regionu. Jedna položka z tohoto seznamu obsahuje název pokoje, počet připojených hráčů a tlačítko pro připojení do hry. Ve spodní částí se nachází tlačítko pro návrat do hlavního menu a tlačítko pro vytvoření nové hry.

5.3 Upcoming Stuff

Poslední obrazovka slouží jako informační panel, kde se nacházejí věci, které se připravují a budou v budoucnu ve hře. Na této obrazovce se nachází jen texty a samozřejmě tlačítko pro návrat do hlavního menu. V horní části je opět název hry.

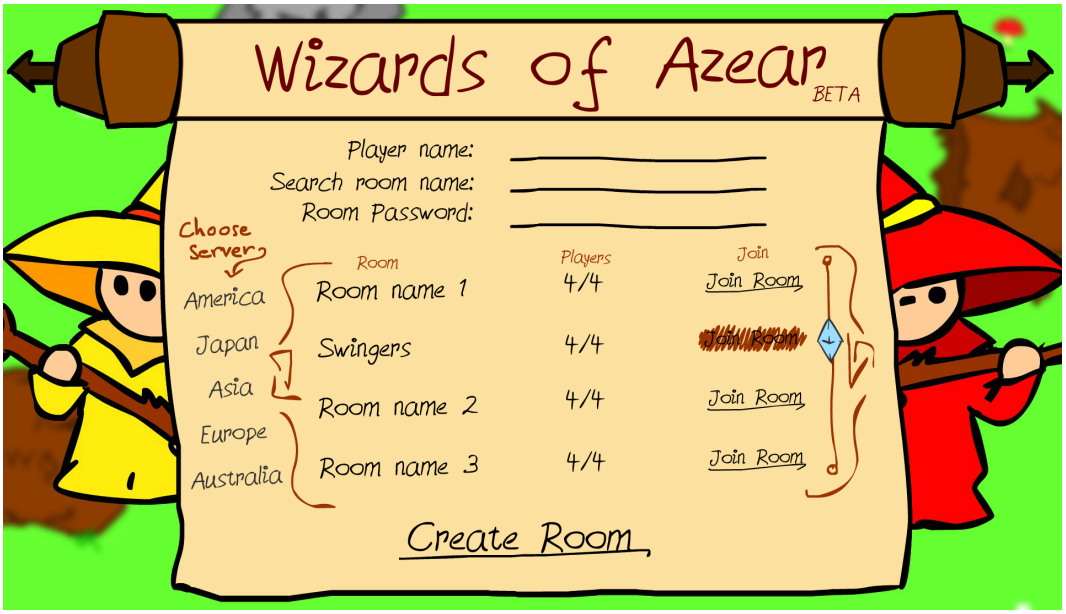
5.4 Uživatelské rozhraní ve hře

Když se uživatel připojí do hry první co vidí je mapa.

Ve spodní části obrazovky je malý panel, na kterém hráč vidí barvu svého kouzelníka, zrovna vybranou zbraň a počet střel, které ještě může použít. Vpravo od tohoto panelu se nacházejí dvě malá tlačítka. Jedno slouží pro vypnutí zvukových efektů a druhé pro vypnutí hudby.

Když hráč ve hře stiskne klávesu `TAB` zobrazí se mu malý panel, na kterém se nachází seznamu hráčů, jejich skóre a čas do konce kola.

Když hráč stiskne klávesu `ESC` zobrazí se mu stejný panel, na kterém se nachází text a dvě tlačítka. Po stisku prvního tlačítka se hráč odpojí a přesune



Obrázek 2: Obrazovka Lobby Menu

se do lobby. Po stisku druhého tlačítka hráč zůstane ve hře. V budoucnu se bude v tomto panelu nacházet i základní nastavení hry.

Po skončení kola se všem hráčům zobrazí panel s obrázkem a textem, kde se hráč dozví konečné umístění hráčů a jejich skóre. Ve spodní části se hráči zobrazí informace za jak dlouho začne další kolo.

Obrázek 3: Ukázka obrazovky na konci kola

6 Zvuky ve hře

Ve hře se nachází řada zvukových efektů a pár hudebních skladeb pro dokreslení atmosféry hry. Většina zvukových efektů a všechny hudební skladby jsou volně šiřitelné. Některé zvukové efekty jako třeba zvuk po kliknutí na tlačítko v menu nebo chůze kouzelníků po trávě jsem nahrál a upravil sám. Do budoucna plánuji mít veškeré zvuky ve hře vlastní.

Engine Unity3D disponuje řadou nástrojů pro jednoduchou implementaci a práci se zvuky. V základu stačí na objekt, který bude zvuk vydávat, vložit komponentu `Audio Source`. V této komponentě můžete nastavit zvuk, který se začne přehrávat hned po vytvoření objektu. Můžete nastavovat prioritu zvuku, rádius zvuku, hlasitost a rychlost. Můžete přepínat mezi stero a mono anebo také nastavovat 3D vlastnosti zvuku.

V případě mé hry může každý hráč současně vydávat dva zvuky. Abych toho docílil nestačí mi mít jen jeden `Audio Source`. Každý `Audio Source` totiž zvládne vydávat jen jeden zvuk současně. Proto mám vytvořený prázdný objekt, zařazený jako pod-objekt hráče, který má svůj vlastní `Audio Source`. Samotný objekt hráče tak vydává zvuk kroků a pod-objekt vydává zvuk střelby.

Naproti tomu existuje komponenta `Audio Listener`. Tato komponenta slouží k zachytávání zvuků ze scény. `Audio Listener` nelze nijak nastavovat nebo upravovat. Stačí ji tedy jen vložit na objekt a všechny zvuky v dosahu jsou zachytávány. V této hře je jen jedna statická scéna a jedna statická kamera, která zachycuje celou scénu. To znamená, že všechny zvuky ve scéně budou slyšet vždy a všude, a tak mi stačí mít `Audio Listener` jen na této kameře.

