FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Implementace hry Inpemo
Student:	Bc. Michal Šveigr
Vedoucí:	Ing. Miroslav Balík, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Cílem práce je realizovat hru pro více hráčů, která je založena na existujícím návrhu hry Inpemo. Návrh hry Inpemo (game design document) byl vytvořen Bc. Petrem Nohejlem, Bc. Tomášem Bohuslavem, Bc. Karímem Abu Nofalem a Bc. Michalem Šveigrem v rámci předmětu Počítačové hry.

Analyzujte, navrhněte a implementujte 3D online hru pro více hráčů pro OS MS Windows vycházející z herního návrhového dokumentu hry Inpemo.

1. Prozkoumejte nástroje používané pro vývoj podobných her.
2. Prozkoumejte síťové architektury online her a diskutujte jejich výhody či nevýhody.
3. Vyberte vhodné nástroje pro realizaci a implementaci samotné hry.
4. Navrhněte architekturu hry.
5. Implementujte prototyp navržené hry a zajistěte, aby implementace využívala online služby poskytnuté Bc. Petrem Nohejlem.
6. Prototyp otestujte Vámi navrženými testy.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 14. září 2020 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Implementace hry Inpemo

Bc. Michal Šveigr

Katedra softwarového inženýrství
Vedoucí práce: Ing. Miroslav Balík, Ph.D.

6. května 2021

Poděkování

Velice chci poděkovat panu Ing. Miroslavu Balíkovi, Ph.D za to, že mi umožnil vytvořit tuto práci a také za čas a konzultace, které mi věnoval. Dále chci poděkovat panu Ing. Michalu Valentovi, Ph.D za poskytnutí licence verzovacího systému Perforce a Ing. Petru Nohejlovi za vytvoření podpůrných služeb BVision. Rovněž bych chtěl poděkovat své skvělé rodině a přítelkyni, kteří mne podporovali po celou dobu studia a bez kterých by tato práce nikdy nevznikla.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principu při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisu, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 6. května 2021 . . . . . .

Odkaz na tuto práci

Šveigr, Michal. *Implementace hry Inpemo*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Tato diplomová práce se věnuje návrhu a implementaci online hry pro více hráčů Inpemo. Hra je určena pro operační systém Microsoft Windows 10 a vychází z existujícího herního návrhového dokumentu. První část práce uvede čtenáře do problematiky vývoje online her, porovná již existující podobné hry a představí jeden z nejznámějších volně dostupných nástrojů Unreal Engine 4 sloužící k vývoji takových her. Druhá část práce představuje návrh samotné hry a implementaci prototypu včetně integrace podpůrných služeb. Poslední část práce popisuje testování výsledného prototypu.

Klíčová slova hra, multiplayer, online, herní engine, Unreal Engine, Epic Online Services

Abstract

This master thesis is about design and implementation of online multiplayer game Inpemo. The game is designed for operating system Microsoft Windows 10 and it is based on an existing game design document. Thesis introduces the issues of online game development, compare the existing similar games and present one of the most famous free tools for game development, Unreal Engine 4. The second part of the thesis is about the design of the game Inpemo and about implementation of prototype, including the integration of support services. The last part is focused on testing the result prototype.

Keywords game, multiplayer, online, game engine, Unreal Engine, Epic Online Services

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Videoherní průmysl je dnes už významnou součástí zábavního průmyslu a jeho popularita neustále roste. Pokud porovnáme jednu z prvních úspěšných her Pong, vydanou v roce 1972 firmou Atari, Inc. [2], s některou současnou populární hrou, jako je například League of Legends [3] nebo Cyberpunk 2077 [4], velice obtížně budeme hledat podobnosti. Videohry za posledních pár desetiletí urazily obrovský kus cesty, k čemuž mimo jiné přispívá i velký technologický pokrok, který v posledních letech zažíváme. Jedním z velice důležitých technologických milníků byl příchod Internetu. Ten vývojářům otevřel dveře do zcela nového videoherního odvětví dnes známého jako online multiplayer.

Situace, kdy mohou hráči z různých koutů světa hrát společně, začali vývojáři hojně využívat. Dnes již existuje mnoho herních žánrů, jako MOBA či MMORPG, pro které je online komunikace základním stavebním prvkem. Ta s sebou ovšem nese i novou úroveň komplexity a problémů, například zpoždění v komunikaci, synchronizaci herního světa nebo výpadky jednotlivých zpráv. To vývojářům přineslo spoustu netriviálních výzev, kterým je potřeba při vývoji čelit.

S rozvíjejícím se videoherním průmyslem roste i počet vývojářů a s ním i počet nástrojů a technologií, které je možné k vývoji her využívat. Vývojáři mají k dispozici různá hotová řešení, knihovny, připravené šablony či komplexní herní frameworky, které často obsahují i vlastní vývojová prostředí. Takovéto komplexní frameworky jsou známy spíš pod názvem herní enginy. Některé z nich poskytují, mimo jiné, nástroje k řešení problémů spojených s online komunikací.

Cílem této práce je představit nejdůležitější problémy vývoje online her a prozkoumat nástroje a postupy k jejich řešení. Tyto poznatky poté aplikovat na návrh a implementaci prototypu hry Inpemo, která vychází z herního návrhového dokumentu, který byl vytvořen autorem této práce Bc. Michalem Šveigrem, Ing. Petrem Nohejlem, Bc. Karímem Abu Nofalem a Bc. Tomášem Bohuslavem. Závěr práce se zabývá testováním výsledného prototypu.

Kapitola 1

Problematika vývoje multiplayer her

Jedním z důležitých faktorů, který pomohl hrám stát se tolik populárními, je právě možnost hraní více hráčů v rámci jedné hry. To přineslo nové možnosti, díky kterým hráči mohou například navzájem soutěžit či využít kooperace a bojovat společně bok po boku. Právě multiplayer hry, neboli hry pro více hráčů, patří dnes mezi ty nejvíce populární [5].

Na začátku této kapitoly budou popsány typy multiplayer her, přičemž se práce dále soustředí pouze na online hry. Následně budou vysvětleny důležité pojmy, zmíněny nejčastější problémy a představeny herní enginy, které se s tématem online multiplayeru pojí.

1.1 Typy multiplayer her

Jak už bylo naznačeno v úvodu, návrh a vývoj her pro více hráčů se může značně lišit od vývoje her pro jednoho hráče. To záleží především na tom, jaký typ multiplayeru se vývojáři rozhodnou pro danou hru zvolit. Multiplayer hry se obecně dělí do dvou kategorií.

