DELTA - Střední škola informatiky a ekonomie, s.r.o.
Ke Kamenci 151, Pardubice

Branch Brawl - multiplayer 3D hra

Jan Skládal
4.A
Informační technologie (18-20-M/01)

2023/2024

Zadání maturitního projektu z informatických předmětů

Jméno a příjmení:	*Jan Skládal*
Pro školní rok:	*2023/2024*
Třída:	3.A
Obor:	*Informační technologie 18-20-M/01*
Téma práce:	*3D FPS hra v Unity – Forest fights*
Vedoucí práce:	

Způsob zpracování, cíle práce, pokyny k obsahu a rozsahu práce:

Cílem projektu je vytvořit 3D hru z prvního pohledu hráčské postavy. Hra bude zaměřená na prvky přežití a tedy nebude mít jednotlivé úrovně, ale bude arkádního stylu. Při každém spuštění hry se hráč objeví v lese bez vybavení. Prvním cílem je sehnat zbraň, kterou tvoří předměty náhodně rozmístěné po mapě. Hlavní myšlenkou hry je fakt, že zbraně tvoří větve/klacky, které fungují jako střelné zbraně.

Průběh hry se dělí na kola, kde má hráč příležitost využívat nově přístupné předměty dle aktuálních map a hledat různé způsoby, jak zbraně s předměty kombinovat. Mezi koly hráči dostávají možnost rozhodnout o následné mapě.

V rámci projektu nastuduji, jak pracovat na objemnějším projektu v prostředí herního vývoje v Unity a jak se vypořádat s tvorbou hry v prostředí multiplayeru.

Hra bude tvořena v enginu Unity a skripty budou psané v C#.

Stručný časový harmonogram (s daty a konkretizovanými úkoly):

Září – Hráč – ovládaní, pohyb, interakce s okolím, plánování

Říjen – Generace zbraní, funkce ohledně zbraní, střelba, level systém zbraní, vylepšování zbraní

Listopad – Multiplayer, systém zdraví a kol

Prosinec – Tvorba map, práce na vizuální stránce hry

Leden – Beta verze hotová – testování hry

Únor – Ladění nalezených chyb a finalizace hry samotné; Začátek práce na dokumentaci

Březen – Dokumentace projektu

Prohlášení

Prohlašuji, že jsem maturitní projekt vypracoval(a) samostatně, výhradně s použitím uvedené literatury.

V Pardubicích dne 31.3.2024 .....

Poděkování

Upřímně děkuji panu Mgr. Janu Mottlovi za odborné vedení maturitní práce a za rady při tvorbě projektu.

Anotace

Tato dokumentace popisuje vývoj 3D hry pro více hráčů a problematiku, se kterou se můžete v rámci multiplayer her setkat. Obsahuje popis samotné hry, implementaci, strukturu projektu, různé algoritmy a obsáhlý seznam grafických děl.

Klíčová slova

C#, Unity Engine, Hra pro více hráčů, 3D grafika, TPS

Annotation

This documentation describes the development of a 3D multiplayer game and the issues that you may encounter in multiplayer games. It contains a description of the game itself, the implementation, the structure of the project, the various algorithms and a comprehensive list of the graphical assets.

Keywords

C#, Unity Engine, Multiplayer game, 3D graphics, TPS

Obsah

Úvod

Videoherní průmysl v dnešní době tvoří nepostradatelnou součást naší kultury s více než 3 miliardami hráčů v roce 2023 [1]. Není proto divu, že tento průmysl vzrostl na průmysl multimiliardový.

Takovým dětským snem každého hráče, který na hrách vyrůstal, je si kromě hraní i nějakou hru také vytvořit. Minimálně to tak bylo i v mém případě, a když se bavíme o dětském snu, dostáváme se k myšlence. K myšlence, která tento projekt vytváří.

Nápadem bylo vzít hráče zpět do dětství a rozpomenout si na dobu, kdy jako dítě jste na zahradě a naleznete klacek. Není to ale ledajaký klacek. Tento klacek až moc podobně připomíná střelnou zbraň! Jako dítě vás hned začínají napadat všelijaké scénáře, kdy se objevíte například uprostřed bojového pole, a právě tento klacek vám tvoří vašeho silného, střely střílejícího parťáka. Co kdybych vám ale řekl, že nyní tento svět nezůstává pouze ve světě dětské fantazie, ale přenáší se do vašich herních obrazovek! A přesně o tom je hra Branch Brawl.

Cílem tohoto projektu je vypořádat se s výzvou herního vývoje a naučit se pracovat s nástroji k tomuto účelu dostupnými. Zjistit nejen jak se hry vyvíjí, ale jaký je celý proces za vytvářením vlastního vymyšleného světa, následného usazení konceptu hry, který do tohoto světa bude zapadat, a k tomu ty již vymyšlené herní koncepty z myšlenek přetáhnout do reality a reálně tyto systémy implementovat, a to vše bez jakékoliv předchozí zkušenosti v tomto směru. A aby to nebylo vše, pochopit koncept multiplayer her a jejich implementace a vše na závěr zaobalit do vlastnoručně tvořeného grafického vizuálu.

1. Teoretická část

1.1 Topologie sítí u multiplayer her

V nejen herním, ale obecně ve světě počítačových sítí pracujeme ze základu s 2 modely síťového propojení.[4]

- Server - Klient
- Peer to Peer (P2P)

1.1.1 Topologie Server - Klient

Server-klient model je jedním z nejběžnějších přístupů k implementaci multiplayer her. Je ideální pokud je požadována stabilita a bezpečnost, proto je využíván u většiny světových herních titulů jakými může být například League of Legends, CS:GO, Minecraft a další.

Tato topologie závisí na existenci 1 dedikovaného serveru, na který se následně klienti napojí. V tomto modelu veškerá komunikace prochází právě přes tento dedikovaný server, a to nám umožňuje veškerou komunikaci na serveru monitorovat.[7][5]

Výhody:

- Centrální autorita: V server-klient modelu slouží server jako centrální autorita, která řídí herní logiku a udržuje konzistenci stavu hry mezi všemi hráči. To zajišťuje, že veškeré akce a události v hře jsou synchronizované a správně interpretované všemi účastníky.
- Bezpečnost a integrita dat: Centrální server umožňuje implementaci bezpečnostních opatření pro kontrolu nad proudícími daty. To dělá obtížnější vytvářet podvodné modifikace a provádět manipulaci nad zasílanými údaji.
- Lepší řízení výkonu a zátěže vůči klientovi: Model umožňuje více možností pro optimalizaci výkonu, která přetahuje zátěž z klienta na server. Vytváříme tak lehčí klienty.

Nevýhody:

- Závislost na serveru: Pro provoz tohoto modelu je potřeba aby běžel stabilní a spolehlivý server. Pokud hráčům neumožňujeme spouštět vlastní instance serveru, tak budeme vždy omezováni v počtu maximálního počtu hráčů určeného výkonem a množstvím vlastněných serverů.

- Náročnější implementace: Tento model pro nás znamená, že musíme místo 1 univerzálního programu vytvářet programy 2. Jeden který bude sloužit jako serverový program a druhý právě ve formě klienta, který následně budeme distribuovat mezi hráče.
- Nákladné na provoz: Mít neustále spuštěný server pro provoz je nákladné.

1.1.2 Topologie Peer to Peer (P2P)

Peer to peer model je často používanou alternativou pro tvorbu multiplayer her. Je nejčastěji používán u menších a arkádových titulů.

Pokud se bavíme o peer to peer komunikaci můžeme k ní přistupovat 2 způsoby. **Klient - Klient**, kdy si jsou klienti rovnocenní a **Host - Klient**, kdy jeden z klientů zároveň zastává roli serveru a poté pracuje podobně jako Server - Klient model.

První varianta Klient - Klient se v herním průmyslu téměř nepoužívá z důvodu hodnot spolehlivosti a udržení konzistence. V následujícím textu budu mluvit pouze o Host - Klient modelu.[7][5]

Výhody:

- Nenákladné: Model eliminuje potřebu centrálního serveru. Pro funkčnost hry nemusím jakožto tvůrce hry nic kontrolovat. Hráči si komunikaci zprostředkovávají sami. Proto je toto řešení velmi populární u *indie* her.

Nevýhody:

- Bezpečnostní rizika: Vzhledem k absenci přístupu ke komunikaci hráčů ztrácíte jako tvůrce hry autoritu nad komunikací.
- Výhoda hostícího klienta: Hostícímu klientovi jde jen stěží zabránit možnost podvádění, vzhledem k faktu, že server běží přímo na jeho zařízení. Výhodou hostícího klienta je i nulová latence z jeho strany. To tomuto klientovi dává nadměrnou výhodu nad ostatními klienty.

1.2 Nutnost autority serveru

Pokud se bavíme o multiplayer hrách, často zmiňovaným tématem je podvádění hráčů. Pokud se hráč rozhodne podvádět, tak na úkor jeho komfortu silně zhoršuje zážitek ostatních hráčů. Naším cílem je možnostem podvádění zamezit. To není ale tak snadný úkol právě proto, že

potřebujeme zachovat responzivitu hry a zároveň pracovat s latencí. Více o problematice latence v navázání na tuto kapitolu je rozepsáno v kapitole *Problematika latence v multiplayer hrách* 1.3.

Jedním ze základních a nejvíce účinných principů je zachování autority serveru. Autorita serveru má kromě anticheat výhod i výhodu v zachování kontinuity herní relace. To znamená, že 1 instance hry, právě ta na serveru, určuje správný stav hry a ostatní instance slouží jako loutky pro zobrazení dění na serveru. To také zamezuje vytváření situačních konfliktů.

V tomto modelu klient zastává takzvané funkce "hloupého klienta". Hloupý klient nemá autoritu a celkově správným principem je tomuto klientovi něvěřit. To znamená také omezit jeho vstupy pouze na ty, ke kterým musí mít oprávnění. Vezmeme ku příkladu pohyb hráče. Špatným přístupem by bylo povolit klientovi pohyb stylem, kdy klient zadá vstup k pohybu, ten se zpracuje lokálně a nakonec přemístění vyvolá zavoláním serveru "přemísti mne na souřadnici [22, 23]". Toto klientovi otevírá endpoint pro přemísťování bez omezení, a tím umožňuje jednoduché zfalšování vstupu. Náhle zkušený cheater má ve hře schopnost teleportace. Lepším přístupem je místo zpracování vstupu na straně klienta vstupní hodnotu přeposlat na server. Příkladem je upravená zpráva "Hráč se pohnul vpřed o vzdálenost 1". Samotný pohyb se poté zpracuje až na serveru. Samozřejmě toto také plně neřeší možnost zfalšování vstupu, ale již to vytváří překážky, se kterými se potencionální cheater musí vypořádat. Zároveň se tento přístup také jednodušeji zaobalí do dalších kontrolních vrstev, které detekují nesprávné chování klienta. Příkladem těchto vrstev může být limitování počtu vstupu v rámci herního ticku.

Praktikou je přeposílat hráčské vstupy a samotné zpracování těchto vstupů konat až na serveru, nikoliv u klienta.

Porušení autority serveru

Mohou však nastat situace, které nemají jednoduché řešení pro autoritu serveru. V tyto výjimečné situace je promíjející převést autoritu na stranu klienta. Jakožto pro vývojáře je však důležité vědět, kdy si toto narušení může dovolit.

Jedním z nejznámějších výskytů autority klienta je střelba na protihráče. Pokud jakožto klient vystřelím kulku vůči nepříteli a mám zaměřovač přímo na jeho hlavě, očekávám, že protihráče zasáhnu. Problémem nastává fakt, že v tento moment já v mé lokální projekci světa vidím pozici protihráče opožděně vůči reálnému stavu na serveru. Pokud bych v tento moment vystřelil na serveru, hráče nemusím zasáhnout. Nevýhodou je, že pokud nyní umožníme kli-

entovi zaslat zprávu ”Zasáhl jsem hráče X a ubírám mu N životů.”otevře se další vstup pro cheatery k zneužití. Autorita klienta se nedá plně ochránit proti cheatům, cheatování se ale dá ztížit způsobem implementace, která se hůře falšuje. Takovým řešením v tomto případě může být místo poslání přímé reference na hráče, který byl zasažen, poslat aktuální pozici střílejícího hráče, poté poslat pozici zasaženého hráče a náledně vykonat střelbu na serveru v rámci simulace kdy kontrolujeme jestli by opravdu střelba byla schopná protihráče zasáhnout a jestli je pro hráče reálné se v těchto pozicích vyskytovat. Nyní již místo zprávy ”Zasáhl jsem X hráče”. posílám zprávu ”střílím jakožto X hráč z Y pozice na protihráče v Z pozici”. Tento přístup stále má chyby, ale vytváření cheatů je náhle několikanásobně náročnější a většinu potencionálních cheaterů od vytváření cheatů odradí. [2]

1.3 Problematika latence v multiplayer hrách

Hry pro více hráčů, které operují přes internet, musejí řešit nepříznivé problémy spjaté se síťovou komunikací. Jsou to problémy se kterými se, v porovnání k hrám pro jednoho hráče nebo lokálním multiplayer hrám, nesejdeme. Největším z těchto problémů je latence (lag). Latence je veličina popisující zpoždění mezi provedením akce a zobrazením očekávaného výsledku. Čím vyšší je latence, tím je hra více nehratelnou.[6][3]

Latence se stává obvzlášť problémem, když potřebujeme jako klient vykonat akci na serveru. V ten moment my zadáme příkaz, on se přepošle na server, tam se provede a poté je výsledek poslán zpátky na klienta. Tato akce však není instantní a vytváří se prodleva, než se výsledek vrátí. Jako hráč však chci v rámci responzivity dostat výsledek ihned. Pokud ho neobdržím, bude pro mne zážitek z hry neintuitivní.