7 Animace objektů

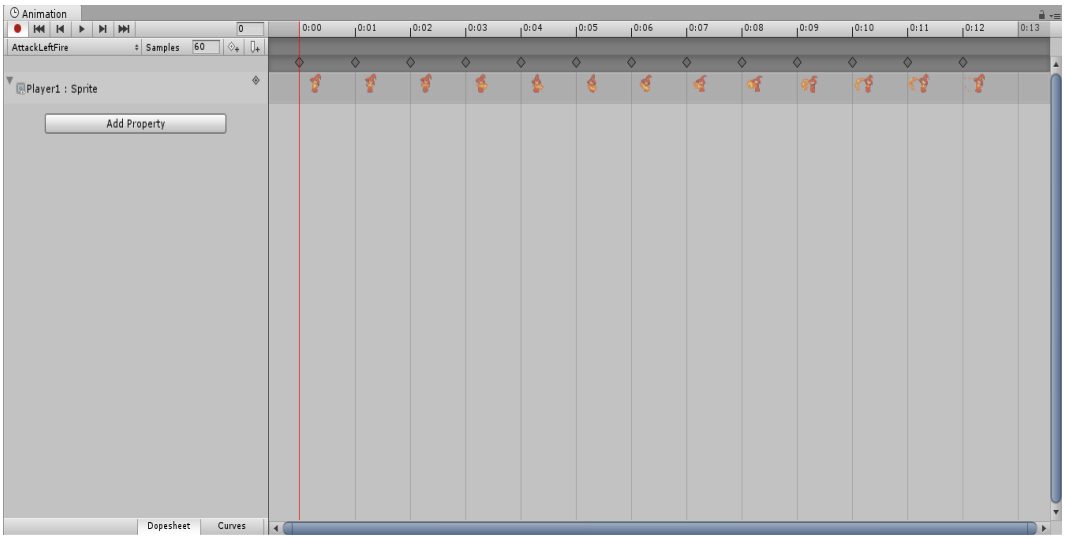
Ve hře se nachází poměrně velké množství animací. Největší počty animací se nachází u objektů hráčů (kouzelníků).

Hra v současném stavu obsahuje čtyři barvy kouzelníků. Každá barva má své vlastní animace. Pro každou barvu tedy existují vždy 4×5 (4 směry a 5 zbraní) animací pro každou akci - animace pro stání na místě, střelbu a dále smrti po zásahu jednotlivými zbraněmi. Další z animací, které hra obsahuje, jsou animace projektilů jednotlivých zbraní, a rozbití projektilů.

Pro tvorbu animací používám nástroj od Unity3D. Obecně tak stačí vybrat objekt, který chcete animovat a otevřít si nástroj `Animation Window`. Po spuštění toho nástroje budete vidět buď seznam již existujících animací na vybraném objektu, nebo tlačítko pro vytvoření nové animace.

7.1 Animation Window

`Animation Window` obsahuje časovou osu kam můžete vkládat a upravovat klíčové snímky. Dále máte možnost výběru různých akcí, které vám vytvoří nový klíčoví snímek. Jedinou z akcí vytváření klíčových snímku, kterou jsem použil při tvorbě hry, byla vložit klíčový snímek pomocí obrázku. Také máte možnost vkládat klíčové snímky otočením objektu, změnou velikosti, přesunutím objektu a změnou nastavení fyziky. Také můžete měnit veřejné proměnné, které obsahují skripty na právě animovaném objektu. Tohle má dost velkou výhodu pokud chcete proměnnou zaznamenat kdy skončila nějaká animace. Stačí jako poslední klíčový snímek vložit změnu stavu proměnné v určitém skriptu a další chování zpracovat až v kódu. Pro zjišťování v jakém stavu se animace nachází pak existuje více možností. Tohle je jeden z příkladů.



Obrázek 4: Animation window

Ve chvíli, kdy vytvoříte první animaci a uložíte ji, vloží se vám do objektu nová komponenta `Animator`, tato komponenta je v rámci animací nejdůležitější.

7.2 Animator Controller

S komponentou `Animator` se vytvoří objekt `Animator Controller`, který pak sebou nese všechny animace, které jste pro animovaný objekt vytvořili a veškeré nastavení.

Jedno z možných nastavení komponenty `Animator` je možnost přiřazení objektu `Animator Controller`. Vy můžete objektu hráče přiřadit `Animator Controller` třeba projektilu nějaké zbraně, a od té chvíle se budou na objektu hráče zobrazovat animace projektilu. Já tuto možnost využil pro změnu vzhledu jednotlivých kouzelníků, a to v případě, že hráč vezme jinou zbraň.

Příklad může být takový. Červený kouzelník se pohybuje po scéně se základní zbraní. Má tak v komponentě `Animator` přiřazený `Animator Controller` pro animace se základní zbraní. Když pak na scéně vezme jinou zbraň, právě v komponentě `Animator` se změní `Animator Controller`, který obsahuje animace červeného kouzelníka, držícího jinou zbraň.

Nejdůležitější částí komponenty `Animator` je tzv. relační schéma animací. Když vytvoříte přes `Animation Window` několik animací na jednom objektu, tak při otevření okna `Animator` uvidíte plochu se stavy. Mimo základní stavy jako `Vstup`, `Konec` nebo `Jakýkoliv stav`, se na ploše nachází stavy znázorňující jednotlivé animace. Náhodná animace se automaticky označí jako výchozí a tím pádem se po spuštění hry a vytvoření objektu začne přehrávat právě tato animace.

Vy tak v tomto schématu určujete relace mezi animacemi abyste mohli kontrolovat jak se budou animace střídat, jak poběží za sebou a tak dále.

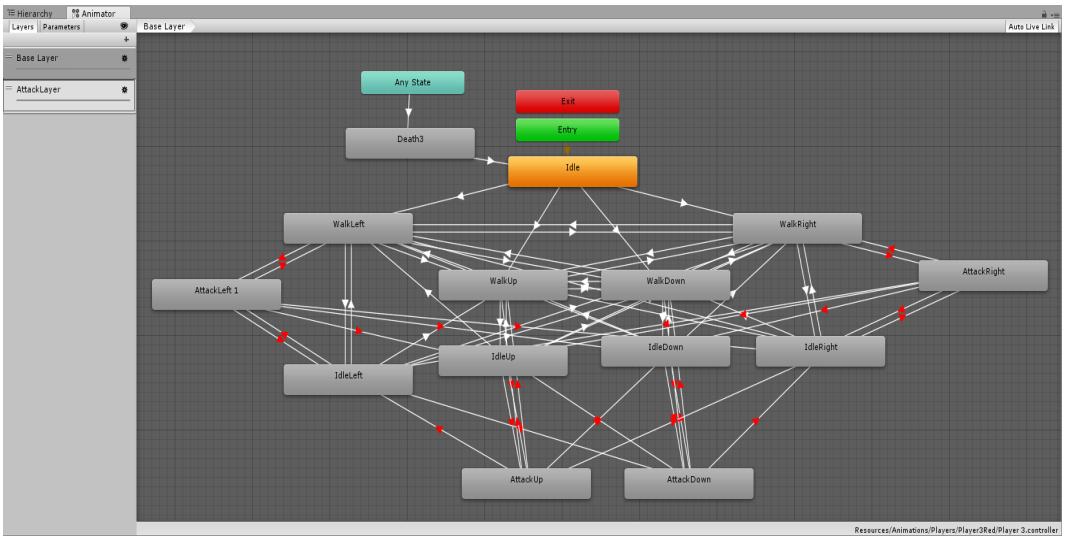
Mezi dvěma animacemi tedy vytvoříte relaci kde nastavíte, za jakých podmínek se z první animace spustí druhá. Přitom máte možnost si vytvářet různé parametry typu `float`, `int`, `bool` nebo `trigger`, kterými můžete přechod mezi animacemi (stavy) podmiňovat.