1.1.1 Couch

První kategorií je *couch multiplayer*, překládaný jako „gauč multiplayer“. Spočívá ve hraní více hráčů na jednom zařízení. Ovládání řeší hry různými způsoby. Některé umožňují ovládat hru více hráčům na jedné klávesnici nebo připojit více gamepadů, jiné zase nastavují herní model tak, že se hráči střídají po tahu, dokončení úrovně, či pevně daném časovém úseku. Tyto hry lze ještě rozdělit do třech základních podkategorií.

Obrázek 1.1: Splitscreen ve hře Rocket League

Hotseat Pojmem *hotseat* se označují hry, ve kterých může v daný okamžik hrát pouze jeden hráč. Jeho skóre, ať už průběžné nebo celkové, se zaznamenává a po odehrání jeho časového úseku, tahu nebo pokusu pokračuje další hráč. Tímto způsobem se hráči střídají, dokud hra není ukončena. Příkladem hry využívající tento princip je *Heroes of Might and Magic*.

Samescreen V případě módu *samescreen* hrají všichni hráči paralelně a každý z nich má svůj vlastní ovladač, případně jinou periférii, pomocí které hru ovládá.

Splitscreen Tato podkategorie je téměř identická s podkategorií *samescreen*. Jedinou odlišností je rozdělení obrazovky tak, že každý hráč má pro sebe vymezenou její část. *Splitscreen* je často využíván zejména v konzolových hrách. Jednou takovou je hra *Rocket League* (viz obrázek 1.1).

Některé hry využívají také hybridního modelu *dynamic splitscreen*, kdy hra většinu času funguje jako *splitscreen* a v případě, že se hráči ve hře dostanou dostatečně blízko k sobě, hra přejde do módu *samescreen*.

1.1.2 Online

Druhá kategorie je v dnešní době tou populárnější a rozšířila se hlavně díky nástupu Internetu. Často se označuje také jako *Netplay*. Hráči nesdílejí jedno zařízení, nýbrž každý z nich hraje na svém vlastním. Jednotlivé herní akce

spolu zařízení komunikují pomocí sítě, typicky Internetu, a synchronizují tak stav hry.

Do této kategorie patří také *LAN multiplayer*, celým názvem *local area network multiplayer*. Ten se odlišuje tím, že umožňuje hrát hráčům pouze na úrovni lokální sítě, což se dnes využívá především pro různé turnaje a esportové akce. Eliminují se tím totiž běžné problémy Internetu, jako latence nebo výpadky datagramů[1].

Když se řekne slovo multiplayer, většina hráčů si ho spojí právě s *online multiplayer*, jelikož do této kategorie spadá drtivá většina dnešních nejpopulárnějších her pro více hráčů. Hra Inpemo se do ní řadí také a z tohoto důvodu se dále práce věnuje právě žánru *online multiplayer*.

1.2 Síťová komunikace a její principy

Hry jsou často označovány jako simulátory světů, ve kterých se neustále něco děje. Aby hráč měl ten správný herní zážitek a svět působil plynule, je třeba, aby mezi jednotlivými simulačními kroky byl co nejmenší možný časový rozestup. Toho lze docílit pouze vysokou aktualizační frekvencí, která se u dobře optimalizovaných her pohybuje v řádu vyšších desítek či dokonce nižších stovek aktualizací za sekundu. Pokud je do tohoto procesu začleněna i síťová komunikace, objevuje se zde zásadní problém.

V online hrách pro více hráčů má každý hráč vlastní instanci herního světa. Tento svět je nutné synchronizovat napříč všemi hráči. Ovšem rychlost i kapacita přenosu dat po síti je řádově nižší, než rychlost a kapacita přenosu mezi hardwarem uvnitř jednoho zařízení. Tento rozdíl může ještě umocnit fakt, že hráči od sebe často jsou vzdáleni desítky i stovky kilometrů, přičemž je mezi nimi mnoho síťových prvků.

1.2.1 Latence a Round trip time

Latenci lze definovat jako dobu mezi odesláním datagramu a jeho přijetím cílovým zařízením. *Round trip time*, běžně označován jako RTT, je doba, za kterou datagram urazí cestu mezi odesílatelem, cílovým zařízením a zpět. Nelze říci, že latence je rovna polovině RTT, protože doba cesty mezi odesílatelem a cílovým zařízením nemusí být stejná jako doba cesty opačné. RTT navíc také obsahuje dobu zpracování na cílovém zařízení.

Jev, kdy je latence vysoká a její dopady hráč negativně pociťuje při hraní, je označován jako „lag“. Každý lag negativně ovlivňuje hráčský zážitek, nejčastěji se projeví jako nepříjemné zpoždění mezi stisknutím klávesy a provedením příslušné akce ve hře. Schopnost hry tolerovat taková zpoždění závisí mimo jiné i na žánru hry. Obecně lze říci, že hráči hrající hry, které si zakládají na

[1] Datagram je základní jednotka přepravována v počítačové síti. Obsahuje hlavičku a tělo.

realtimové komunikaci s velkým množstvím herních akcí, například FPS[2] hry, jsou případnými lagy ovlivněni mnohem více, než hráči hrající strategické či tahové hry, u kterých je prahová hodnota latence pro citelné pocítění lagů často opravdu vysoká.

1.2.2 Protokoly

Komunikační protokol je sada pravidel, podle kterých probíhá komunikace. Určuje syntaxi a význam jednotlivých zpráv. Zařízení v počítačových sítích spolu komunikují pomocí rodiny protokolů *TCP/IP* [6].

1.2.2.1 TCP

Transmission Control Protocol umožňuje vytvoření kanálu mezi dvěma zařízeními. Data jsou přenášena ve formě proudu. Výhodou protokolu je garance doručení dat a to ve správném pořadí. V případě, že některá data nebyla doručena, jsou po určitém časovém úseku odeslána znovu. K tomu je potřeba, aby strana, která data přijímá, průběžně obdržená data potvrzovala. Dále protokol zajišťuje rozdělení dat do packetů a reguluje rychlost přenosu tak, aby přenosová linka nebyla přetížena. TCP protokol je specifikován v RFC 793 [7]. Aplikace, pro které jsou data určena, jsou rozlišovány pomocí portů. Hlavička má minimálně 20B.

1.2.2.2 UDP

User Datagram Protocol, speficikován v RFC 768 [8], na rozdíl od protokolu TCP nenavazuje spojení ani negarantuje doručení dat. Data mohou dorazit v jiném pořadí, než v jakém byla odeslána. Protokol také neřeší vytíženost linky. Všechny tyto zodpovědnosti jsou přeneseny na aplikaci. Stejně jako u TCP, aplikace jsou rozlišovány pomocí portů. Hlavička má 8B.