Latence se nedá eliminovat, dá se s ní však pracovat. Existují určité praktiky, které se používají k přelstění hráče. Cílem je zároveň přelstít hráče vytvořením iluze responzivity a k tomu zachovat autoritu serveru.

1.3.1 Client prediction

Jak již z anglického názvu vyplývá, praktika client prediction slouží na bázi předpovídání klienta jakým způsobem se vstup, co bude zaslán na server, zachová. Znázornění, jakým způsobem toto předpovídání funguje v praxi, je v následující sekci znázorněno na pohybu hráče.

1. Hráč zadá vstup ”Jdi vpřed o vzdálenost 1.” zmáčknutím klávesy pro chůzi vpřed. **Hráč je na pozici [20, 23] v čase 10.**

2. Na server je zaslán RPC (remote procedure call) ”Jdi vpřed o vzdálenost 1.”, v tento moment se také vstup propočítá lokálně u zadávajícího klienta. Výsledná pozice tohoto zpracování se zapíše lokálně společně s aktuálním tickem (hodnotou herního času). **Hráč je na pozici [21, 23] v čase 11, pozice zapsána.**

3. Na serveru je RPC obdržen, hráč je posunut. Zpět na klienta je zaslán výsledek zpracování s původní tick hodnotou. **Hráč byl posunut na pozici [21, 23] v čase 11.**

4. Klient obdržuje výsledek zpracování a ten je porovnán s předpovězenou hodnotou. Pokud se hodnoty neshodují, hráč je přemístěn na pozici zadanou serverem. **Hodnoty [21, 23] a [21, 23] se shodují, předpovědění bylo úspěšné, přemístění se nekoná.**

Tím ale samotný proces nekončí. V momentu, co se dostaví zpracovaná hodnota ze serveru, klient již s vysokou pravděpodobností zaslal nespočet dalších vstupů, které již také byly poslány na server pro ověření. Toto zpracování popisuje praktika *Server reconciliation* 1.3.2 a bez této praktiky client prediction nemůže fungovat.[6][3]

1.3.2 Server reconciliation

Tato praktika funguje ruku v ruce s praktikou *Client prediction* 1.3.1, a tak i zároveň tato kapitola navazuje na zmíněnou kapitolu.

V moment, kdy obdržíme od serveru zprávu se zpracovanou pozicí a tato pozice se neshoduje s naší předpovězenou, by měl být hráč přemístěn na pozici zpracovanou. Toto přemístění nemůže být vykonáno jen tak. Klient s nejvyšší pravděpodobností již zpracoval nespočet jiných vstupů, a tak by jeho aktuální předpovězená pozice ztratila všechny vstupy, které byly zadané po této nesprávné predikci.[6][3]

Způsob řešení tohoto problému jaký zastává tato praktika je následný. Po zmíněné akci upravení a přesunutí na správnou pozici je využito zapsaných vstupů z minulosti. Vstupů zadaných po vstupu špatně předpovězeném. Tyto vstupy jsou znovu provedeny, tentokrát již však z nově upravené pozice, a jejich hodnoty výsledků jsou aktualizovány. Tento postup je znázorněn na nadcházející ukázce.

1. Klient obdržuje ze serveru odpověď, že pozice zpracovaného vstupu v čase 12 je [22, 23]. Klient nachází lokální záznam pozice z času 12. Tím je [23, 23]. Hodnoty se neshodují a tak dochází na přepisování.

2. Objekt je přemístěn na správnou serverovou pozici. Tím začíná proces rekonciliace. **Aktuální lokální pozice je změněna na [22, 23].**

3. Klient vybírá všechny vstupy zadané po čase 12. Za každý zápis vstup zprocesovává a přepisuje původní hodnotu stavu objektu v daném čase novým stavem po zprocesování. Tento úkon se provádí, dokud nejsou všechny vstupy znovu zpracovány. **Mezi dalšími vstupy bylo 3x posunutí vpřed, aktuální zobrazovaná pozice u klienta tedy je [22 + 3, 23] = [25, 23] a všechny mezifáze jsou zapsány odpovědně.**

Vlastní implementaci těchto praktik v rámci tohoto projektu popisuji v diagramu aktivit 3.6 v kapitole *Pohyb hráče* 3.4.7.[6][3]

2. Jak funguje hra

Cílem projektu bylo vytvořit hru pro více hráčů s propojením hráčů skrze internetovou síť. Hra spočívá v tom, že se hráč objeví v aréně a jeho cílem je zůstat poslední naživu. Kola se opakují, dokud jeden z hráčů není úplným vítězem. Mezi koly dostává hráč možnost vylepšovat svoje vybavení užitím rougelike prvků. Čím je toto řešení originální od odstatních podobných her je fakt, že hlavní zbraní se stává klacek, který si hráči musejí sami nalézt a následně vylepšovat.

2.1 Hráčské vstupy

Vstup	Funkce
W	Chůze vpřed
A	Chůze vlevo
S	Chůze vzad
D	Chůze vpravo
Mezerník	Skok
E	Sebrání itemu
Q	Zahození itemu
Levé tlačítko myši	Střelba

2.2 Průběh hry

Začátek hry

Na začátku hry se připojení hráči objeví v aréně bez jakéhokoliv vybavení. V tento moment začíná odpočet. Dokud neskončí odpočet, hráč nemůže opustit svoji pozici. Hráč má čas pro zmapování prostředí a zamyšlení se nad strategií pro začátek kola.

Po uplynutém odpočtu souboj začíná a hráč je propuštěn. Cílem kola je nalézt nejlepší funkční zbraň (klacek), tím získat výhodu do dalších kol a zůstat jakožto poslední živý hráč.

V momentě, kdy je pouze 1 hráč naživu, je mu navýšen počet vyhraných kol o 1 (hráč potřebuje 3 vyhraná kola pro celkové vítězství ve hře) a kolo se ukončuje. Hráči se přemísťují do vylepšovacího kola.

Vylepšovací kolo

Vylepšovací kolo začíná výběrem ze 3 karet vylepšení.

Po výběru jsou hráčům představeny jejich inventáře s vlastněnými vylepšeními zbraně (klacku). Zde se jim zobrazuje zbraň (klacek), kterou našli v prvním kole (pokud žádnou zbraň (klacek) nenalezli, je jim přidělena nejzákladnější varianta zbraně (klacku)). Hráč v této fázi upravuje svoji zbraň (klacek) přetahováním nově získaných vylepšení na zbraň (klacek).

Po odsouhlasení všech hráčů k postupu do dalšího kola jsou hráči opět přesměrováni do arény a začíná běžné soubojové kolo.

Běžné soubojové kolo

Probíhá stejně jako první kolo hry. Rozdílem je, že se nyní již po mapě neobjevují sbíratelné zbraně (klacky). Hráč se totiž objeví již vybaven zbraní (klackem) z jeho inventáře.

Konec hry

Hra končí, když jeden z hráčů dosáhne 3 vyhraných kol.

2.3 Princip fungování klacku

Hlavní zbraní hry je klacek. Každý hráč může mít právě 1 zbraň (klacek). Vlastnosti zbraně (klacku) následně může upravovat pomocí vylepšení, která získává mezi soubojovými koly hry.

2.3.1 Střelný tok klacku

Střelný tok zbraně (klacku) je veličina vytvořená pro zjednodušení abstraktního popisu fungování zbraně (klacku) při aplikování vylepšení.

2.3.2 Vylepšení klacku

Vylepšení zbraně (klacku) se dělí na 2 typy:

- **Rozdělovače**: Funkcí rozdělovačů je dělit (větvit) střelný tok zbraně (klacku) z jednoho na dva nebo více dalších. Rozdělovače nijak neupravují hodnotu střelného toku zbraně (klacku).
- **Vylepšovače**: Funkcí vylepšovačů je upravit hodnotu střelného toku zbraně (klacku). Vylepšovače nijak nedělí (nevětví) střelný tok zbraně (klacku).

2.3.3 Seznam obsažených vylepšení

- **Ohnivé vylepšení**
- **4-fázové nabíjecí vylepšení**

- **Šiškové vylepšení**
- **Dvojrozdělovač**

2.3.4 Seznam typů hlavňové střelby a jak jich dosáhnout

- **Kulka**: Střela, které ihned zasahuje zaměřený cíl, základní forma střelby (*žádné vylepšení*)
- **Zápalná kulka**: Střela, která při zasažení zapálí zasaženého hráče (*ohnivé vylepšení*)
- **Šišky**: Projektil, ubírá život při přímém zásahu (*šiškové vylepšení*)
- **Výbušné šišky**: Projektil, při kolizi vybuchuje (*šiškové vylepšení* + *zápalné vylepšení*)

Projektil = střela, která letí pomalu a má spád

3. Implementace

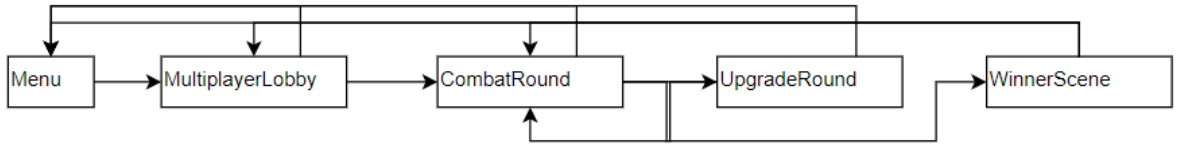
V této části popisuji jednotlivé herní mechaniky a systémy do detailu, ne pro běžného hráče. Projekt byl vytvářen v herním enginu Unity, proto se v následující části vyskytují výrazy a pojmy specifické pro toto vývojové prostředí.

3.1 Použité nástroje a služby

Unity Engine, C#, Unity Netcode, Unity Relay

3.2 Struktura

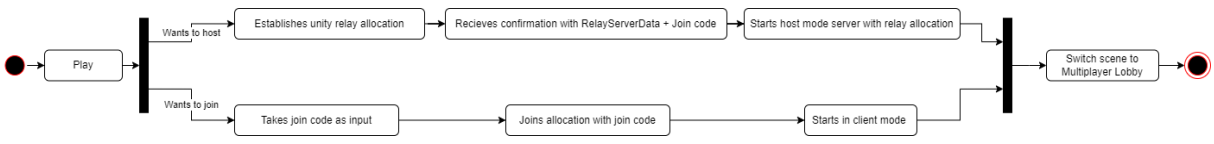
Projekt se pohybuje v rozsahu 5 scén.



Obrázek 3.1: Scenes

První scénou je scéna **Menu**. Ta slouží jako hlavní a první scéna, se kterou se hráč setká. Zde si hráč volí, zda-li chce server hostovat, nebo se k serveru připojit. Po zvolení volby je vytvořený NetworkManager viz diagram 3.3. Při hostování se vytvoří server a relay alokace zavoláním UnityRelay api. Při volbě join je klient napojen na relay alokaci skrze kód získaný od hostujícího klienta. Pokud jakákoliv akce selže, klient je vrácen zpět do menu. Pokud je vše vykonáno úspěšně, klient je přesměrován na scénu MultiplayerLobby.

Ostatní scény spadají do sekce multiplayer scén, které nefungují, pokud nejsou načteny v rámci správného napojení na server a není provedena validní alokace skrze Unity Relay.