Příklad je následující. Na objektu hráče právě probíhá animace chůze. Z tohoto stavu je pak relace vedoucí do stavu animace střelby. Mám vytvořený parametr bool a nastavil jsem relaci tak, že pokud tento parametr bude pravda animace chůze se zastaví a spustí se animace střelby. Z toho stavu pak zase musí vést relace do jiných stavů s nastavením, že pokud parametr bude nepravdivý střelba se ukončí a bude pokračovat jiná animace, například zase chůze. Pak už si stačí ve skriptu, který se stará o animace, vybrat hráčovou komponentu `Animator` a v ní nastavit tento parametr na pravdu vždy když střílí a na nepravdu když se nestřílí.

Velmi zajímavá a užitečná věc je vytváření více vrstev tohoto relačního schématu. Jednotlivé vrstvy mají své vlastní parametry a vlastní nastavení relací. U každé vrstvy pak můžete nastavovat i její váhu a tím upravovat, která vrstva má větší prioritu. Užitečné je to proto, že vám dovolí být v každé vrstvě v jiném

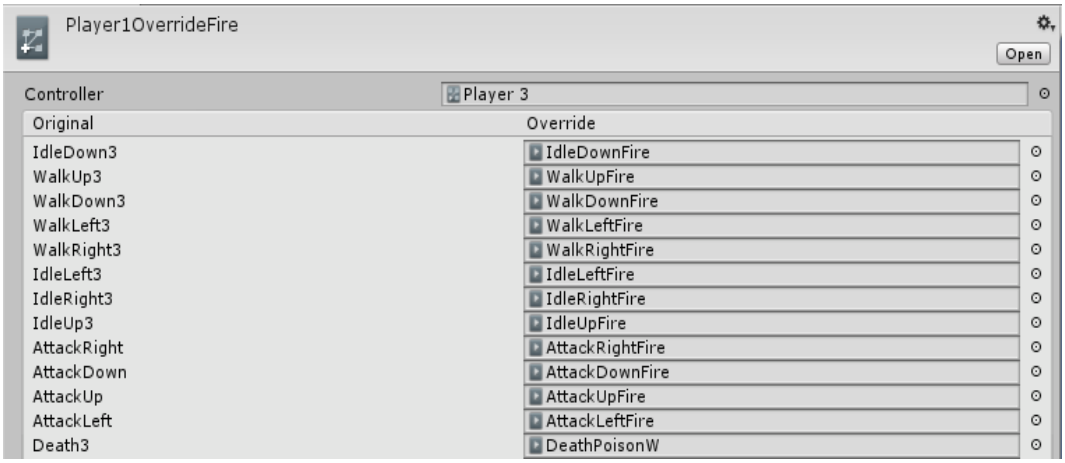
stavu. Tohle jsem použil při překrývání animací.

Základní logikou mé hry je, že se může zároveň chodit i střílet, což znamená, že se musí současně pustit animace chůze i střelby. Právě k tomu slouží vrstvy. V hlavní vrstvě mám animace chůze a ve vedlejší vrstvě animace střelby. Když pak hráč stiskne tlačítko pro střelbu, upraví se parametr ve vedlejší vrstvě, tím pádem se neovlivní podmínky relací v hlavní vrstvě a obě animace poběží zároveň.



Obrázek 5: Relační schéma animací hráče

Tohle relační schéma i na poměrně malé hře, jako je tato, může být dost složité. A vytvářet jej znovu pro každou barvu hráče a pro každou zbraň, by bylo velice neefektivní. Proto v Unity3D existuje objekt s názvem `Animator Override Controller`. Tento objekt má argument očekávající `Animator Controller`. Když mu nějaký přiřadíte zobrazí se vám seznam stavů, které originální `Animator Controller` obsahuje a každý z nich můžete jednoduše nahradit jiným stavem neboli jinou animací. Zůstane vám pak nastavení relací jako u originálu a samotný objekt se tváří, že je normální `Animator Controller`. Pak už stačí mít vytvořené jedno schéma relací, animace pro každou barvu zvlášť, a přes tento šikovný objekt tyto animace měnit.



Obrázek 6: Animator Override Controller

8 Ovládání hry

Ve všech obrazovkách, které se ve hře nacházejí, se veškeré uživatelské rozhraní ovládá pomocí kurzoru neboli myši.

Ve hře pak hráč ovládá svého kouzelníka pomocí šipek. Střelbu hráč vyvolá stisknutím mezerníku, panel se skóre hráčů pomocí tabulátoru, a nabídku pomocí klávesy *escape*. Hráč se může pohybovat jen do čtyř základních směrů (nahoru, dolů, doleva, doprava).

9 Networking

9.1 Připojení do hry

Jak už bylo psáno výše, pro networking hra používá Photon Engine, který funguje na principu regionů a následně pak pokojů. Když se uživatel dostane na obrazovku *Lobby Menu* má možnost vybrat si jeden z nabízených regionů.

Po kliknutí na region se klient připojí k master serveru a automaticky do lobby daného regionu. V tu chvíli se mu zobrazí seznam vytvořených pokojů (pokud nějaké existují), nebo obrázek s hláškou, že zatím žádná hra neexistuje. Lobby se automaticky aktualizuje každých pět sekund, aby měl hráč správné informace o pokojích.

Hráč má možnost napsat svou požadovanou přezdívku, která se bude zobrazovat ve hře ostatním hráčům. Pokud pole pro přezdívku nechá nevyplněné, vygeneruje se mu jméno automaticky.

Pokud se hráč chce připojit k existující hře, jednoduše klikne na tlačítko *Join* u konkrétního pokoje. V tu chvíli se mu v závislosti na počtu hráčů v pokoji přiřadí ID (0-3) a podle ID se mu následně přiřadí barva kouzelníka. Hned poté je připojen do hry a je mu vytvořen samotný objekt kouzelníka.

Pokud se hráč v Lobby rozhodne vytvořit nový pokoj a klikne tak na tlačítko *Create Room*, vytvoří se nový pokoj s názvem, který je konkatenací hráčova jména a náhodného čtyřčíslí. Hráčovi je přiděleno ID 0 a do vlastnosti pokoje přidán časový záznam, který značí kdy kolo začalo (z toho záznamu si pak každý připojený klient vypočítává zbývající čas do konce kola), následně je mu vytvořen objekt kouzelníka a hra začíná.