1.2.2.3 Vhodný protokol pro hry

Oba zmíněné protokoly jsou založeny na protokolu IP [9]. Aby bylo možné určit, který protokol je pro online hry vhodnější, je potřeba alespoň zjednodušeně pochopit, jak TCP protokol zajišťuje všechny jeho garance.

Jak bylo uvedeno, TCP protokol přenáší data proudem. Vzhledem k tomu, že IP protokol je založen na packetech a TCP je na něm postaven, musí TCP nějak rozdělit proud dat na packety. Tento mechanismus se nazývá *Naglův algoritmus* [10] a je zajištěn pomocí bufferu. Data z proudu jsou do bufferu ukládána a v momentě, kdy je jich dostatečné množství, jsou odeslána příjemci ve formě packetu.

To může být problém pro hry, které posílají velké množství malých dat. Může se tedy stát, že se TCP na základě algoritmu rozhodne neodeslat tato

[2] FPS je označení pro herní žánr *first person shooter* - střílečka z pohledu první osoby.

data do doby, než jich v bufferu bude dostatečné množství pro vytvoření přiměřeně velkého packetu. Pro spoustu herních akcí je takové chování problém, protože povahy her často vyžadují, aby se akce, například informace o uživatelských vstupech, dostávaly k ostatních hráčům co nejrychleji je to možné. Pokud jsou data s informacemi o těchto akcích zpožděna, bude mít hráč velice špatnou uživatelskou zkušenost, jelikož aktualizace budou chodit zpožděně a v malé frekvenci, namísto včas a často. Implementace TCP obvykle poskytují možnost *TCP_NODELAY*, pomocí které lze vynutit deaktivaci *Naglova algoritmu*, nicméně to obvykle nestačí.

TCP tedy v zásadě rozdělí proud dat na packety, které odešle pomocí IP protokolu a čeká na potvrzení o doručení. V případě, že potvrzení nepřijde, odesílá daný packet znovu. Na straně příjemce packety převezme a zrekonstruuje z nich proud. Duplicitní packety jsou ignorovány a ty, které přišly ve špatném pořadí, jsou správně zařazeny. Vždy, když je packet ztracen a nedoručen, musí se vše zastavit a počkat, než bude znovu poslán a úspěšně doručen. Tedy i nově doručená data nebudou předána aplikaci do té doby, než budou všechna dříve odeslaná data úspěšně doručena. V případě vypadnutí jednoho, byť nedůležitého packetu, mohou být zpožděna všechna následně odeslaná, kriticky důležitá herní data. Navíc poté, co jsou data z bufferu, kde mohou být nahromaděna za několik aktualizačních kroků, předána aplikaci, je nutné je zpracovat v rámci jednoho aktualizačního kroku, což opět přináší hráči trhaný a špatný herní zážitek.

Na základě výše zmíněných důvodů se pro online multiplayer využívá protokol UDP, nad kterým je většinou vystavěn vlastní protokol pro zaručení spolehlivosti doručení a dalších vlastností dle potřeb dané hry.

1.2.3 Topologie

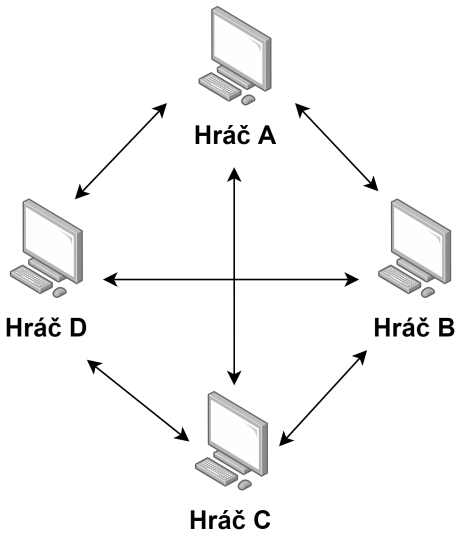
Topologie sítě určuje, jakým způsobem jsou zařízení v síti navzájem propojena. V kontextu online her popisuje, jakým způsobem jsou herní zařízení propojena tak, aby všichni hráči měli k dispozici aktuální herní stav.

1.2.3.1 Peer-to-peer

V této topologii neexistuje žádný centrální server 1.2. Každé zařízení komunikuje se všemi ostatními a má plnou kontrolu nad simulací své verze herního světa. Zároveň s každým dalším připojeným hráčem se zvyšuje spotřeba šířky pásma, jelikož každé zařízení musí navázat *n - 1* spojení, kde *n* je počet hráčů. Topologie bývá často používána ve hrách žánru *RTS*[3].

Častým principem při používání *peer-to-peer* topologie je rozdělení hry do série tahů a příkazů. Příkazy jsou na začátku každého tahu vyhodnoceny a na základě jejich významu a obsahu je prováděna simulace herního světa. K tomu

[3]Realtime strategie



Obrázek 1.2: Topologie peer-to-peer

je potřeba, aby hra byla deterministická a aby byl každý tah vyhodnocován na všech zařízeních se stejnou množinou příkazů.

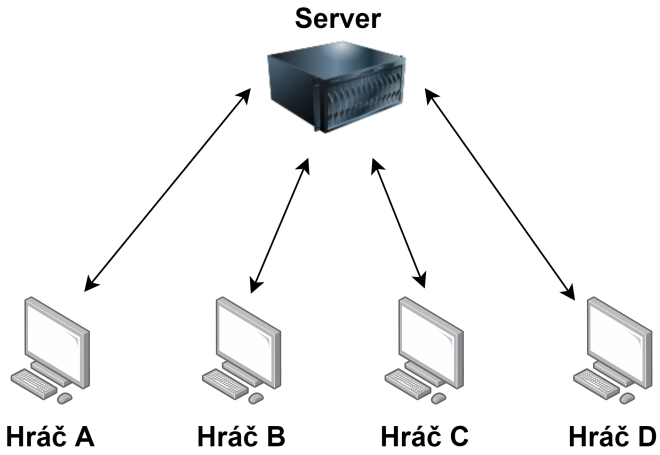
Pokud herní logika není deterministická, může nastat případ, že simulace stejné množiny příkazů bude mít u dvou hráčů jiný výsledek. Poté je nutné řešit, který z výsledků simulace je ten správný a opravit podle toho stav hry. Podobný problém může nastat, pokud dva hráči v jeden čas vykonají totožnou herní akci, například seberou stejný předmět, který může vlastnit pouze jeden hráč. Je nutné rozhodnout, kdo předmět dostane.