Obrázek 3.2: Diagram aktivit procesu hostování/ napojení se na server

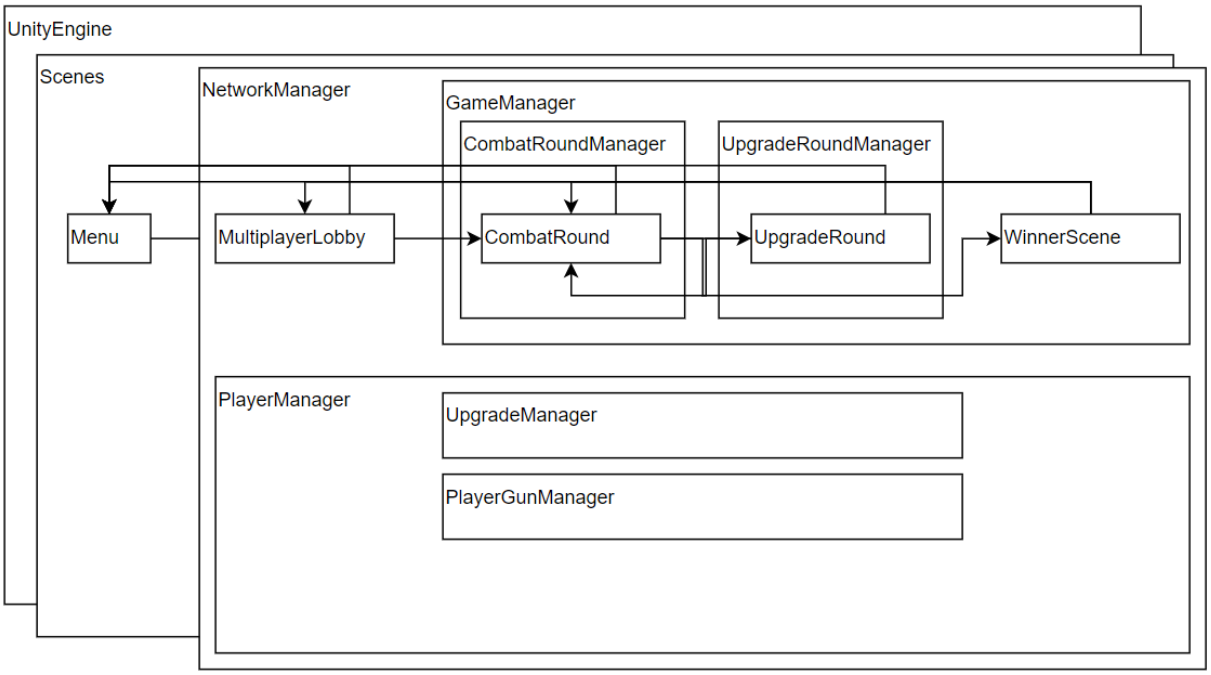
Druhou scénou je scéna **MultiplayerLobby**. Tato scéna slouží pro umožnění ostatním hráčům se připojit do session před započetím hry. Za každého připojeného hráče je vytvořena právě 1 instance PlayerManageru, která zůstává po celou dobu existence objektu NetworkManager. Po splnění všech souhlasů se započetím hry v ReadyButton komponentě (využívá NetworkSuccessBtn) host vyvolá u všech klientů přesměrování na scénu CombatRound. Tím se spouští hra a je vytvořená komponenta GameManager, která spravuje chod průběhu hry. Opět vyobrazeno v diagramu 3.3. Na stejné úrovni také pracuje vrstva UpgradeManager a PlayerGunManager pod každou 1 instancí PlayerManageru.

Třetí scénou je scéna **CombatRound**. Scéna slouží k souboji hráčů. Na začátku je za každý PlayerManager instancován a spawnut právě jeden PlayerObject, který řeší jednotlivé hráčské mechaniky v rozsahu 1 hráčského kola. O samotném hráčském objektu a jeho nespočtu funkcí se dočtete více v kapitole 3.4 Postava hráče. Po dokončení herního kola je rozpoznáno, zda-li některý z hráčů je ve vítězné pozici. Pokud ano, jsou klienti hostem přesměrováni do scény WinerScene, pokud tomu tak není, jsou přemístěni do scény UpgradeRound.

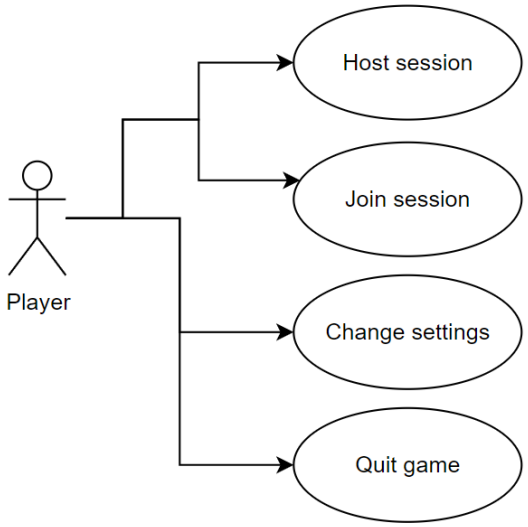
Čtvrtou scénou je scéna **UpgradeRound**. Scéna slouží k úpravě a aplikování vlastněných vylepšení zbraně (klacku). Scéna operuje ve 2 fázích. V první fázi si klient vybírá nové vylepšení zbraně (klacku). V druhé fázi již probíhá zmíněná úprava. Po odsouhlasení všech hráčů o připravenosti jsou klienti přemístěni zpět do CombatRound scény.

Pátou scénou je scéna **WinnerScene**. Tato scéna slouží pro zobrazení vítězného žebříčku po skončení hry.

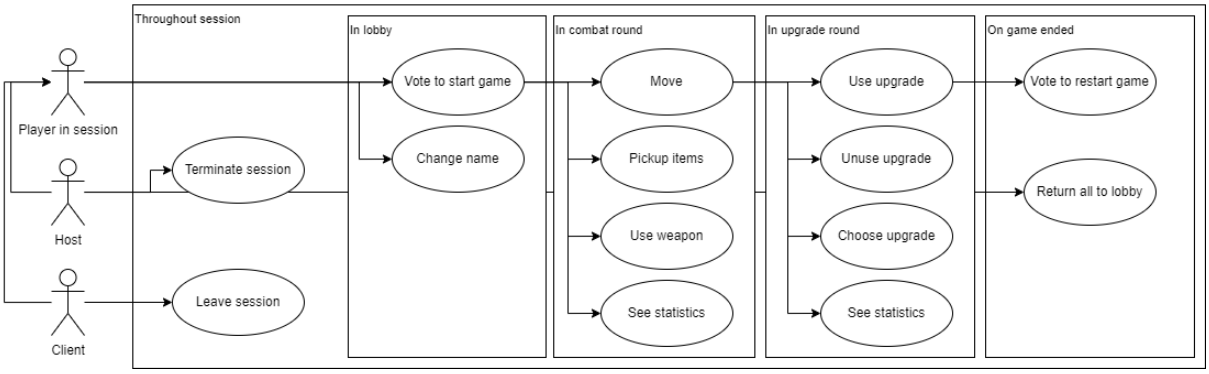
Jednotlivé funkce z pohledu uživatele jsou popsány v UseCase diagramu viz. příloha 3.4 a 3.5.



Obrázek 3.3: Scenes and managers



Obrázek 3.4: UseCase základní diagram



Obrázek 3.5: UseCase diagram v rámci herní session

3.3 Manažer hráče - PlayerManager

Vytváří se s každým připojeným klientem. Ovládá a odkazuje se na objekty a data spjaté se zastupujícím klientem.
Obsahuje 3 spravující třídy:

- PlayerManager
- UpgradeManager
- PlayerGunManager

3.3.1 Třída PlayerManager

Dědí z třídy: NetworkBehaviour[9]
Účel: Slouží jakožto správce klienta, vytváří a ničí hráčský objekt.

Veřejné proměnné:

Název proměnné	datový typ	funkce
PlayerName	NetworkVariable[11]<FixedString32Bytes>	Drží hodnotu jména hráče.
PlayerObject	Player	Odkazuje na třídu v instanci hráčského objektu.

Veřejné metody:

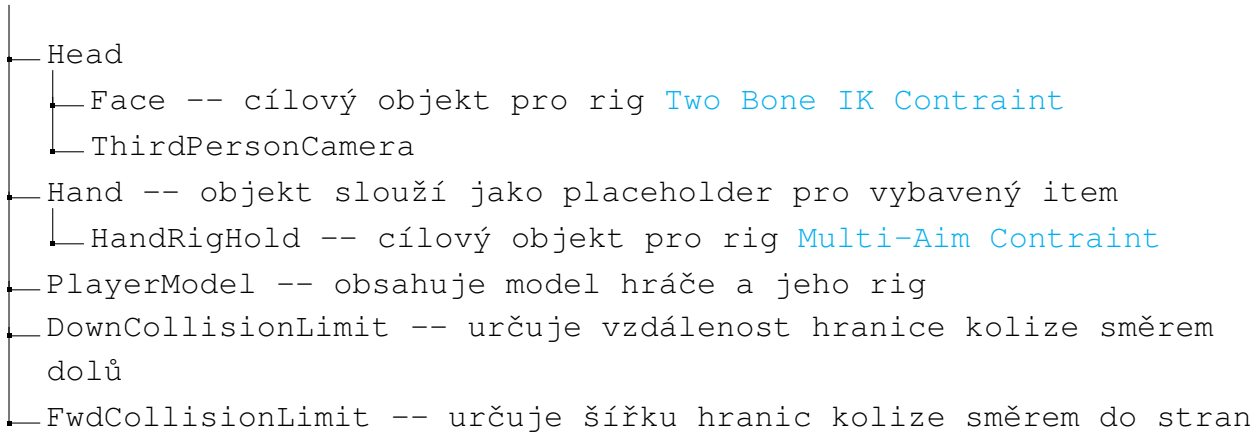
Název metody	argumenty	funkce
SpawnPlayerObject	Vector3 spawnPosition Quaternion rotation = new() bool areControlsDisabled = true bool withCamera = true	Instancuje a spawne hráčský objekt.
DespawnPlayerObject	–	Zničí hráčský objekt napříč všechna připojení.

3.4 Herní objekt hráče - PlayerObject

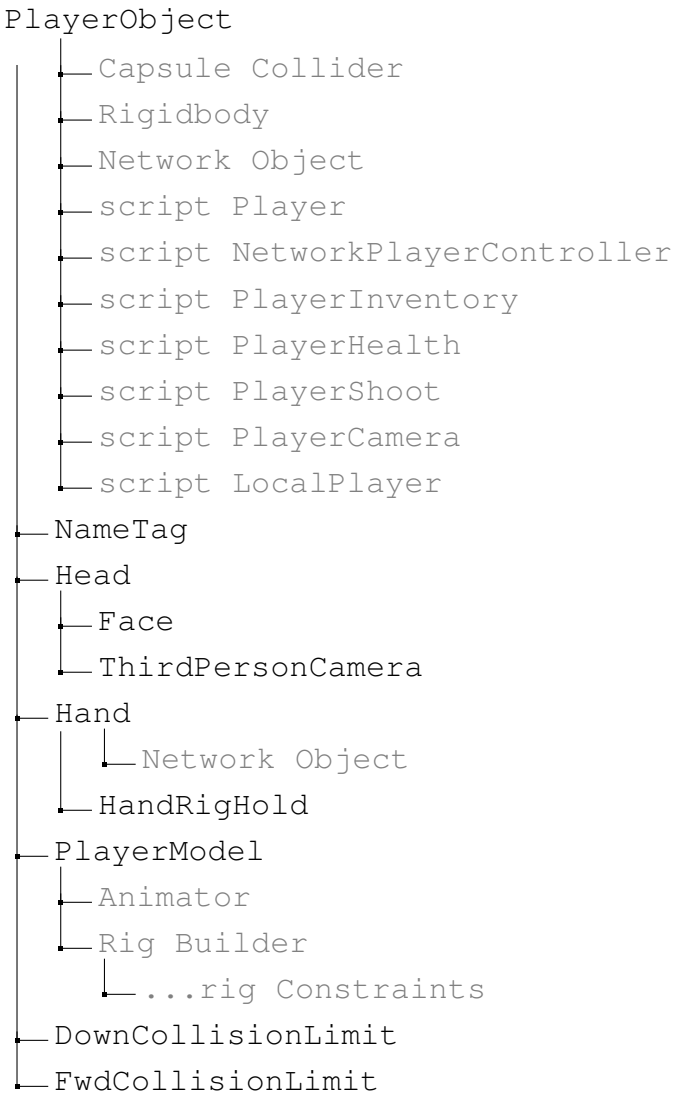
Herní objekt hráče je nejkomplexnějším objektem celého projektu. Obsahuje 7 sobě specifických tříd, které přidávají vrsty funkcí a navzájem se doplňují.