9.2 Synchronizace polohy

Jedním z nejdůležitějších problémů networkingu v takové hře je správně a co nejpřesněji synchronizovat polohu všech hráčů napříč všemi klienty. Při mém prvním pokusu o synchronizaci jsem si jednoduše posílal informace o poloze a následně tam objekty přesouval.

Tohle je z hlediska přesnosti nejideálnější řešení, protože když nastavíte dost malý interval kdy se bude funkce dotazovat na novou polohu, tak už pak záleží jen na odezvě serveru, jinak bude mít synchronizovaný objekt téměř identickou polohu s originálem.

Je tu ale jeden velký problém, kvůli kterému se toto řešení nedá použít. Pokud budeme objektu jednoduše pře-nastavovat souřadnice polohy na nové, tak nám bude objekt poskakovat po scéně místo toho aby se pohyboval plynule jako originál. Tento problém se dá ale taky celkem snadno vyřešit, a to použitím interpolace.

Unity samo o sobě poskytuje funkci `Lerp`, která očekává 3 argumenty. První argument je vektor počáteční polohy, druhý argument vektor cílové polohy, a třetí argument je rychlost interpolace mezi těmito vektory.

Použitím této funkce tedy můžeme jednoduše vypočítat plynulý posun objektu k cílové pozici. S tím nám ale přichází ne úplně malé zpoždění, které je samozřejmě závislé na rychlosti interpolace, kterou vybereme. Pokud vybereme stejnou rychlost jakou se pohybují originální objekty hráčů, tak na první pohled bude synchronizace vypadat správně a vše se bude pohybovat stejně rychle.

Na druhou stranu máme už jisté zpoždění od serveru, takže pokud se bude synchronizovaný objekt posouvat stejně rychle jako originální, tak budeme mít mezi nimi značné rozdíly.

Pokud ale vybereme vyšší rychlost, synchronizované objekty se budou pohybovat rychleji než originální, a to také není ideální. V praxi se využívá kombinace extrapolace a interpolace. Extrapolace je založená na odhadování budoucích hodnot, a v kombinaci s interpolací je toto řešení schopné podávat celkem rozumné výsledky.

Photon Engine obsahuje i nějaké hotové skripty, které můžete libovolně využívat nebo se jimi inspirovat. Jedním z nich je i skript, ve kterém je implementována synchronizace pohybu, a stačí jej správně nastavit, popřípadě trochu upravit a funguje opravdu dobře.

Tohle řešení jsem využíval většinu času vývoje této hry, ale nakonec jsem přešel k jinému řešení. Změnil jsem pohyb hráče z klasického posunování po scéně na pohyb silou (když hráč stiskne šipku nahoru, nastaví se objektu hráče síla působící nahoru).

Pro synchronizaci tohoto pohybu jsem tak využil jede z již hotových skriptů a to skript pro synchronizaci fyziky objektů. Tento skript tak synchronizuje veškeré síly působící na objekt, a to dost přesně.

Tohle řešení jsem následně doplnil ještě o teleportování, když je vzdálenost mezi synchronizovaným a originálním objektem moc velká, a výsledek se dost přiblížil mé ideální představě.

9.3 Synchronizace animací

Synchronizace animací je obecně jednoduchá. V mém případě jsou animace ovládány přes parametry objektu `Animator Controller`, které jsou popsány v kapitole 7. To znamená, že mi stačí si každou změnu těchto parametrů posílat také do všech ostatních klientů a `AnimatorController` se postará o všechno sám. Lze pro toto řešení také využít existujícího skriptu od Photonu, ale implementace je opravdu snadná.

9.4 Synchronizace vystřelených projektilů

Dokud se vývojář nedostane do tohoto bodu, je synchronizace pohybu, která je popsána v kapitole 8.2. naprosto postačující a stačila by na většinu multiplayerových her.

Nějaká odezva mezi klienty samozřejmě je, ale není to nic hrozného a pokud je server, na kterém se hraje, dostatečně rychlý, je to opravdu minimální rozdíl.

Bohužel, ale přichází na řadu synchronizace projektilů, které mají určitou dráhu letu, a tak přibývá další prvek, který bude mít nějaké zpoždění oproti originálu. Když se pak sečte odezva pohybu hráčů a odezva pohybu projektilů, už je to dost vidět, a hra se stává nehratelnou.

Ve většině FPS (first person shooter) hrách, se tento problém moc neřeší, protože projektily jsou implementovány pomocí paprsků, které nemají skoro žádnou dráhu letu a celou scénu protnou takřka okamžitě. Proto synchronizace nehraje roli, stačí na každém klientu vystřelit paprsek ve stejném směru jako v originále a synchronizace je hotová.

V případě mé hry jsou tu ale projektily, které mají dráhu letu, a tak je možné se jim vyhýbat nebo naopak předpovídat kam se hráč posune a tzv. předsazovat střelbu. S tím přichází stejný problém jako u hráče, a to synchronizace pohybu.

První řešení, které se nabízí, je jednoduše vytvořit projektil na všech klientech, nastavit jim stejný směr a stejnou sílu, a nechat je „letět“. Pokud by ale byla na serveru odezva třeba 50 milisekund, tak se projektil na ostatních klientech vytvoří o tu dobu později. Na originálním klientu tak bude projektil o 50 milisekund dál než na ostatních klientech.

Nejpřesnější řešení je, vypočítat si pomocí rychlosti a aktuální odezvy serveru vzdálenost, kterou už originální projektil urazil a vytvořit ostatní projektily na tom místě. Tohle funguje opravdu dobře, ale při vyšších odezvách se projektily začnou vytvářet někde v půlce scény, kde již žádný hráč nestojí, a to nevypadá dobře.

Já nakonec použil stejné řešení jako u synchronizace pozic hráčů. Tohle řešení je náročnější na celkové posílání dat mezi klienty a reálně by se na projektily nemělo využívat, protože když budou ve hře tisíce projektilů, zatíží to server ještě víc a způsobíte si tím jen větší odezvu. V této hře, ale mohou být naráz na scéně jen desítky projektilů a tak to má minimální dopad.

10 Popis skriptů

10.1 Player Scripts

V této kapitole budu popisovat skripty, které se starají o vše okolo objektů hráče. Všechny tyto skripty se nachází ve složce na cestě *„Assets/Scripts/Players/“*.

10.1.1 Animation Handler

Skript `AnimationHandler.cs` slouží pro přepínání animací při změně zbraně. V případě, že hráč na scéně vezme nějakou zbraň, je zavolána funkce z tohoto skriptu, která zařídí změnu vzhledu kouzelníka. Obsahuje například jednu hlavní veřejnou funkci `changeWeaponAnimations`, která jako argument očekává index zbraně, na kterou má byt změněn vzhled.