Aby byla zajištěna konzistence příkazů v rámci každého tahu, je nutné před začátkem vyhodnocení tahu počkat, než všichni ve hře zúčastnění zašlou příkazy, které mají být v daném tahu vyhodnoceny. To má za následek, že všichni hráči mají shodnou latenci s tím hráčem, který má latenci nejhorší.

1.2.3.2 Client-server

Hlavním znakem této topologie je přítomnost centrální jednotky - serveru (viz obrázek 1.3). Vždy existuje jedno zařízení, které je označeno jako server. Zbylá zařízení, která se označují jako klienti, jsou k serveru připojena a ten s nimi komunikuje. Klient nenavazuje spojení s ostatními klienty, může komunikovat pouze se serverem.

Tato topologie má několik výhod. Jednou z těch hlavních je menší zatížení sítě, jelikož je možné simulovat část světa pouze na serveru a zasílat klientům jen ta data, která jsou pro ně relevantní. Tím se zároveň ušetří i výkon samotných klientů, kteří nemusejí část simulací vyhodnocovat. Také to může



Obrázek 1.3: Topologie client-server

omezit případné podvádění, jako například *wall-hack*[4]. Ostatní hráči mohou dostávat informace o pozicích dalších hráčů pouze tehdy, pokud je mají vidět. Omezení podvádění lze ještě podpořit validací dat a požadavků od klientů. Konkrétně na příkladu z FPS hry může validace vypadat tak, že pokud klient oznamuje serveru začátek střelby, server může zkontrolovat, zda hráč má dostatek nábojů v zásobníku a akci mu případně odepřít.

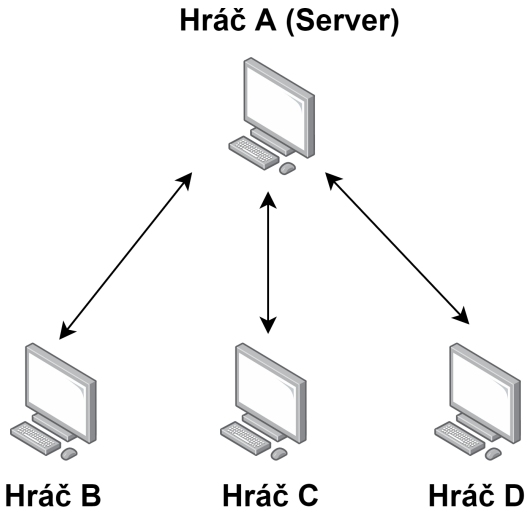
Nevýhodou jsou vysoké požadavky na výkon serveru. Zodpovědnost simulace herního světa a komunikace jeho změn je ze všech klientů přenesena na server. Rozlišují se dva základní typy serverů.

Dedicated server V tomto případě je server typicky kompletně oddělen od jakýkoliv klientských procesů. Jeho úlohou je simulace herního světa a komunikace s klienty. Nezobrazuje žádné grafické výstupy, nepřehrává zvuky ani efekty, ve většině případů se jedná o samostatnou konzolovou aplikaci.

Velkou výhodou dedikovaných serverů je jejich způsobilost k tomu, aby byly hostovány v data centrech, případně cloudech. Ta jsou většinou velmi stabilní s velkým výkonem a dostupná po celém světě. Nevýhodou je jejich finanční náročnost, díky čemuž jsou hojně využívána především herními tituly s vydatnými rozpočty. Malá studia často nemají dostatek financí, aby si hostování dedikovaných serverů mohla dovolit.

Listen server Alternativou k dedikovaným serverům jsou listen servery, v jejichž případě je server zároveň i aktivním účastníkem hry, tedy hráčem (viz obrázek 1.4). Klientské a serverové procesy nejsou odděleny, server používá stejnou aplikaci jako klienti. Před začátkem hry je nutné, aby byl vybrán jeden z hráčů, který bude serverem. Tento hráč se často označuje jako „host“. Je

[4]Druh podvodu, který umožňuje vidět ostatní hráče skrz jinak neprůhledné zdi.



Obrázek 1.4: Topologie listen-server

důležité, aby host měl výkonné zařízení včetně stabilního a rychlého připojení k síti, jelikož oproti ostatním klientům ponese navíc ještě serverovou zátěž.

Výhodou listen serveru je především cena. Hráči si hostují servery sami, tudíž vývojáři nemusejí vynakládat další prostředky, aby zajistili hosting. Výběr vhodného hosta je hlavní nevýhodou této varianty serveru. Většinou není možné bezpečně určit, kdo z hráčů bude stabilním hostem. V případě, že host opustí hru, ať z důvodu síťového problému nebo úmyslného vypnutí hry, jsou zbylí hráči odpojeni ze hry. Některé hry tento problém řeší pomocí techniky *host migration*. Pomocí této techniky je možné v rámci běžící hry vybrat nového hosta, který převezme roli aktuálního serveru a ostatní klienti se na něj připojí. Nicméně aby bylo možné tuto techniku použít, je vyžadována částečná komunikace mezi klienty. Proto se v případě listen serverů často využívá hybridní topologie, která se chová jako client-server, ale zařízení jsou ve skutečnosti propojeny pomocí peer-to-peer topologie.

1.2.4 NAT traversal

Pokud se vývojáři rozhodnou použít peer-to-peer nebo listen server topologii, musí mimo jiné řešit ještě jeden problém. Drtivá většina zařízení, na kterých hráči hrají, jsou v privátních sítích, schované za různými síťovými prvky, například routery. To zamezuje navázání spojení mezi jednotlivými hráčskými zařízeními.

NAT traversal je technika, díky které lze navázat spojení pomocí IP protokolu přes síťové prvky, které provádějí NAT, což je funkce umožnující překládat adresy z vnitřního adresního rozsahu do veřejného a naopak [11]. Tato

technika není využívána pouze ve hrách. Často je využívána různými VoIP[5] či videochatovacími aplikacemi.

Existuje několik variant této techniky a protokolů [12], které je popisují. Mezi nejznámější patří:

- **NAT hole punching** - obecná technika, každý klient musí kontaktovat server, který si dočasně uloží jeho externí adresu, interní adresu a informace o portu. Server následně poskytuje tyto informace všem klientům.
- **STUN** - protokol, který umožňuje aplikacím zjistit veřejnou IP adresu a mapování portů, které následně mohou být využity ke komunikaci.
- **TURN** - jedná se také o protokol. Spojení mezi klienty není vytvořeno přímo, pro komunikaci je využit server jako prostředník, který přeposílá zprávy mezi klienty.
- **ICE** - protokol definující kdy a jakým způsobem mají být protokoly STUN a TURN použity.