Struktura objektu:

```
PlayerObject
└─NameTag -- ukazatel jména hráče
```



Struktura objektu s komponenty:



Jak je to s kolizemi?

Jak lze ze struktury poznat, hráčský objekt řeší kolize dvěma způsoby. Proč? Protože pro ri-

gidbody vlastnosti využívá systém fyziky poskytnutý od Unity, ale tento systém není dostatečně spolehlivý v rámci předpovídatelnosti pohybu hráče mezi zařízeními. Proto samotný vstup od hráče propisuje skrze vlastní implementaci kontroly kolizí. Více k zpracování pohybu v kapitole 3.4.7 Pohyb Hráče.

3.4.1 Třída Player

Dědí z třídy: NetworkBehaviour[9]
Účel: Slouží jako základní třída hráčského objektu a řeší například jeho inicializaci, či ovládá funkčnost jiných systémů postavy.

3.4.2 Třída PlayerInventroy

Dědí z třídy: NetworkBehaviour[9]
Účel: Zařizuje správnou funkci pro sbírání a odhazování itemů.

3.4.3 Třída PlayerHealth

Dědí z třídy: NetworkBehaviour[9]
Účel: Spravuje systém zdraví hráčského objektu.

Veřejné proměnné:

Název	Datový typ	Popis
Health	NetworkVariable<int>	Reprezentuje hodnotu počtu životů. Pokud je tato hodnota na nule setter nastaví instanci hráče na mrtvého.

Veřejné metody:

Název	Argumenty	Popis
Damage	int amount	Odebere hodnotu z počtu životů.
Heal	int amount	Přidá hodnotu k počtu životů. Limituje celek na maximální hodnotu 100.

3.4.4 Třída PlayerCamera

Dědí z třídy: NetworkBehaviour[9]
Účel: Vytváří a spravuje instanci hráčské kamery a služby spojené s ní.

3.4.5 Třída PlayerShoot

Dědí z třídy: MonoBehaviour
Účel: Pokud je hráč lokální, zaznamenává vstup pro střelbu od uživatele a spouští veřejnou akci ShootInput.

Veřejné proměnné:

Název	Datový typ	Popis
ShootInput	Action<bool>	Vyvolá se, když hráč vyvolává vstup střelby. Bool hodnota udává zda-li se jedná o započetí stisknutí vstupu a nebo je vstup stisknutý déle (užitečné například pro určení semi-automatická vs automatická střelba).

3.4.6 Třída LocalPlayer

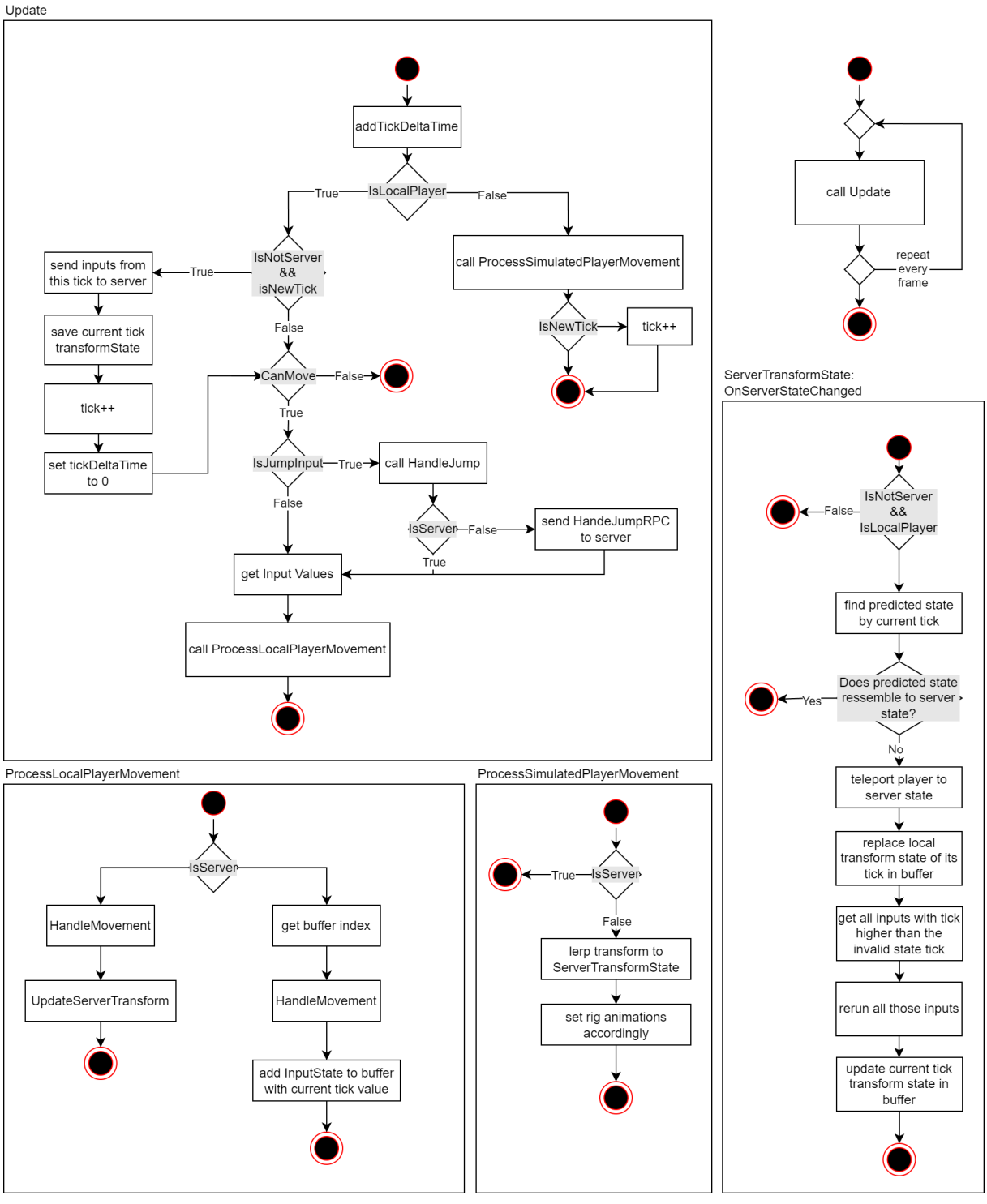
Dědí z třídy: MonoBehaviour
Účel: Je aktivní pouze pokud je postava lokálním hráčským objektem. Je určena k lokálně specifickým funkcím.

3.4.7 Pohyb hráče

K pohybu hráče slouží třída NetworkPlayerController, která zajišťuje synchronizaci pohybu skrze připojené klienty. Třídá obsahuje také samotnou funkci pro zpracování pohybu dle vstupů, nazývá se HandleMovement, která je více popsána v kapitole 3.4.7.1.

Pohyb hráče je třeba synchronizovat mezi serverem a klientem. To na serveru není problém, u klienta nám ale vznikají komplikace. Cílem je dosáhnout responzivnosti na straně klienta, ale zároveň dodržet autoritu serveru. V implementaci aplikuji praktiky předpovídajícího klienta a serverové rekonciliace. Proč a jakým způsobem tyto praktiky fungují, popisuji v kapitolách 1.3.1 a 1.3.2. Samotná implementace v tomto projektu je rozšířena navíc o seskupování více vstupů v rámci jednoho herního ticku za účelem snížení frekventovanosti posílání packetů dat mezi klientem a serverem.

Celý postup za zpracováním pohybu je popsán v diagramu aktivit 3.6.



Obrázek 3.6: Diagram aktivit - NetworkPlayerController

3.4.7.1 Metoda HandleMovement

Parametry:

typ	název
Vector3	movementInput
Vector3	rotationInput
float	deltaMultiplier

Zpracování pohybového vstupu

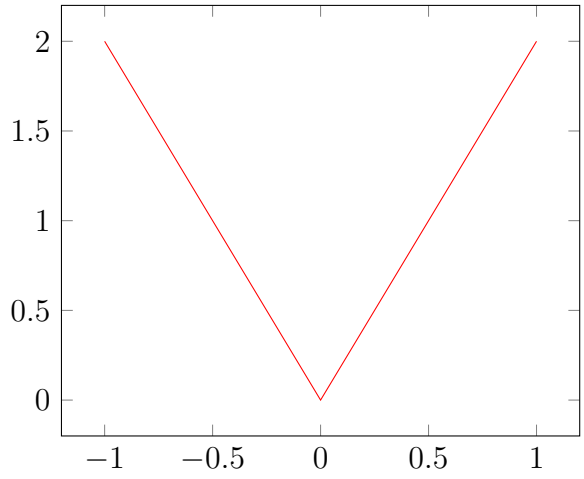
Pro optimalizaci je vykonáváno, pouze pokud není `movementInput` hodnota nulovým vektorem.

1. Nejdříve vypočítáme hodnotu pohybového vektoru vynásobením `movementInput` s `deltaMultiplier` (s hodnotou se pracuje jako s lokální hodnotou vůči hráčskému objektu).
2. Vypočteme offsetový vektor se směrem pohybového vektoru s absolutní velikostí dle nastaveného poloměru šířky kolize herní postavy.
3. Přidáme offsetový vektor k pohybovému vektoru.
4. Vyšleme raycast ze středu hráčského objektu ve směru pohybového vektoru s jeho absolutní velikostí. Pokud dojde ke kolizi, snížíme velikost pohybového vektoru na velikost dle vzdálenosti zásahu.
5. Tím úspěšně získáváme reálnou možnou vzdálenost pohybu hráče vůči kolizím (stále s přidaným offsetem dle šířky hráče).
6. Následně z nové pozice vyšleme raycast směrem dolů v určené vzdálenosti dle bodu dolní podstavy hráče pro nastavení správného umístění hráčské postavy ve výšce. Pokud dochází ke kolizi, posuneme cílový pohybový vektor o chybějící výškový rozdíl nahoru.
7. Poté odečteme offsetový vektor od pohybového vektoru a skrze funkci *transform.Translate* posuneme postavu hráče na novou pozici.
8. Na úplný konec zkontrolujeme, jestli není pohyb hráče až moc vysoký. Pokud ano, tak pohybovou akci zrušíme. Kontrola je prováděna porovnáním, zda-li dvojitá vzdálenost posunutí nahoru je větší než-li vzdálenost pohybu do strany.

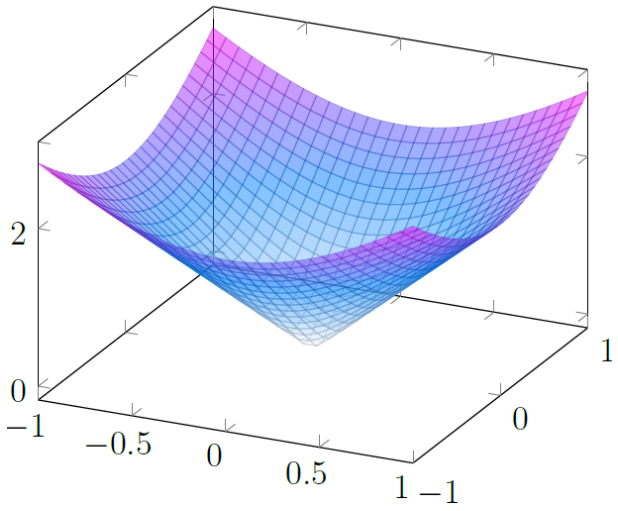
$$0.5 * vzdálenost\ nahoru > vzdálenost\ do\ strany$$

$$0.5y > \sqrt{x^2 + z^2}$$

Vizualizováno v grafu zjednodušeného průřezu 3.7 a v trojrozměrném grafu 3.8.



Obrázek 3.7: Průřez vizualizace omezení pohybu



Obrázek 3.8: Vizualizace omezení pohybu

Zpracování rotačního vstupu

Pro optimalizaci je vykonáváno, pouze pokud není `rotationInput` hodnota nulovým vektorem.

1. Otočíme celý hráčský objekt o hodnotu X vynásobenou `deltaMultiplier` kolem osy X. Rozsah je omezen na -90° až 90°.

2. Otočíme hlavu a ruku hráčského objektu o hodnotu Y vynásobenou `deltaMultiplier` kolem osy Y.

3.4.7.2 Struct TransformState

TransformState je struktura, která nám slouží k zapsání aktuální pozice hráče. Je přizpůsobená pro rozhraní INetworkSerializable[8].