10.1.2 Movement

Skript `Movement.cs` slouží k obstarávání všeho okolo pohybu hráčů. Obsahuje funkci, která čeká na vstup z klávesnice a podle toho, která šipka je stisknuta, určuje směr chůze. Nastavuje animace chůze a zařizuje i zvuk při chůzi.

10.1.3 Player's Death

Skript `PlayersDeath.cs` je jeden z nejsložitějších skriptů v této kapitole, protože obstarává vše, co se stane ve chvíli, kdy je nějaký hráč trefen projektilem. Obsahuje funkce pro zapínaní animace smrti, pro zvuk smrti, funkce pro vybrání vhodného bodu, kde se znovu zrodí, počítá dobu ke zrození, a volá funkce posílané do všech klientů.

Když je nějaký hráč zasažen a umírá, tak se jeho objekt nemaže, jen se skryje. V tomto skriptu se řeší i vypínaní dalších skriptů jako `Movement.cs`, aby hráč nemohl chodit když je mrtev, stejně jako skript `Shooting.cs` a další. Vypíná se mu taky collider, aby přes něj mohli chodit ostatní hráči, a také létat projektily.

V rámci networkingu jsem na tomto skriptu strávil spoustu času, protože v průběhu vývoje se právě ohledně smrti hráče objevovalo spousta problému s tím, aby se funkce spojené se smrtí projevily na všech klientech stejně.

Největší problém byl, jak vyřešit kontrolování zásahu projektilem. V průběhu hry totiž existují dvě verze každého hráče. Originální verze (lokální verze, kterou ovládá hráč), síťová verze (verze, kterou vidí všechny ostatní klienti). Síťová verze nesmí například obsahovat stejné skripty jako ta originální, protože pak by lokální hráč stiskem kláves ovládal nejen svého hráče, ale taky síťové verze ostatních hráčů.

Proto jsem musel vyřešit problém, s kontrolováním zásahu projektilem. V případě, že bude na serveru velká odezva a projektil netrefí obě verze zároveň ale například trefí originální verzi o něco dřív než síťovou. Problém byl tedy takový, na jaké verzi hráčova objektu čekat na kolizi s projektilem. A co se má stát, když projektil trefí, nejdřív verzi hráče, která nemá funkce na obstarávání toho stavu.

10.1.4 Player Spawn

Skript `PlayerSpawn.cs` se stará o prvotní vytvoření hráče při připojení do hry. Stará se o vybrání správných animací (barvy) podle ID, které hráč dostal. Zajišťuje vybrání prázdného místa na scéně pro zrození hráče. Stará se jak o vytvoření originální lokální kopie hráče, na kterém zapíná všechny potřebné skripty, tak i o vytvoření síťové kopie všem ostatním klientům.

10.1.5 Respawn Animation Handler

Skript `RespawnAnimationHandler.cs` obsahuje zatím jen jedinou funkci, která se stará o smazání objektu, na kterém je spuštěna animace respawnu hráče, přes všechny klienty a to v případě, že animace respawnu je ukončena.

10.1.6 Teleport

Skript `Teleport.cs` zajišťuje přesun objektu, pokud zaznamená kolizi s jedním z dvojice teleportů umístěných ve scéně na pozici druhého teleportu.

Tento skript se využívá v první mapě, kde se dá procházet z jednoho kmene do druhého, a tudíž z jedné strany mapy na druhou. Tento skript má nejen hráč, ale také všechny projektily, a tak je možné střílet nečekaně jedním teleportem do druhého.

10.2 Score scripts

V této kapitole budu popisovat skripty, starající se o počítání a zobrazování skóre hráčů. Všechny tyto skripty se nachází ve složce na cestě *„Assets/Scripts/Score“*.

10.2.1 Score

Skript `Score.cs` není skript jako takový. Obsahuje třídu `Score`, která obsahuje metody pro práci se skóre každého hráče. Například metody `increaseScore`, `increaseDeaths`, `resetScore` a další.

10.2.2 Score Table

Skript `ScoreTable.cs` je zodpovědný za správné zobrazení současného přehledu skóre všech hráčů vždy, když hráč stiskne tlačítko *TAB*.

Při prvním stisknutém tlačítku *TAB* se vytvoří objekt panelu s texty, z vlastností pokoje se vybere seznam hráčů, z něj pak jména hráčů a jejich současná skóre.

10.3 UI scripts

V této kapitole budu popisovat skripty, které se starají o správné zobrazování variabilních věcí v uživatelském rozhraní a také se starají o všechny zvuky ve

hře.

Všechny tyto skripty se nachází ve složce na cestě *„Assets/Scripts/UI/“*.

10.3.1 Dont Destroy This

V Unity3D se při přepnutí z jedné scény do druhé smažou všechny vytvořené objekty v původní scéně. V případě, že potřebujete některý z objektů zachovat (obsahuje třeba skript, kde jsou uloženy důležitá data), musíte na něj vložit tento skript, který se postará o to, aby při přepnutí scén byl zachován.

Skript `DontDestroyThis.cs` obsahuje jedinou funkci, která zajišťuje právě zachování objektu mezi scénami.

10.3.2 Level Sounds

Skript `LevelSounds.cs` se stará o spouštění hudby pro příslušnou mapu.

10.3.3 Player Sound Controller

Skript `PlayerSoundController.cs` obsahuje funkce pro vypínaní a zapínaní zvuku efektů, které vydávají objekty hráčů.

10.3.4 Respawn Time

Skript `RespawnTime.cs` se stará o zobrazení zbývajícího času do znovuzrození hráče.

Obsahuje funkce pro vytvoření samotného panelu s textem, a jiné.

10.3.5 Sounds Control

Tento skript obsahuje oproti skriptu `PlayerSoundsControl.cs` z kapitoly 10.3.3. funkce pro práci se zvukem a efekty všech objektů vydávajících zvuky mimo hráče.

10.3.6 UI Handler

Skript `UIHandler.cs` obsahuje funkce, které zobrazují nabídku po stisku klávesy *ESC*, a stará se o správné nastavení kamery pro různá rozlišení.

10.3.7 UI Sounds

Skript `UISounds.cs` obsahuje funkce, které spouští zvuky, které jsou slyšet v menu obrazovkách. Například zvuk při kliknutí na tlačítko a zvuk při projetí nad tlačítkem.

10.3.8 Update Bot UI

Skript `UpdateBotUI.cs` obsahuje funkce pro nastavování správných obrázků kouzelníku ve spodním panelu ve hře. Dále obsahuje funkci, která vrací pole obrázků, ukazujících, kolik zbývá hráči projektilů k vystřelení. Tato funkce se pak používá se skriptu `Shooting.cs`.

Obrázek 7: Spodní panel

10.4 Utilities

V této kapitole budu popisovat pomocné skripty, které přímo neovlivňují dění ve hře.