1.2.5 Synchronizace herního světa

Mít absolutně konzistentní svět napříč všemi hráči je přáním nejednoho vývojáře, bohužel to není možné. Z podstaty věci nemohou mít všichni hráči v jeden moment stejná data. Brání tomu fyzikální zákony a principy. Aby hráčský zážitek nebyl inkonzistencí světa příliš ovlivněn, používají se různé metody, jak inkonzistenci před hráčem co nejvíce skrýt.

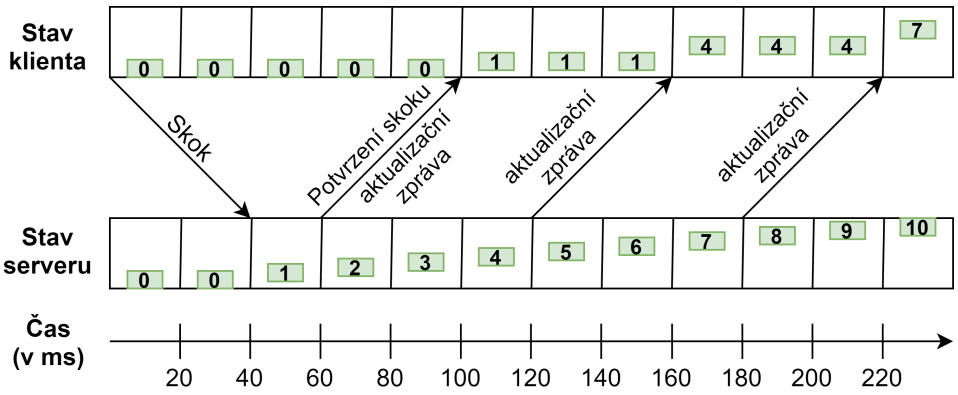
Pro snadnější pochopení budou tyto metody představeny na zjednodušeném modelovém příkladu, který uvažuje topologii client-server. Latence je 40 milisekund, aktualizace jsou zasílány každých 60 milisekund a simulační frekvence je 50 snímků za vteřinu, tedy každých 20 milisekund. Doba mezi aktualizacemi bude nadále označována jako aktualizační doba.

Na obrázku 1.5 je vyobrazena komunikace mezi serverem a klientem včetně jejich stavů. Klient v čase 0 oznámí serveru, že chce vyskočit. Server tento požadavek obdrží o 40 milisekund později kvůli latenci, provede simulaci a odešle aktualizační zprávu zpět klientovi, který ji obdrží o dalších 40 milisekund déle. Na základě těchto časů je RTT 100 milisekund. Číslo uvnitř objektu vyobrazuje vzdálenost objektu od země v daném snímku.

Modelový příklad má několik problémů, které lze částečně eliminovat. Hlavní z nich jsou:

- Skok je na straně serveru plynulý, kdežto u klienta je trhaný.
- Při výpadku aktualizační zprávy bude u klienta skok trhaný ještě více.
- Klient má zpožděnou reakci na vstup o 100 milisekund.

[5] Volání přes Internet, např. Skype



Obrázek 1.5: Modelový příklad výskoku

1.2.5.1 Interpolace

Frekvence simulace je třikrát větší oproti aktualizační frekvenci, takže klient vyhodnotí tři snímky mezi jednotlivými aktualizačními zprávami. Pokud v aktualizační zprávě obdrží novou pozici objektu, zobrazuje tuto pozici do doby než přijde další aktualizační zpráva, tedy několik snímků po sobě. Pohyb díky tomu vypadá nespojitě a trhaně. Tento problém lze částečně eliminovat pomocí interpolace a interpolační rezervy.

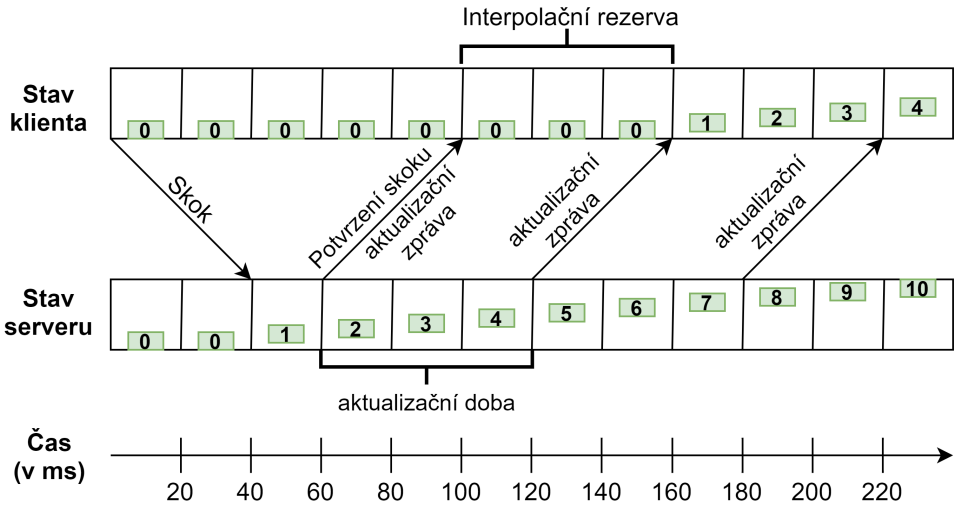
Interpolační rezerva je doba, o kterou jsou klienti navíc uměle opožděni oproti serveru. V případě, že je tato doba větší než doba mezi aktualizačními zprávami, klienti vždy ví, v jakém stavu se bude herní svět nacházet. Ideální interpolační rezerva je rovna aktualizační době.

Interpolace je metoda používaná k přibližnému určení hodnoty funkce v nějakém intervalu, přičemž jsou známy její hodnoty v jiných bodech tohoto intervalu. Interpolaci lze definovat následovně [13]: - je dána funkce $f(x)$, pro kterou jsou známy hodnoty $f(x_0)$, $f(x_1)$, ... $f(x_n)$. Interpolace znamená nalezení funkční hodnoty $f(x)$, pokud platí $x_0 < x < x_n$.