Hodnoty:

Typ	Název	Popis
public int	Tick	Zapisuje kdy udaný stav nastává.
public Vector3	Position	Celková pozice objektu hráčské postavy.
public Quaternion	Rotation	Celková rotace objektu hráčské postavy.
public Quaternion	Facing	Rotace pohledu hráčské postavy (může se lišit od celkové rotace).

3.5 Stavba klacku

Zbraň (klacek) se skládá ze 3 typů komponent:

Název	C# třída	Popis
Zdroj klacku	GBase	Vytváří střelný tok. Má 1 ukončení na které lze umístít **vylepšení**, či **hlaveň**.
Vylepšení klacku	GUpgrade	Umisťuje se mezi **zdroj** a **hlaveň**. Upravuje střelný tok.
Hlaveň klacku	GMuzzle	Generuje se na konci střelného toku. Při aktivaci vyvolá střelbu. Střelbu si musí definovat každý typ hlavně zvlášť.

Stavba zbraně (klacku) je implementována dvěma způsoby – v lokálním prostředí a ve sdíleném prostředí (lokálním prostředím je například scéna vylepšování zbraně (klacku), kdy není potřeba, aby byl objekt rozpoznán mezi propojeními, a ve sdíleném je například scéna souboje, kdy ten samý objekt je potřeba zobrazovat synchronizovaně u více hráčů najednou).

3.5.1 Herní objekt zdroje - GBase

Struktura objektu:

```
GBase
├─ Point -- určuje místo pro umístění child komponenty
└─ Model
```

Využití objektu

Objekt využíváme v případě generace zbraně jako zdrojovou, rodičovskou komponentu. Následně nám zprostředkovává funkce pro ovládání samotné fuknčnosti zbraně.

$$\textit{Tento objekt} \rightarrow \frac{\textit{Gmuzzle}}{\textit{GUpgrade}}$$

3.5.2 Herní objekt vylepšení - GUpgrade

Struktura objektu:

```
GUpgrade
└─ GPartStencil
   ├─ DestinyPoints -- určuje místa pro umístění child komponent
   │  └─ ...Points
   └─ Model
```

Struktura objektu s komponenty:

```
GUpgrade
└─ GPartStencil
   │  └─ class GUpgrade
   │     ├─ UpgradeId -- Id číslo pro Upgrade definované v třídě
   │     │  UpgradeManager
   │     ├─ TotalOutline -- odkaz na Outline komponentu
   │     ├─ PartOutline -- odkaz na Outline modelu komponentu
   │     └─ Destiny[] -- odkaz na DestinyPoints
   ├─ DestinyPoints -- určuje místa pro umístění child komponent
   │  └─ ...Points
   └─ Model
```

Využití objektu

Objekt využíváme v případě generace zbraně jako navazující komponentu. Komponenta je umístěna na jakoukoliv Point komponentu spadající pod GBase nebo GUpgrade, následně přesměrována a upravuje volání funkcí z rodičovského objektu do child objektu (child objektem může být Gmuzzle nebo GUpgrade).

$$\frac{\text{GBase}}{\text{GUpgrade}} \rightarrow Tentoobjekt \rightarrow \frac{\textit{Gmuzzle}}{\textit{GUpgrade}}$$

3.5.3 Herní objekt hlavně - GMuzzle

Struktura objektu:

```
GMuzzle
├─ Point -- určuje místo pro umístění child komponenty
└─ Model
```

$$\frac{\text{GBase}}{\text{GUpgrade}} \rightarrow Tentoobjekt$$

3.5.4 Herní objekt Point

Struktura objektu:

`Point`

Slouží jako propojující bod mezi komponenty zbraně. Obsahuje odkaz na rodičovskou GBase nebo GUpgrade a odkaz na child objekt GMuzzle nebo GUpgrade.

3.5.5 Šablona herního objektu GPartStencil

```
GPartStencil
├─DestinyPoints -- určuje místa pro umístění child komponent
│ └─...Points
└─Model
```

3.5.6 Abstraktní třída GPart

Dědí z třídy: NetworkBehaviour[9]
Účel: Slouží jakožto základ pro všechny třídy tvořící součásti zbraně (klacku).
Dědí z ní třídy:

- GBase
- GUpgrade
- GMuzzle

Metody:

Název metody	typ	argumenty	funkce
Shoot	abstract	ShootData shootData, Player owner	Tvorba střelného toku.

3.5.7 Třída GBase

Dědí z třídy: GPart

Veřejné metody:

Název metody	typ	argumenty	funkce
Shoot	override	bool isFirstShot, Player owner	Zdrojovou funkcí střelného toku. Při vybavení do ruky hráče je její zavolání vázáno k vstupu střelby hráče.

3.5.8 Abstraktní třída GUpgrade

Dědí z třídy: GPart
Účel: Slouží jakožto základ pro třídy vylepšení

Veřejné proměnné:

Název	Datový typ	Popis
UpgradeId	ulong	ukládá Id vylepšení.
Destiny	GDestiny[]	zde se odkazují navazující vylepšení.

Veřejné metody:

Název	Typ	Argumenty	Popis
Shoot	override	ShootData shootData, Player owner	Upravuje vstupní hodnotu *ShootData* a spouští ji u svých *Destiny* komponent.
ReplacePart	-	UpgradeWithPart uwp, bool isNetwork	Vytvoří instanci zadaného vylepšení a nahradí ho za nastávající. IsNetwork učuje, jestli se jedná o lokální či síťové prostředí.

3.5.9 Abstraktní třída GMuzzle

Dědí z třídy: GPart
Účel: Slouží jakožto základ pro třídy hlavně

Veřejné proměnné:

Název	Datový typ	Popis
–	–	–

Veřejné metody:

Název	Typ	Argumenty	Popis
Shoot	override	ShootData shootData, Player owner	Přebírá vstupní *ShootData* komponentu a definuje a vyvolává funkci střelby

3.5.10 Ukládání klacku

Na každou hráčskou postavu se váže 1 manažer zbraně (klacku) *PlayerGunManager* 3.6.5 Ten má na sobě síťovou proměnnou (NetworkVariable[11]) GunBaseSaveData sloužící k univerzální formě zapsání struktury zbraně (klacku) v grafové podobě. Obsahuje metody pro spawn

zbraně (klacku) pro lokální použití, spawn zbraně (klacku) pro sdílené použití, zapsání struktury do textové podoby, přeložení z textové podoby do *GunBaseSaveData* podoby.

3.5.10.1 Třída GunBaseSaveData

Třída GunBaseSaveData slouží k zapsání struktury zbraně (klacku) do grafové podoby. Je přizpůsobená pro rozhraní INetworkSerializable[8].

Konstruktor:

argument - GBase gbase – Převezme zdroj zbraně (klacku) *GBase* a dle něj zapíše strukturu zbraně (klacku) do grafové podoby.

Ukázka grafové podoby

```
Zdroj - je vždy stejný
├─1 - Id součástky(rozdvojovač)
│ └─4 - Id součástky(zápalné vylepšení)
└─3 - Id součástky(šiškové vylepšení)
```

3.6 Systém vylepšení

Systém vylepšení má 2 vrstvy

1. Všechna vylepšení
2. Vylepšení se součástkou

Důvod tohoto řešení je prostý. Základním typem vylepšení je vylepšení, které nám pozměňuje některou vlastnost hráčské postavy. Může jí být například rychlost pohybu či síla výskoku. Po té tu máme vylepšení, které nám přímo nemění hráčské statistiky, ale přidávají nám součástku zbraně do inventáře. Takové vylepšení nazýváme ”Vylepšení se součástkou”nebo také *UpgradeWithPart*.

Jeden typ vylepšení se součástkou má v rámci implementace stejný postup přidání jakožto jiný typ vylepšení se součástkou, proto se nám vyplatí, v rámci znovupoužitelnosti kódu a jednoduššímu přidávání nových vylěpšení, použít vlastní abstraktní třídu.

Hlavní komponentou pro zapsání a správu všech vylepšení je UpgradeManager, zároveň je UpgradeManager využíván pro zapsání všech vlastněných vylepšení.

Poté je tu PlayerGunManager. V něm se zapisují jednotlivé vlastněné součástky vylepšení zbraně (klacku). Vlastněná vylepšení se nám zde zapisují v listu tříd GUpgradeData.

3.6.1 Třída UpgradeManager

Účel: Slouží jakožto manažer všech vylepšení.

Privátní statické proměnné:

Název	datový typ	Popis
_upgradeTypes	List<Upgrade>	Obsahuje instance od všech hráčských vylepšení.

Ukázka *_upgradeTypes* deklarace:

```
private static List<Upgrade> _upgradeTypes = new() {
    new UpgradeG2Splitter((int)Upgrades.G2Splitter),
    new UpgradeGChargeEnhancer((int)Upgrades.GChargeEnhancer),
    new UpgradeGConeEnhancer((int)Upgrades.GConeEnhancer),
    new UpgradeGFireEnhancer((int)Upgrades.GFireEnhancer),
};
```

Veřejné statické metody:

Název	Argumenty	Popis
GetUpgradeById	int id	Vrací instanci vylepšení podle id hodnoty z listu *_upgradeTypes*.
GetRandomUpgrade	-	Vrací náhodné vylepšení.
GetRandomSetOfNonrepeatingUpgrades	int amount	Vrátí pole neduplikátních vylepšení o velikosti *amount*.

3.6.2 Enum Upgrades

Účel: Zde se definuje jaké vylepšení má jaké Id. Je veřejné a určené pro jednoduché flexibilní užití v jiných scriptech.

```
public enum Upgrades
{
    G2Splitter = 1,
    GChargeEnhancer = 3,
    GConeEnhancer = 4,
    GFireEnhancer = 5,
    //...
}
```

3.6.3 Abstraktní třída Upgrade

Účel: Určuje základní funkce a proměnné pro kartu vylepšení.

Veřejné proměnné:

Název proměnné	Datový typ	Popis
Id	int	Id hodnota vylepšení.
Name	string	Název vylepšení.

Veřejné metody:

Název metody	Typ	Argumenty	Funkce
OnAdd	abstract	PlayerManager pm	Má definovat, jak se má zachovat při **přidání** vylepšení k hráči předanému v argumentu.
OnDelete	abstract	PlayerManager pm	Má definovat, jak se má zachovat při **odebrání** vylepšení od hráče předanému v argumentu.
InstantiateCard	–	Action<int>onSelected, int optionIdx	Vytvoří kartu vylepšení, přidá na ni onClick akci dle předané akce z argumentů, do které předává index karty opotionIdx také z argumentů.
GetImageSprite	–	–	Vrací Sprite s obrázkem vylepšení.
GetIconSprite	–	–	Vrací Sprite s ikonou vylepšení.

3.6.4 Abstraktní třída UpgradeWithPart

Dědí z třídy: Upgrade

Účel: Rozšiřuje třídu *Upgrade* o funkce spjaté se součástkou, kterou reprezentuje.

Veřejné metody:

Název metody	Typ	Argumenty	Funkce
OnAdd	override	PlayerManager pm	Při vyvolání **přidá** dané vylepšení do PlayerGunManageru hráče.
OnDelete	override	PlayerManager pm	Při vyvolání **odebere** dané vylepšení z PlayerGunManageru hráče.
InstantiatePrefab	–	–	Instancuje a vrací herní objekt vylepšení.
GetBranchCount	abstract	–	Má vracet počet zakončení pro dané vylepšení.

3.6.5 Třída PlayerGunManager

Dědí z třídy: NetworkBehaviour[9]

Účel: Slouží jakožto manažer pro zbraň (klacek) hráče.

Veřejné proměnné:

Název proměnné	Datový typ	Popis
GunCurrentData	NetworkVariable[11]<GunBaseSaveData>	Hodnota reprezentuje aktuální zbraň se všemi aplikovanými vylepšeními.
GUpgradeInv	IEnumerable<GUpgradeData>	Pouze get přístup pro získání dat ohledně vlastněných a aplikovaných vylepšení.

Veřejné metody:

Název metody	Typ	Argumenty	Funkce
FindGUpgradeDataByUpId	–	int upgradeId	Najde vlastněnou GUpgradeData hodnotu dle zadaného upgrade id.
AddGUpgrade	–	int upgradeId, bool isUsed = false	Přidá vylepšení zbraně do gun upgrade inventáře. Pokud je isUsed true, je rovnou zapsáno jako aktuálně používané.
RemoveGUpgrade	–	int upgradeId	Odebere vylepšení zbraně z gun upgrade inventáře.
UseGUpgrade	–	int upgradeId	Navýší počet použitých vylepšení zbraně dle upgrade id v gun upgrade inventáři.
UnuseGUpgrade	–	int upgradeId	Sníží počet použitých vylepšení zbraně dle upgrade id v gun upgrade inventáři.