Všechny tyto skripty se nacházejí ve složce na cestě *„Assets/Scripts/Utilities“*.

10.4.1 Console Handler

V průběhu vývoje se mi začaly hromadit věci, které jsem pro testování musel různě vypínat a zapínat nebo upravovat a tak jsem si vytvořil jednoduchou konzoli, přes kterou mohu volat různé funkce kdykoliv během běhu hry.

Skript `ConsoleHandler.cs` se taky stará o zobrazování a skrývání konzole a samotné volání funkcí.

10.4.2 Information Handler

Skript `InformationHandler.cs` obsahuje funkce spouštěné z konzole a základní informace o nastavení pro následné spuštění hry. Na objektu, na kterém je spuštěn tento skript, je také spuštěn skript `DontDestroyThis.cs` z kapitoly 10.3.1. a tudíž se tento objekt vyskytuje napříč všemi scénami.

10.4.3 Load Level

Skript `LoadLevel.cs` obsahuje funkce, které zařizují načtení nové scény (*Lobby Menu*, *Main Menu* a další).

10.4.4 Respawn Handler

Skript `RespawnHandler.cs` obsahuje funkce, které se starají o tzv. vypnutí místa pro znovuzrození hráčů a pro vytváření zbraní. Funguje to tak, že pokud hráč stojí na nějakém bodě pro zrození, tento bod se vypne, a hráč, který se má zrovna narodit, se na tomto místě nenarodí.

Stejně tak to funguje i se zbraněmi, tímto se zabrání skládání více zbraní na jedno místo a dalším problémům s tím spojených.

10.4.5 Round Timer

Skript `RoundTimer.cs` obsahuje veškeré funkce obsluhující počítání času do konce kola, ukončení kola a následné spuštění kola nového. Obsahuje funkce, pro vypínaní hráčů, resetování časového limitu, resetování skóre a další.

10.4.6 End Screen Handler

Skript `EndScreenHandler.cs` obsahuje funkce, které slouží pro správné nastavení obrazovky, která se zobrazuje na konci každého kola. Když skončí kolo musí seřadit hráče podle skóre, a nastavit obrázky a texty se jmény a počtem skóre, které se pak zobrazují v této obrazovce.

10.4.7 Settings

Skript `Settings.cs` obsahuje funkce pro nastavení objektů hráčů a objektů hráčů umělé inteligence.

10.5 Guns

V této kapitole budu popisovat skripty, které obstarávají vše kolem zbraní a projektilů.

Všechny tyto skripty se nachází ve složce na cestě *„Assets/Scripts/Guns/“*.

10.5.1 Bullets

V této podkapitole budu popisovat skripty, starající se o projektily. Tyto skripty se nachází ve složce na cestě *„Assets/Scripts/Guns/Bullets“*.

Bullet 1 Action

Skript `Bullet1Action.cs` obsahuje funkce, které se starají o akce, které vykonává projektil vystřelený ze základní zbraně. Obecně jsou v tomto skriptu funkce pro pohyb projektilu po scéně. Také obsahuje funkci, která kontroluje vzdálenost, kterou projektil urazil a pokud přesáhne určitou hranici, je projektil zničen.

Bullet 2 Action

Druhá zbraň v pořadí už je trochu složitější než základní zbraň, a tak skript `Bullet2Action.cs` obsahuje složitější funkce. Tato zbraň vystřelí 3 projektily. Jeden projektil (prostřední) letí rovně, další dva letí s menším vychýlením, každý na jinou stranu.

Skript se tedy stará o správné otočení a posun každého projektilu. Mimo to také obsahuje funkci pro let projektilu a také obsahuje funkce pro kontrolu vzdálenosti letu.

Bullet 3 Action

Třetí zbraň je ještě trochu složitější než ta druhá. Tato zbraň vystřelí 5 projektilů za sebou. Každý je trochu jinak vychýlen a střídají se také pozice (jeden projektil letí výš, druhý níž a takhle dokola). Tento skript se tedy také stará o správné otočení a posun každého projektilu, stejně tak jako let a kontrolu vzdálenosti.

Bullet 4 Action

Projektil používající tento skript je velmi podobný základnímu projektilu. S tím rozdílem, že když tento projektil narazí na nějakou překážku, ať už hráče nebo objekt na mapě, tak se vytvoří větší objekt s animací výbuchu.

Tento jediný projektil má plošný účinek. Až na funkci výbuchu, je tedy skoro stejný jako skript z kapitoly 10.5.1.1.

Bullet 5 Action

Zbraň vystřelující (v tomto případě pokládající) tento projektil, je odlišná od všech ostatních. Tento projektil je statický, a na místě, kde jej hráč položí, zůstane do doby, kdy na něj nějaký hráč šlápne.

Oproti ostatním skriptům, je zde absence funkce pohybu po scéně a kontrola vzdálenosti.

Bullets Collision

Skript `BulletsCollision.cs` je spuštěn na všech projektilech. V tomto skriptu je jedna nejdůležitější funkce a to `OnTriggerEnter2D`. Tato funkce je spuštěná ve chvíli, kdy projektil zaznamená kolizi s jiným colliderem, který je označen příznakem trigger.

V této funkci se pak nachází akce pro jednotlivé projektily - co se s nimi stane po nárazu do hráče, nebo jiného objektu.

Trap Action

Skript `TrapAction.cs` je spuštěn na objektu, který je vytvořen, když hráč šlápne na past vytvořenou zbraní, jejíž akce je popsána v kapitole 10.5.1.5.

Tento objekt se vytvoří nad hráčem a padá směrem na něj s tím, že je na něm spuštěná animace. Poté, co narazí na hráče, je objekt zničen a hráč zabit. Skript tak obsahuje funkce, které se starají o výše popsané akce.

10.5.2 Bullet Spawn Handler

Protože se hráč může pohybovat jen do 4 směrů, má samotný objekt hráče okolo sebe 4 pod-objekty, které určují pozici, na které se budou vytvářet projektily.

Skript `BulletSpawnHandler.cs` obsahuje funkce, které zabraňují hráčově možnosti střílet, pokud se dotýká nějakého objektu ve scéně (mimo hráče).

Příklad. Když hráč jde doleva tak dlouho až narazí na levou hranu scény, a pokusil by se tímto směrem vystřelit, tato funkce mu v tom zabrání.

10.5.3 Equip Gun

Tento skript obsahuje jedinou funkci, která je volána ze skriptu `GetGun.cs`, který je popsán v kapitole 10.5.4.

Tato funkce se stará o správné nastavení nové zbraně, kterou hráč ve scéně sebral.

10.5.4 Get Gun

Skript `GetGun.cs` obsahuje jednu funkci, která čeká na kolizi se zbraní vytvořenou ve scéně.