Základním typem interpolace je lineární interpolace, která je založena na proložení dvou bodů přímkou [14]. Když jsou známy hodnoty funkce $f(x_0)$ a $f(x_1)$ a platí, že $x_0 < x < x_1$, pak

$$f(x) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0} * (x - x_0).$$

Použití lineární interpolace na uvedeném modelovém příkladě je vidět na obrázku 1.6, kde je interpolační rezerva rovna aktualizační době. Klient má sice při použití této techniky pocitově větší latenci, nicméně za cenu plynulého pohybu. Pokud je interpolační rezerva rozumně nastavena, umělé zpoždění má výrazně menší vliv na herní zážitek než neplynulý pohyb, proto je tento kompromis volen velmi často.



Obrázek 1.6: Použití lineární interpolace

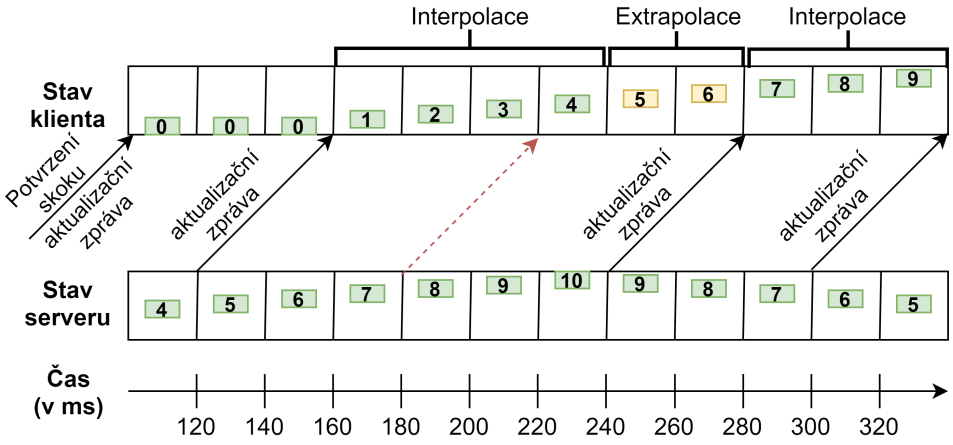
1.2.5.2 Extrapolace

Interpolace sice dokáže přinést hráči plynulejší zážitek, ale nedokáže ho přiblížit k tomu, co se děje na serveru. Zároveň, když není aktualizační zpráva doručena, klient nemá na základě čeho interpolovat. Pro řešení těchto problémů je vhodné využít extrapolaci.

Extrapolace funguje na stejném principu jako interpolace s tím rozdílem, že hledaná hodnota leží mimo interval známých hodnot [15]. Nejčastěji se extrapolace používá dvěma způsoby - společně s interpolací nebo nahrazením interpolace. Pro správné extrapolování budoucího stavu je důležité, aby klient používal stejnou simulační logiku, jako používá server. K tomu je potřeba, aby klient znal hodnoty všech simulačních parametrů, případně alespoň jejich odhady.

Extrapolace doplňující interpolaci se využívá především při výpadku aktualizační zprávy. Na obrázku 1.7 je vidět výpadek zprávy, která byla odeslána ze serveru v čase 180. Klient v čase 240 již nemá data z budoucího stavu hry, tudíž nemá na základě čeho interpolovat. V tento moment je vhodné přepnout na extrapolaci a data budoucího stavu odhadnout tak, aby hráč výpadek zprávy nepocítil na plynulosti hry. Poté, co dorazí další aktualizační zprávy, je možné opět interpolovat.

V situacích, kdy je důležité, aby herní stav byl časově co neblíže tomu serverovému, je možné použít primárně extrapolaci. Když klient obdrží aktualizační zprávu, je přibližně o polovinu RTT starší, tedy stav na serveru je přibližně o polovinu RTT napřed. Z toho vyplývá, že k tomu, aby byl klientův stav herního světa co nejblíže k tomu serverovému, musí klient provádět navíc



Obrázek 1.7: Použití extrapolace při výpadku zprávy

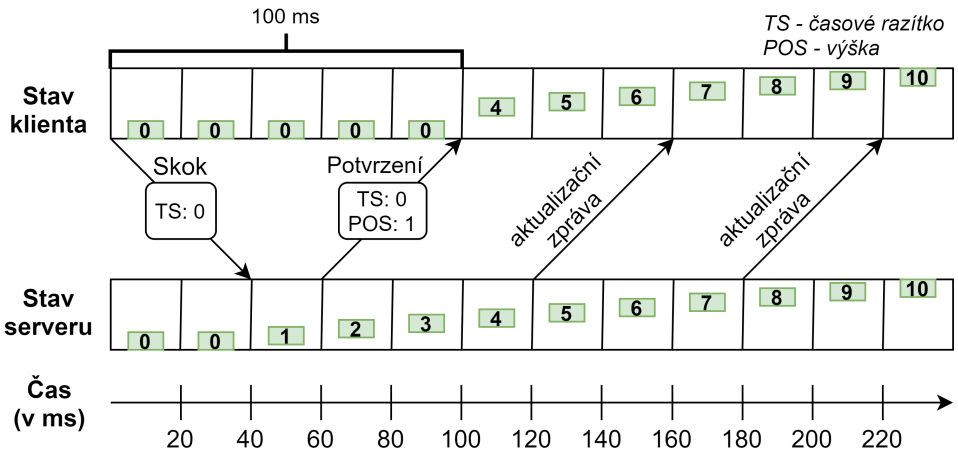
simulaci právě o polovinu RTT. K provedení této simulace klient tudíž musí být schopný odhadnout RTT. Přístupů k získání tohoto odhadu je více. Jeden ze základních spočívá v přiložení timestampu[6] do každé akční zprávy a jejího potvrzení. Klient do každé akční zprávy přidá timestamp, který zaznamenává jeho lokální čas. Server následně požadavek přijme, vyhodnotí ho a timestamp pouze překopíruje do potvrzení. Na základě timestampu v potvrzovací zprávě může klient spočítat čas, který uběhl od odeslání. Tento čas je právě RTT.

Obrázek 1.8 znázorňuje použití extrapolace s uvedeným přístupem odhadu RTT na modelovém příkladu. Klient v čase 0 odesílá na server požadavek zahájení výskoku. Server požadavek vyhodnotí a odešle klientovi zpět potvrzení s daty o poloze, včetně překopírovaného timestampu. Klient následně dopočítá $RTT = aktuální\ čas$ - $timestamp$, na základě čehož je odhadované časové zpoždění rovné $1/2 * RTT = 1/2 * (100 - 0) = 50$. Klient je tedy o 50 milisekund opožděn oproti serveru, proto musí extrapolovat stav herního světa o 50 milisekund.