3.6.6 Třída GUpgradeData

Účel: Ukládá data spjatá s hráčem a jedním specifickým vylepšením.

Veřejné proměnné:

Název proměnné	Datový typ	Popis
UpgradeId	int	Id hodnota vylepšení.
TotalCount	int	Celkový počet vlastněných vylepšení.
UsedCount	int	Počet aktuálně používaných vylepšení.

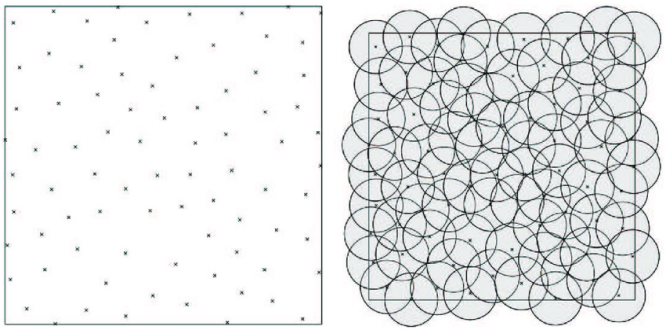
3.7 Tvorba herní mapy

3.7.1 Item spawn

Náhodné rozmístění jednotlivých itemů po členitém terénu mapy je určováno ručně tvořenou heatmapou. Intenzita barevného kanálu určuje pravděpodobnost pro objevení itemu v dané souřadnici viz obrázek 3.9.

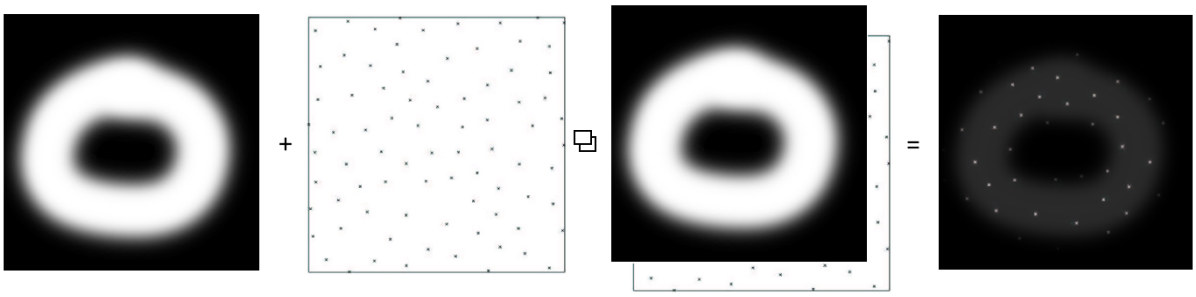
Samotnou hustotu a pravidelné rozmístění nám zprostředkovává poisson disc sampling algoritmus. Vizualizace poisson disc samplingu viz obrázek 3.10.

Obrázek 3.9: Item spawn heatmapa



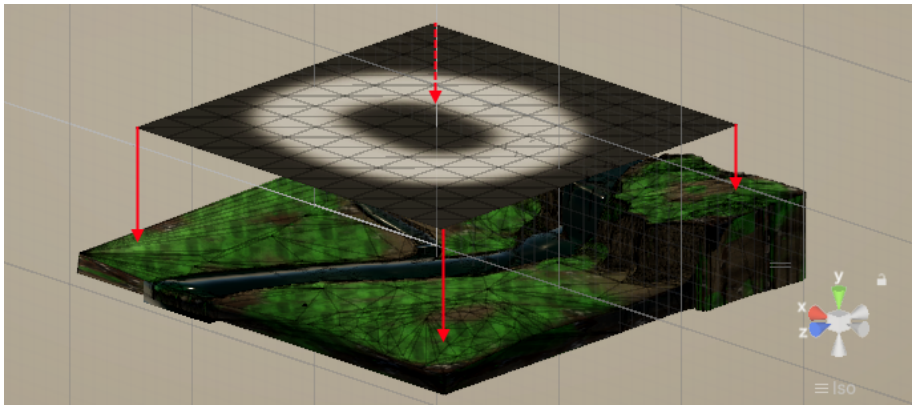
Obrázek 3.10: Ukázka jak funguje poisson disc sampling

Tyto 2 vrstvy přes sebe určují souřadnice X a Z, kde se instance objektu vytvoří. Vizualizováno v obrázku 3.11.



Obrázek 3.11: Vizualizace vrstvení map

K samotnému umístění však potřebujeme i třetí Y souřadnici. Tu potřebujeme určit takovým způsobem, aby se objekt objevil na ”povrchu” terénu. Z každého místa proto vysíláme raycast s vektorem (0,-1,0) (směr dolů v herním prostoru) a vyhledáváme kolizi s objektem ve vrstvě ”Terrain”. Vizualizováno v obrázku 3.12. Při zásahu zapíšeme získanou pozici a posuneme o 1 hodnotu na ose Y (nahoru) pro zamezení propadnutí terénem. Na výsledné pozici vytváříme instanci itemu. Itemu je nastavena náhodná rotace.



Obrázek 3.12: Vizualizace promítnutí heatmapy na terén

3.7.2 Rozmístění hráčů

Pozice rozmístění hráčů na základní mapě určuje 8 ručně umístěných objektů. Při rozmístění hráčů se tyto pozice vybírají náhodně, však unikátně. Na 1 pozici může být právě 1 hráčská postava.

3.8 Vlastní znovupoužitelné UI komponenty

3.8.1 NetworkSuccessBtn

Účel: Slouží jakožto tlačítko, které vykoná akci pouze pokud bylo zakliknuto všemi připojenými klienty.
Závislosti: Vychází z UnityEngine.UI Button[12] komponenty a závisí na NetworkObject[10] komponentě.
Přidané funkce:

Název	Typ	Popis
OnServerFullfilled	UnityEvent	Je vyvoláno pouze na serveru, když server obdrží potvrzení o připravenosti od všech klientů.
OnPreFullfilled	UnityEvent	Je vyvoláno na nehostícím klientovi v momentu, kdy počet potvrzení o připravenosti odpovídá počtu klientů. Je vyvoláno před odesláním potvrzení o připravenosti na straně tohoto klienta na server. Slouží pro zpracování iluze responzivnosti na straně klienta.

3.8.2 NetworkCoundownTxt

Účel: Slouží jakožto komponenta, která synchronizovaně zobrazuje odpočet u všech připojených klientů. Odpočet běží na serveru a jeho změny jsou odesílány ke klientům. Po dokončení odpočtu

vyvolává event.

Závislosti: Závisí na NetworkObject[10] komponentě.

Přidané funkce:

Název	Typ	Popis
_doneAction	UnityEvent	Je vyvoláno pouze na serveru když skončí odpočet.
_timeLength	int	Určuje dobu odpočtu ve vteřinách.

3.9 Slovník užitých pojmů

Instancovat - Vytvořit objekt v herní scéně z prefab objektu

Spawnout - U objektu, který byl instancován na serveru, zavolá vytvoření instance téhož objektu u všech připojených klientů s propojením na originální objekt na serveru.

4. Grafická stránka hry

Veškerý vizuál, když opomeneme animace hráče, byl vytvořen v rámci vývoje maturitní práce a proto také tvoří značnou část práce samotné.

4.1 Použité grafické nástroje

Blender, Figma, Illustrator, Mixamo

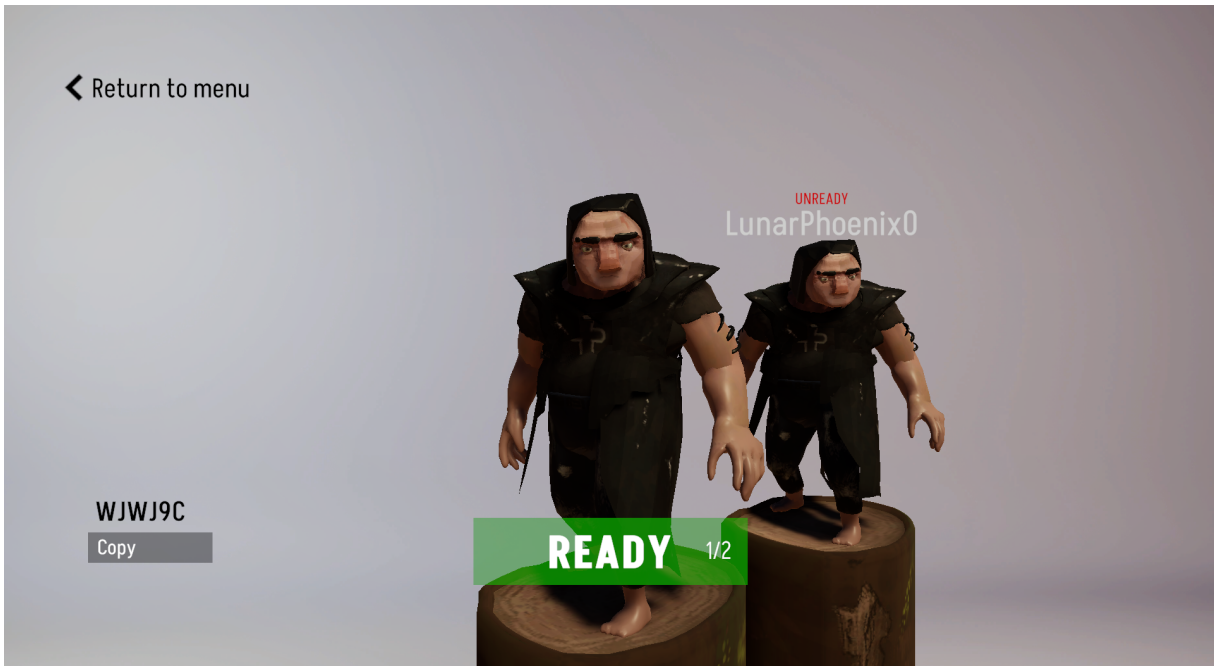
4.2 Práce:

Obrázek 4.1: Název hry

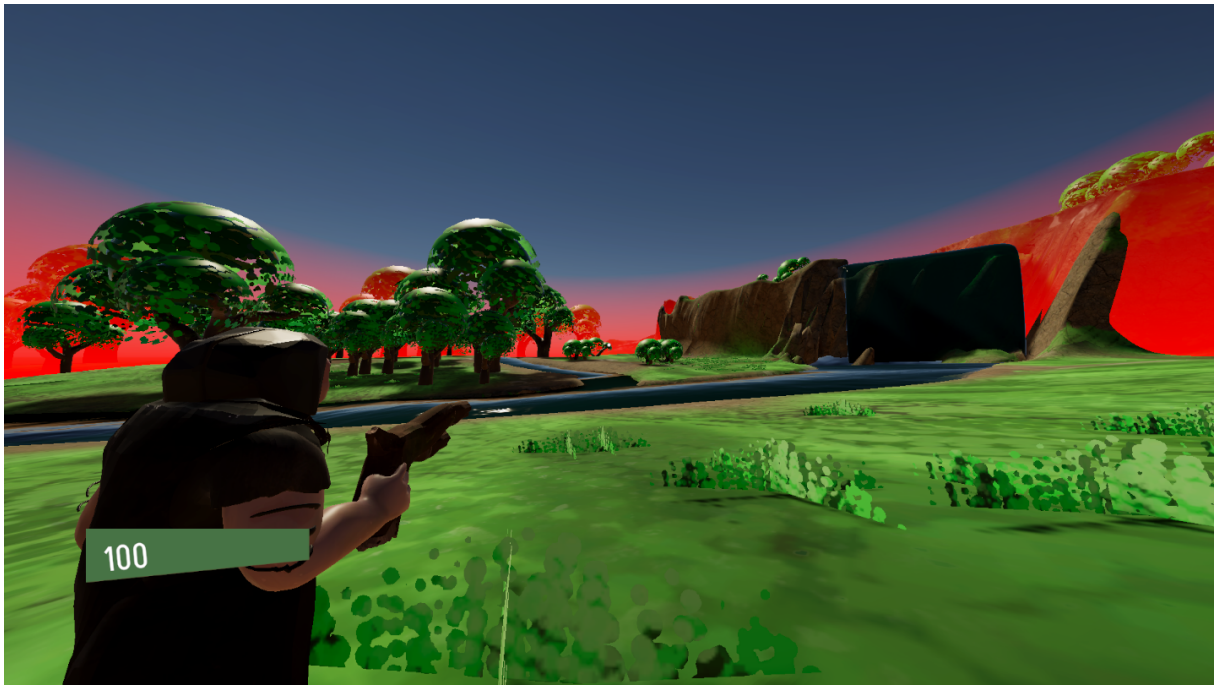
Obrázek 4.2: Ikona hry

4.2.1 Herní scény

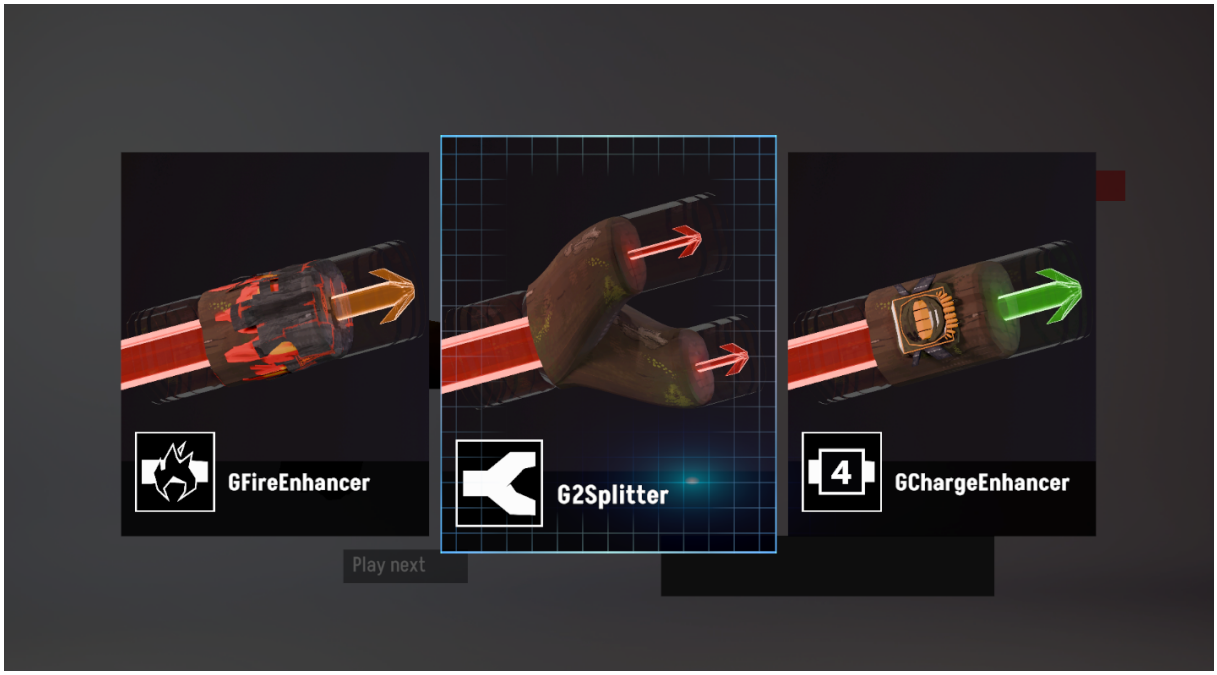
Obrázek 4.3: Herní menu



Obrázek 4.4: Multiplayer lobby



Obrázek 4.5: Soubojové kolo

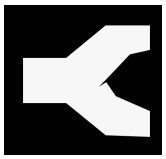


Obrázek 4.6: Vylepšovací kolo fáze 1

Obrázek 4.7: Vylepšovací kolo fáze 1

Obrázek 4.8: Výherní žebříček

4.2.2 Modely zbraně (klacku)

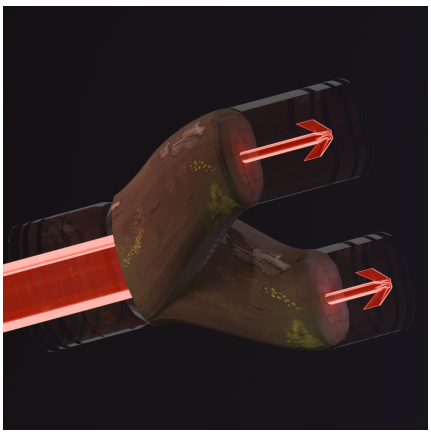


Obrázek 4.9: Ikona rozdvojovače

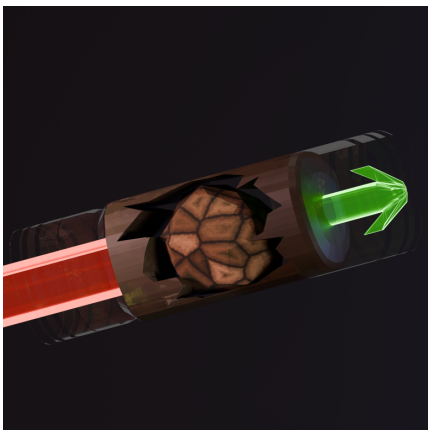
Obrázek 4.10: Ikona šiškového vylepšení

Obrázek 4.11: Ikona zápalného vylepšení

Obrázek 4.12: Ikona 4 fázového vylepšení



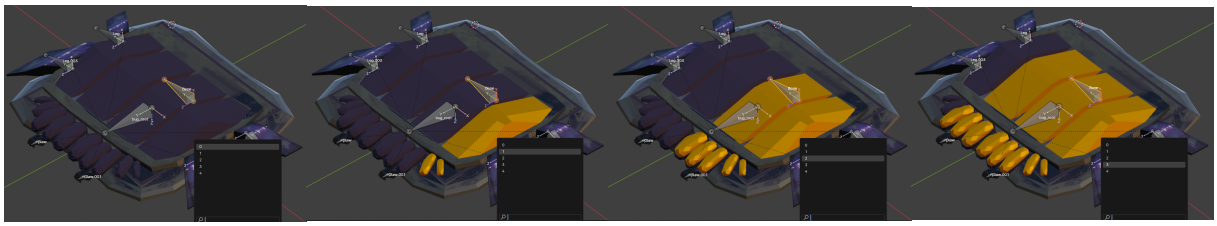
Obrázek 4.13: Render dvojrozdělovače



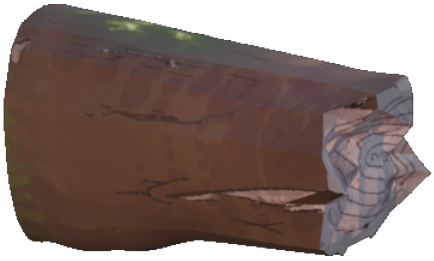
Obrázek 4.14: Render šiškového vylepšení

Obrázek 4.15: Render zápalného vylepšení

Obrázek 4.16: Render 4 fázového vylepšení



Obrázek 4.17: 4 nabíjecí fáze vylepšení

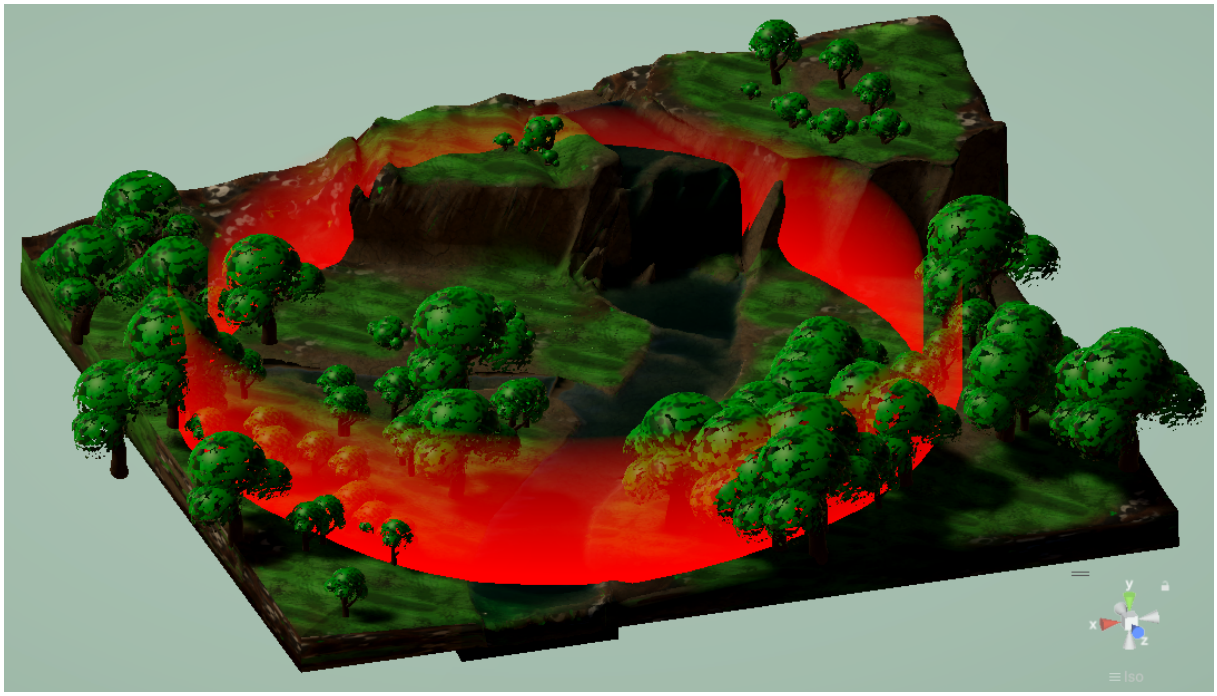


Obrázek 4.18: Základní hlaveň

Obrázek 4.19: Šišková hlaveň

Obrázek 4.20: Základ zbraně (klacku)