V případě kolize hráče s touto zbraní, si funkce vybere ID zbraně a zavolá funkci ze skriptu `EquipGun.cs`, která se postará o správné nastavení a připravení zbraně.

10.5.5 Gun Settings

Skript `GunSettings.cs` obsahuje proměnnou ID zbraně, a funkci, která smaže zbraň ze scény všem připojeným klientům.

10.5.6 Gun Spawn

Skript `GunSpawn.cs` se stará o náhodné vytváření zbraní do scény každých 25 sekund. Tento interval se dá libovolně změnit podle potřeby.

10.5.7 Shooting

Největší a nejsložitější ze skriptů v této kapitole. Skript `Shooting.cs` obsahuje funkce, které zajišťují střelbu právě aktivní zbraní. Obsahuje také funkce, které počítají vystřelené projektily, zařizují přebíjení základní zbraně, nebo výměnu na základní zbraň v případě, že počet možných výstřelů klesne na nulu. Dále se tam nachází funkce pro vydávání zvuků střelby a spuštění animace střelby.

Jedny z nejdůležitějších funkcí jsou funkce `weapon1Fire`, `weapon2Fire`, `weapon3Fire`, `weapon4Fire` a `weapon5Fire`. Tyto funkce zajišťují požadovaný efekt zbraně.

Funkce `weapon1Fire` je jednoduchá. Je to funkce základní zbraně, která nemá žádný efekt, a jednoduše při stisku klávesy pro střelbu vystřelí jeden projektil.

Funkce `weapon2Fire` už je složitější. Tato zbraň střílí 3 projektily, jak je popsáno v kapitole 10.5.1.2. V této funkci se tak počítá rotace, se kterou se pak projektily pohybují.

Funkce `weapon3Fire` je odlišná od jiných. Tato zbraň střílí 5 projektilů v řadě za sebou s menšími rozestupy a odlišnou rotací. První projektil je vystřelen ihned po stisknutí klávesy. Další jsou pak ale vystřeleny vždy až po uplynutí nějaké doby (v současnosti je tento interval nastaven na 0.1 sekund). Stejně jako u předchozí funkce se zde počítá rotace + posun projektilu.

Funkce `weapon4Fire` je funkcionálně totožná s funkcí `weapon1Fire`.

Funkce `weapon5Fire` je úplně odlišná. Tato funkce slouží ke kladení pastí. Protože ale hráč klade pasti pod sebe a nestřílí je zbraní před sebou, jako je tomu u ostatních zbraní. Nedají se tak používat objekty, reprezentující pozici pro vytvoření projektilů. Proto tato funkce, na základně směru, kterým je hráč otočen, počítá pozici, na kterou se past vytvoří.

Další funkce v tomto skriptu slouží například pro upravování obrázků znázorňující zbývající počet projektilů k výstřelu. Funkce, která vybírá *„spawn“* pro projektily na základě pozice a další.

10.6 Networking

V této kapitole budu popisovat skripty, které se starají o správné vytváření pokojů, připojování hráčů do hry, a další.

Všechny tyto skripty se nacházejí ve složce na cestě *„Assets/Scripts/Networking“*.

10.6.1 Lobby Buttons

Skript `LobbyButtons.cs` obsahuje funkce, které nastavují tlačítkům na obrazovce `LobbyMenu` jejich akce.

Pro tlačítka regionů nastavují akce pro připojení do jednotlivých regionů atp.

10.6.2 Lobby Match Maker

Skript `LobbyMatchMaker.cs` se spouští po připojení k regionu tedy po připojení do jednotlivých lobby.

Hned po připojení začne tento skript vyhledávat a zobrazovat pokoje, s následným automatickým obnovováním.

Obsahuje funkce pro vytvoření nového pokoje a také pro připojení ke stávajícímu pokoji.

10.6.3 Match Maker

Skript `MatchMaker.cs` je spuštěn hned poté, co se hráč připojí k novému nebo existujícímu pokoji. V tomto skriptu se hráčovi generuje ID, a nastavují další nutné věci.

Tento skript také obsahuje funkce, které jsou spuštěné jen v případě, že je vytvořen nový pokoj. Funkce, které například nastavují čas začátku hry, aby si ostatní hráči mohli odpočítat zbývající čas do konce kola.

10.7 Bots

Kvůli testování a také možnosti hráče zahrát si tuto hru i bez ostatních lidských hráčů, jsem vytvořil jednoduchou umělou inteligenci, se kterou přichází také několik nových skriptů.

V této kapitole tak budu popisovat všechny skripty, které se týkají této umělé inteligence neboli Botů.

Všechny tyto skripty se nachází ve složce na cestě *„Assets/Scripts/Bots“*.

10.7.1 Bots Animation Handler

Skript `BotsAnimationHandler.cs` obsahuje jedinou funkci, která slouží ke správnému nastavení animací Bota, závislých na jeho současné akci (chůze, střelba, umírání, a další).

10.7.2 Bots Death

Skript `BotsDeath.cs` je velmi podobný skriptu `PlayersDeath.cs` popisovaném v kapitole 10.1.3.

Jsou zde rozdíly hlavně v místech, kde se ve skriptu `PlayersDeath.cs` pracuje se skripty, určenými pro hráče, které samozřejmě Boti nemají, a další.

Jedním z rozdílů je například změna *„profilové“* fotky kouzelníka ve spodní části uživatelského rozhraní. Při úmrtí hráče se změní hlava kouzelníka na zobrazení smrtky, která znázorňuje, že hráč zemřel. Také se při této situaci začíná zobrazovat na obrazovku odpočet času, než se hráč znovu zrodí, a tato akce také u Botů není třeba.

10.7.3 Bots Moving

Skript `BotsMoving.cs` je jedním ze složitějších skriptů v této hře. V tomto skriptu už je obsažena první část umělé inteligence Botů - jejich pohyb. Z počátku měli Boti jen náhodný pohyb s vyhýbáním se objektů, ale to bylo samozřejmě nedostačující. Současný algoritmus je tedy následující.

Bot si na začátku algoritmu vybere nejbližšího nepřítele (ať už jiného bota, nebo hráče) a podle nepřítelovy pozice vůči jemu samotnému, se mu vygenerují dva směry (tohle je zapříčiněno tím, že po scéně je možné se pohybovat jen do 4 směrů, tudíž pokud se chcete s hráčem pohybovat směrem k levému hornímu

rohu, musíte střídavě chodit doleva a nahoru, nebo nejdřív nějakou dobu doleva, a pak zase nahoru, nebo samozřejmě naopak). Z těchto dvou směrů se náhodně vybere jeden, kterým se pak Bot vydá.

Po výběru směru se zkontroluje, zda-li je možné se tímto směrem pohnout, tzn. jestli v tomto směru není nějaká překážka. Pokud se tímto směrem pohnout nedá, vybere se druhý směr z vygenerované dvojice.