Vzhledem k tomu, že rychlost komunikace nemusí být stejná v obou směrech a doba vyhodnocení na serveru může být různá, je polovina RTT pouze odhad, jak dlouho jsou data stará. Již na modelovém příkladu je vidět, že se reálné zpoždění liší o 10 milisekund. Nicméně pro většinu her je tato aproximace dostatečně dobrá.

U objektů, které jsou ovlivněny uživatelským vstupem, se extrapolace výrazně komplikuje, protože není možné dopředu bezpečně předpovědět, co hráč udělá. Jedním ze způsobů, jak částečně předpovědět hráčovo chování, je **dead recogning**.

[6] Časové razítko - bod v čase



Obrázek 1.8: Použití extrapolace pro přesnější odhad herního stavu

Dead recogning je způsob predikování budoucího chování objektu na základě jeho aktuálního chování [16]. Tento způsob predikce je založen na myšlence, že objekt bude dělat to, co momentálně dělá. Při použití tohoto způsobu predikování je jistý výskyt chyb, a to vždy, když objekt změní své chování. Například pokud hráč běží doleva, předpokládá se, že bude pokračovat v běhu doleva. Predikce bude fungovat správně do doby, než hráč zastaví nebo změní směr.

Výsledky extrapolace je potřeba vždy při přijetí nových dat ze serveru kontrolovat a v případě, že se liší, je nutné klientův stav hry opravit. Základní způsoby opravy jsou:

- **Okamžité opravení**. Nejjednodušší způsob opravy, stav je ihned nahrazen tím ze serveru, přičemž hráč velice pravděpodobně uvidí trhané skoky objektů.
- **Interpolace**. Klient plynule interpoluje k dobrému stavu za určitý počet snímků.
- **Úpravy parametrů**. V některých případech může vypadat interpolace nepřirozeně. Pokud to situace dovoluje, lze lehce upravit simulační parametry tak, aby se klient postupně dostal do stejného stavu, jako je server. Například u závodních her lze lehce změnit akceleraci a směr tak, aby klientův herní stav konvergoval ke správnému stavu.

Hry typicky nepoužívají pouze jeden z těchto způsobů, nýbrž jich kombinují více dohromady. Výběr způsobu opravy vždy záleží na žánru hry a konkrétním případu použití. U výše zmíněných závodních her je více žádoucí

použít úpravu parametrů, protože citelné škubání auta může hráče rušit více, než malé upravení simulačních parametrů. Kdežto například ve střílečkách, kde jde především o přesné míření, je žádoucí malé chyby interpolovat a ty větší okamžitě opravit tak, aby nedocházelo k falešným trefám.

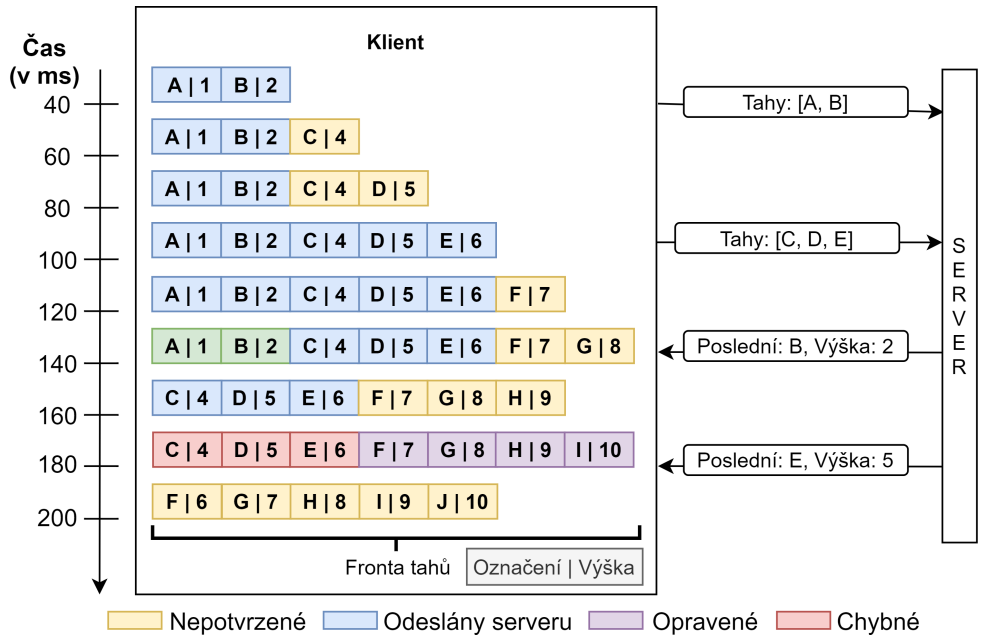
1.2.5.3 Predikce akcí na klientovi

Základní myšlenkou predikování herních akcí je umožnit klientovi nějakým způsobem ihned reagovat na hráčův vstup, případně herní akce tak, aby zpoždění bylo co nejméně citelné. Způsobů, kterými lze predikci realizovat, je nepřeberné množství. Velice záleží na konkrétním objektu, typu akce (např. je akce kriticky důležitá pro hratelnost), žánru hry a dalších aspektech. Je nutné také počítat s tím, že s predikcí se neoddělitelně pojí i výskyt chyb. Z tohoto důvodu nelze obecně říci, jak predikce řešit. Mezi dva hlavní způsoby, které se často používají patří **schování za efekty** a **move strategy** vzor.

Schování za efekty Téměř všechny online hry tuto techniku alespoň v malém množství využívají. Hráč držící raketomet v FPS hře očekává, že při zmáčknutí tlačítka bude ihned vypálena raketa. Bohužel to zpravidla není možné. Následné spárování klientského objektu rakety s objektem rakety na serveru není triviální a přináší mnoho problémů. Proto vývojáři často sáhnou po řešení, při kterém hráči ihned přehrají různé vizuální efekty či animace trvající minimálně po dobu RTT. Hráč tím pádem vidí reakci na svůj vstup okamžitě a zároveň je zde dostatečný čas na to, aby mohla proběhnout výměna dat mezi klientem a serverem. Na příkladu s raketou takové efekty mohou být různé jiskry, záblesky, zvuk startující rakety a mnohé další.

Pokud predikce není správná a server danou akci zamítne, hráč sice uvidí nebo uslyší efekty, které v tento moment nedávají smysl, ale herní stav zůstane synchronizován. Takový kompromis je často dostačující, jelikož k těmto situacím dochází zřídka.