4.2.3 Herní mapa



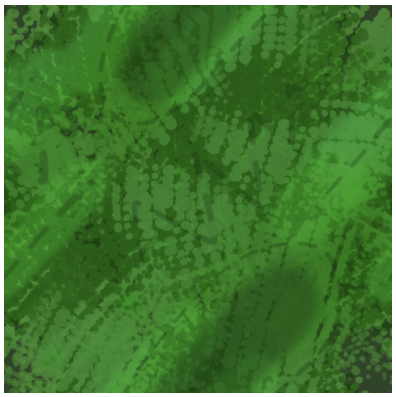
Obrázek 4.21: Základní mapa

Obrázek 4.22: Strom 1

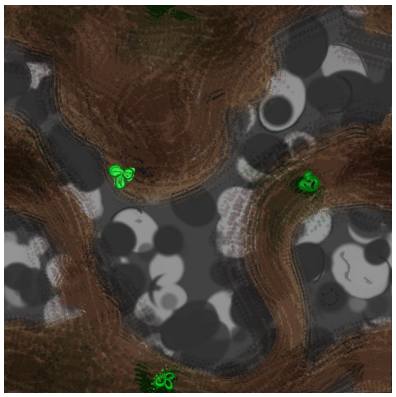
Obrázek 4.23: Strom 2

Obrázek 4.24: Křoví 1

Obrázek 4.25: Tráva 1



Obrázek 4.26: Textura tráva 1



Obrázek 4.27: Textura mix 1

4.2.4 Hráčský model - Pejvl

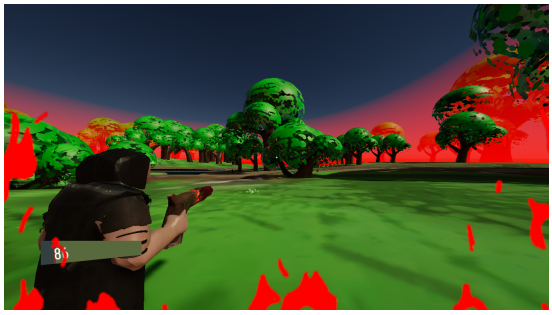
Obrázek 4.28: Pejvl

Obrázek 4.29: Model v běhu

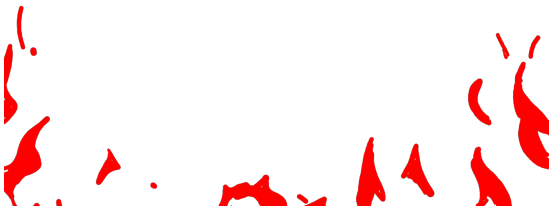
4.2.5 Jiné

Obrázek 4.30: Ikona ruky

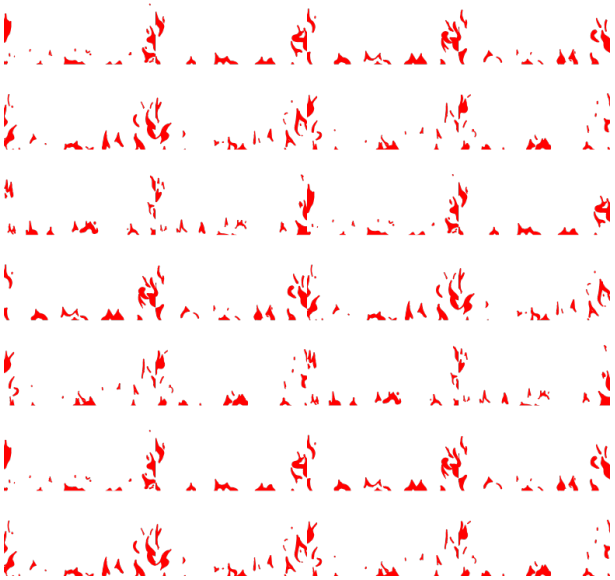
Obrázek 4.31: Ikona zrušení



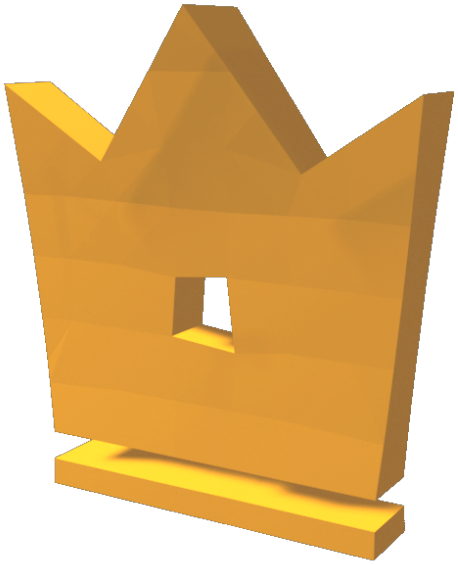
Obrázek 4.32: Pohled zapáleného hráče



Obrázek 4.33: 1 snímek - efekt ohně



Obrázek 4.34: Animace ohně



Obrázek 4.35: 3D ikona koruny

Obrázek 4.36: Borovicová šiška

Obrázek 4.37: ”Tab”okno

Obrázek 4.38: Healthbar

Obrázek 4.39: Healthbar 2

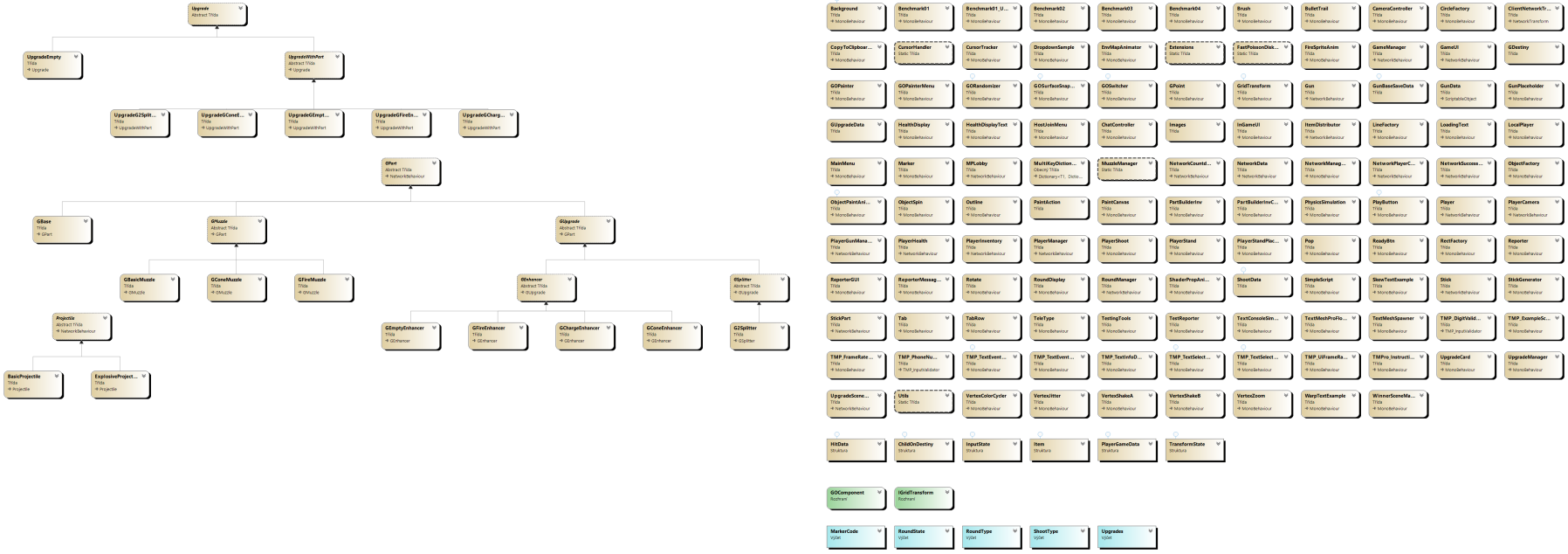
Závěr

Cílem projektu bylo vytvořit a navrhnout minimálně funkční verzi 3D multiplayerové hry Branch Brawl, což se také podařilo. Nicméně to vyžadovalo úpravy původního konceptu. Původní systém získávání itemů v průběhu kola na určitých mapách byl zrušen a nahrazen systémem výběru vylepšení a vylepšování zbraně mezi koly. Druhou změnou byl přechod z původního pohledu FPS na TPS za účelem lepší čitelnosti dění ve hře.

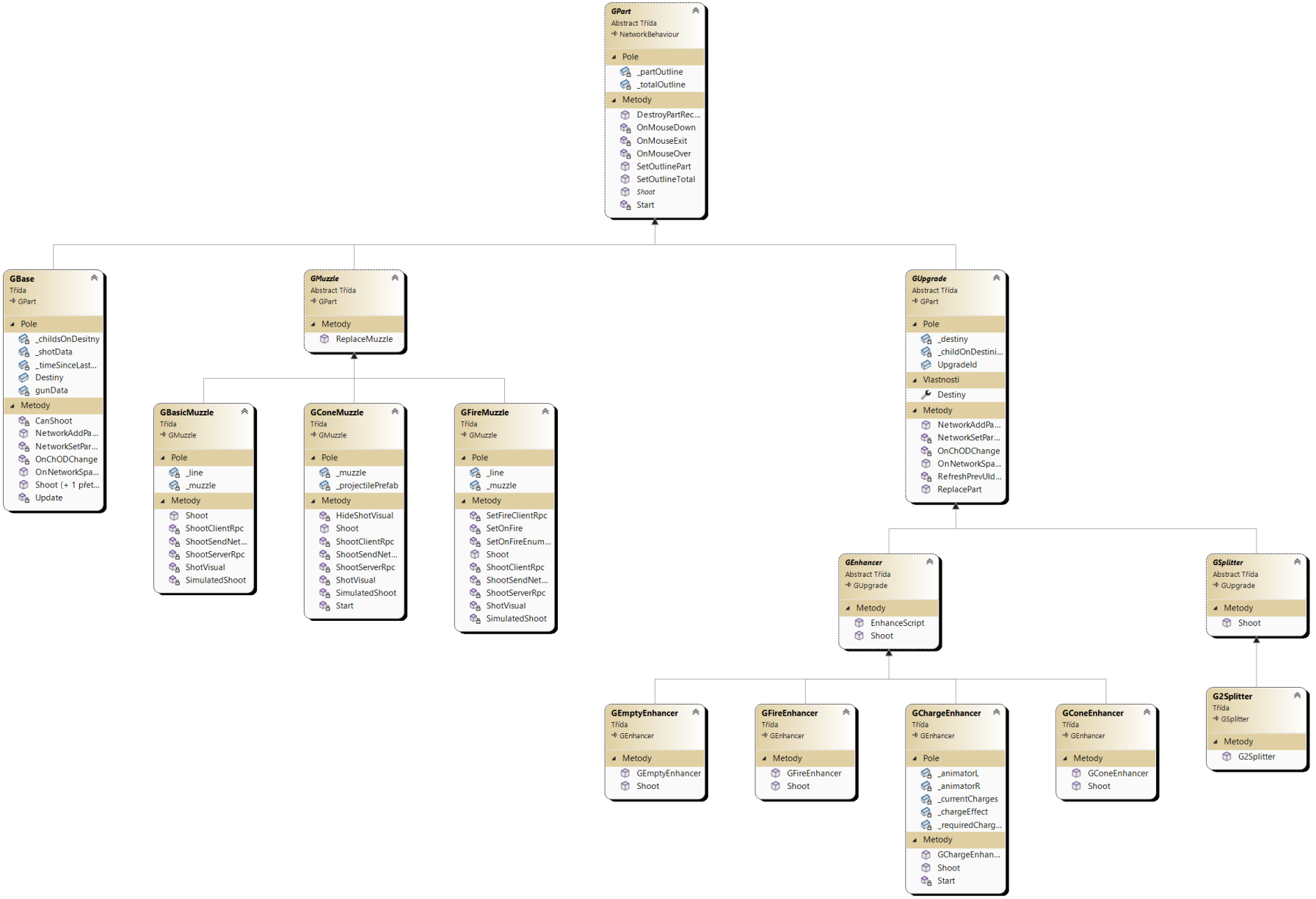
Obsahem práce bylo navržení, implementace a grafické ztvárnění první beta verze hry. Hra v současném stavu není sice finální, ale je již hratelná a poskytuje základy pro budoucí vývoj. Celkový systém zbraní (klacků) byl plně navržen a implementován tak, aby přidávání nových funkcí bylo co nejjednodušší. Aktuální stav hry je také obohacen o vlastnoručně vytvořenou grafiku, která spadá do obsahu projektové práce.

Vývoj projektu mě naučil způsobům vývoje nejen multiplayerových her, ale také mi poskytl zkušenosti s tvorbou rozsáhlých projektů od myšlenky po implementaci. Plánuji nadále rozvíjet projekt i v budoucích letech o nespočet nápadů, které jsem v tomto časovém úseku nedokázal do hry začlenit. Následně mám v plánu distribuci projektu na herní trh, včetně známé platformy Steam, po alespoň částečném dokončení hry.

Přílohy



Obrázek 4.40: Diagram všech tříd



Obrázek 4.41: Diagram tříd součástek zbraně

Obrázek 4.42: Diagram tříd vylepšení

Literatura

[1] FinancesOnline. *Number of Gamers Worldwide*. N/A. URL: https://financesonline.com/number-of-gamers-worldwide/ (cit. 23. 03. 2024).

[2] Gabriel Gambetta. *Client-Server Game Architecture*. N/A. URL: https://gabrielgambetta.com/client-server-game-architecture.html (cit. 23. 03. 2024).

[3] Gabriel Gambetta. *Client-Side Prediction and Server Reconciliation*. N/A. URL: https://gabrielgambetta.com/client-side-prediction-server-reconciliation.html (cit. 23. 03. 2024).

[4] ITnetwork.cz. *Základy - Sítě: Typy používaných sítí*. N/A. URL: https://www.itnetwork.cz/site/zaklady/site-typy-pouzivanych-siti (cit. 23. 03. 2024).

[5] Servers.com. *Differences Between Peer-to-Peer and Dedicated Game Server Hosting*. N/A. URL: https://www.servers.com/news/blog/differences-between-peer-to-peer-and-dedicated-game-server-hosting (cit. 23. 03. 2024).

[6] Unity Technologies. *Lag and Packet Loss*. N/A. URL: https://docs-multiplayer.unity3d.com/netcode/current/learn/lagandpacketloss/ (cit. 23. 03. 2024).

[7] Unity Technologies. *Network Topologies*. N/A. URL: https://docs-multiplayer.unity3d.com/netcode/current/terms-concepts/network-topologies/ (cit. 23. 03. 2024).

[8] Unity Technologies. *Unity Netcode INetworkSerializable Documentation*. N/A. URL: https://docs-multiplayer.unity3d.com/netcode/current/advanced-topics/serialization/inetworkserializable/ (cit. 23. 03. 2024).

[9] Unity Technologies. *Unity Netcode NetworkBehaviour API Documentation*. N/A. URL: https://docs.unity3d.com/Packages/com.unity.netcode.gameobjects@1.8/api/Unity.Netcode.NetworkBehaviour.html (cit. 23. 03. 2024).

[10] Unity Technologies. *Unity Netcode NetworkObject Documentation*. N/A. URL: https://docs-multiplayer.unity3d.com/netcode/current/basics/networkobject/ (cit. 23. 03. 2024).

[11] Unity Technologies. *Unity Netcode NetworkVariable Documentation*. N/A. URL: https://docs-multiplayer.unity3d.com/netcode/1.0.0/basics/networkvariable/ (cit. 23. 03. 2024).

[12] Unity Technologies. *Unity UI.Button Documentation*. 2018. URL: https://docs.unity3d.com/2018.2/Documentation/ScriptReference/UI.Button.html (cit. 23. 03. 2024).

Obrázky, Grafy a Tabulky