Následně se mu náhodně vygeneruje počet sekund. V průběhu této doby bude pronásledovat již vybraného nepřítele. Až tento čas vyprší, Bot se porozhlédne po jiném nejbližším nepříteli.

Celý algoritmus se pak každý frame opakuje.

```
1  private void chooseDirection()
2  {
3     if (!choosedByRay)
4     {
5        getSteps();
6
7        directionIndex = getDirectionIndex(chaseEnemy());
8        int i = 0;
9
10       while (true)
11       {
12          if (checkSpawn(directionIndex))
13          {
14             break;
15          }
16          else if(i < 1)
17          {
18             directionIndex = getDirectionIndex(chasingDirections[i])
19             i++;
20          }
21          else
22          {
23             directionIndex = Random.Range(0, posibleDirections.
                  Length);
24          }
25       }
26
27       actualDirection = posibleDirections[directionIndex];
28    }
29    else
30    {
31       choosedByRay = false;
32    }
33 }
```

Zdrojový kód 1: Funkce, která rozhoduje o směru pohybu Bota

10.7.4 Bots Shooting

Skript `BotsShooting.cs` je znovu velmi podobný skriptu, sloužícímu střelbě hráčů, popsán v kapitole 10.5.7. Obsahuje funkce pro střelbu všemi zbraněmi, změnu zbraní, a tak dále.

10.7.5 Bots Shooting Helper

Skript `BotsShootingHelper.cs` je druhá část umělé inteligence, kterou zatím Boti disponují. V tomto skriptu jsou funkce, které rozhodují, zda-li má Bot vystřelit nebo ne.

Algoritmus je následující. Vybere se směr, kterým se Bot dívá. Následně se vytvoří paprsek na stejné pozici, ve které se nachází a otočí se směrem, kterým se Bot dívá. V další části se paprsek vystřelí a kontroluje se, čím vším proletěl. Pokud paprsek na své cestě zaznamenal nepřítele (hráče, nebo jiného Bota), ale před tím zaznamenal nějakou překážku ve scéně, tak Bot nevystřelí. Pokud však první, na co paprsek narazil, byl nepřítel, Bot vystřelí.

Skript dále obsahuje funkce, které kontrolují všechny 4 směry, a nejen směr na který je Bot otočen. V případě, že by se v nějakém směru objevil nepřítel, a nebyl schován za překážkou, Bot by se otočil a po nepříteli střelil. Tím by se zajistilo stejné chování jako má reálný hráč, který samozřejmě kontroluje všechny směry okolo sebe.

Tyto funkce se však nepoužívají, protože v jejich současném stavu nepodávaly správné výsledky a bohužel mi nezbyl čas na jejich opravení. Podobného efektu jsem ale docílil tak, že ve skriptu `BotsMoving.cs` popisovaném v kapitole 10.7.3. jsem generovaný čas, který Bot stráví pronásledováním jednoho vybraného nepřítele, omezil vždy na jednu sekundu. To znamená, že pokud se kolem Bota bude pohybovat jiný nepřítel, a to i v jiném směru, než je Bot otočený, tak si v další sekundě vybere k pronásledování jeho, otočí se na něj, a může střílet.

Tato logika podává obstojné výsledky, ale bohužel to neřeší stav, kdy je nepřítel od Bota odhalený a jiným směrem, ale třeba na druhé straně mapy. Pokud tedy bude nějaký nepřítel blíž, a klidně i schován za nějakou překážkou, Bot si vybere toho bližšího i když by snadnější cíl byl ten co je dál a není schován.

Závěr

Hra v současném stavu obsahuje plně funkční online multiplayer. Obsahuje jednu herní mapu s možností připojení až čtyř hráčů zároveň. Ve hře je implementováno 5 různých zbraní, zajímavé prvky jako teleportování z jedné strany mapy na druhou, poutavá grafická stránka a jednoduché a uživatelsky přátelské ovládání. Z tohoto pohledu jsem splnil vše, co jsem chtěl a co bylo v zadání této práce. Dále jsem nad rámec zadání práce vytvořil užitečnou konzoli pro jednoduché nastavování hry a jednoduchou umělou inteligenci. Na druhou stranu se mi nepodařilo vyladit synchronizaci klientů do ideálního stavu a na této stránce hry budu muset v budoucnu ještě zapracovat.

Vytvářením této hry jsem si osvojil pokročilejší prvky tvorby 2D her a práci s networkingem. Dále jsem se naučil nahrávat a upravovat zvukové efekty a tvořit 2D animace. Naučil jsem se také, jak důležitou roli hraje návrh systému, i když se nejedná o moc velký projekt.

Conclusions

The game in its current state includes a fully functional online multiplayer. It contains one map with the ability to connect up to four players simultaneously. There are 5 different weapons in the game, interesting features like teleport from one side of the map to another, interesting graphic design and simple and user-friendly control. From this perspective, I did everything I wanted and what was the assignment. I also created a useful terminal for easy game setup and simple artificial intelligence. On the other hand, I have not been able to fine-tune client synchronization to the ideal state and I will still have to work on this.

By creating this game, I have learned the more advanced features of 2D game creation and networking. I also learned to record and edit sound effects and create 2D animations. I also learned how important the system design is, even if it is not a very big project.

A Zprovoznění aplikace

A.1 Potřebné požadavky

Hra je vydána na platformu Windows. Pro spuštění hry není potřeba nic dalšího instalovat. Pro nahlédnutí do projektu je potřeba nainstalovat prostředí Unity3D. Pro nahlédnutí do kódů lze použít jakékoliv vývojové prostředí nebo textový editor.

A.2 Spuštění aplikace

Hru je možné spustit rovnou z přiloženého CD. Popřípadě si spustitelný soubor zkopírovat do počítače a odtud také spouštět.

B Obsah přiloženého CD

Na přiložené CD se nacházejí tři soubory a jedna složka.

`WizardsOfAzear.exe`
: Tento soubor slouží ke spuštění samotné hry.

`UnityDownloadAssistant-5.6.1f1.exe`
: Tento soubor slouží pro instalaci Unity3D prostředí.

`Dokumentace.pdf`
: PDF soubor, který obsahuje tento text.

`WizardsOfAzear/`
: Tato složka reprezentuje samotný projekt, který lze otevřít pomocí prostředí Unity3D.

Literatura

[1] UNITY. Unity Technologies. Unity platforms [online]. [cit. 2017-05-17]. Dostupné z: https://unity3d.com/unity/multiplatform

[2] PHOTONENGINE. Exit Games. Photon PUN Features [online]. [cit. 2017-05-17]. Dostupné z: https://www.photonengine.com/en/PUN