Move strategy Strategie pohybů přináší do herní logiky koncept tahů. Jako tah se označuje struktura, která obsahuje informace o veškerých parametrech a vstupech v daném snímku, které jsou pro simulaci konkrétního objektu relevantní. Tahy jsou postupně ukládány do fronty a ve skupinách odesílány na server. Server na jejich základě provede simulaci a odešle klientovi odpověď s výsledky simulace a s odkazem na poslední tah, který byl v simulaci zahrnut. Klient z fronty odstraní tahy až po poslední tah (včetně), který byl do serverové simulace zahrnut. V případě, že se výsledky ze serveru shodují s výsledky klienta, považuje simulaci za potvrzenou a nedělá nic dalšího. Pokud se výsledky neshodují, klient objektu nastaví shodný stav s výsledným stavem serveru a provede z tohoto nového výchozího stavu simulaci všech tahů, které jsou aktuálně ve frontě.



Obrázek 1.9: Použití strategie tahů na modelovém příkladě

Použití konceptu *move strategy* na modelovém příkladu je vyobrazeno na obrázku 1.9. Poté, co hráč stiskne tlačítko, kterým indikuje zahájení výskoku, klient ihned začne simulovat výskok. Každý snímek si vytvoří nový tah, který obsahuje své označení, informace o hráčově vstupu, stav objektu včetně aktuální výšky a uloží ho do fronty. Simulaci tahů s označením A a B server potvrdí, kdežto výsledek simulace tahů C, D a E se liší. Klient proto musí provést simulaci tahů F, G, H a I znovu s tím rozdílem, že výchozí výška pro simulaci bude ta, kterou spočítal server, tedy 5. Tím se opraví chyba, která při simulaci na klientovi vznikla v čase 60.

1.3 Herní enginy pro vývoj multiplayer her

Dnes již existuje mnoho herních enginů, které jsou zcela zdarma nebo velmi dostupné. To významně pomáhá zejména malým herním studiím, které nemají prostředky na vytvoření vlastních řešení. Pro vývoj multiplayer her toto platí ještě o něco více, jelikož se vývojáři mohou rovnou věnovat vývoji samotné herní logiky. Nízkoúrovňová síťová komunikace a věci s ní spojené jsou často v herních enginech již zaintegrovány a poskytnuty vývojářům.

V této sekci budou představeny z hlediska síťové komunikace dva současně nejvíce používané enginy - **Unreal Engine 4** a **Unity**. Porovnání ostatních funkcionalit je možné nalézt v práci *Overview and Comparative Analysis of*

Game Engines for Desktop and Mobile Devices [17].

1.3.1 Unreal Engine 4

Tento herní engine, zkráceně UE4, je vyvíjen společností *Epic Games* a jeho první verze byla vydána v roce 1998 [18]. Unreal Engine 4 se soustředí primárně na 3D hry. Dříve se zaměřoval především na FPS multiplayer hry, nicméně dnes je v něm možné vyvíjet téměř jakýkoliv herní žánr. Hlavním programovacím jazykem enginu je *C++*, který doplňuje vizuální skriptovací systém *Blueprints*. Podporuje velké množství platforem, jako je Windowns, Playstation 5, Playstation 4, Xbox Series X, Nindendo Switch, Google Stadia, MacOS, iOS, Android a mnoho dalších [19]. Zdrojové kódy enginu jsou veřejně dostupné na *GitHubu* [20] pro všechny vývojáře, kteří mají spárovaný *GitHub* účet s *EpicGames* účtem. Engine je zcela zdarma do té doby, než výdělek překročí 1 000 000 $. V případě překročení této částky si společnost *Epic Games* nárokuje 5% z výdělku [21].

Z hlediska multiplayeru je engine navržen tak, aby striktně využíval topologii client-server, přičemž je možné zvolit variantu dedicated i listen serveru. Herní objekty v tomto enginu mají rozsáhlou hierarchii tříd a podtříd s různou již hotovou funkcionalitou, kterou lze libovolně přetěžovat. S herními objekty se pojí důležitý princip síťových rolí, originálně nazývaných *network roles*. Tyto role určují, jakým způsobem se objekt v dané herní instanci má chovat a zda herní instance má právo s objektem manipulovat. Server má ve většině případů roli autoritativní, tedy může s objekty manipulovat a měnit je, kdežto klienti objekty pouze simulují a přebírají informace od serveru. Herní objekty mají zabudovanou podporu replikace, což v terminologii Unreal Enginu znamená zrcadlení a automatické synchronizování proměnných a stavů objektů ze serveru na klienty. Herní objekty mimo to podporují různé typy RPCs[7]. Engine poskytuje také rozhraní pro integraci podpůrných online služeb. Některé integrace online služeb, jako například integrace *Steamworks* nebo *Oculus*, jsou v enginu již hotové a dostupné v podobě pluginu.

1.3.2 Unity

Unity je 2D a 3D engine vhodný pro hry všeho druhu. První verze byla vydána v roce 2005 společností *Unity Technologies* [22]. Využívá programovací jazyk *C#* a je zde možnost vizuálního skriptování pomocí doplňku *Bolt*, který lze do enginu zdarma přidat. Stejně jako Unreal Engine 4, i Unity podporuje velké množství platforem. Dokud příjmy a financování vývojářů nepřesáhne 100 000 $ za rok, engine je zcela zdarma [23]. Poté je nutné zakoupit některou z licencí.

Unity má knihovnu *UNet*, která poskytuje vysokoúrovňovou abstrakci nad síťovou komunikací, ale zároveň dává vývojářům k dispozici i nízkoúrovňo-

[7] Remote procedure calls - vzdálené volání procedur, umožňuje zavolat funkce na jiném zařízení v rámci sítě

vou transportní vrstvu. Transportní vrstva abstrahuje vytváření socketů pro konkrétní platformy a místo toho umožňuje posílat data v různých režimech, například spolehlivém režimu s garancí doručení. Touto cestou se ovšem většina vývojářů nevydá, jelikož je nutné vytvořit vrstvu, která spojí právě transportní vrstvu s herní logikou. Hodí se pro hry s velmi specifickými požadavky na síťovou komunikaci. Místo toho lze použít vysokoúrovňovou abstrakci. Ta je založená na topologii client-server a podporuje dedicated i listen servery. Unity je silně postavené na komponentové architektuře, tedy herní objekty se rozšiřují pomocí komponent. K tomu, aby komponenta byla schopna komunikovat po síti, musí dědit od *NetworkBehaviour* třídy. Tato třída přidává komponentě podporu replikace a RPCs.

Knihovna *UNet* je dnes označována za zastaralou a neměla by se již používat [24]. Náhradou by mělo být aktuálně vyvíjené řešení *MLAPI* [25]. Existují doplňky třetích stran, které se o síťovou komunikaci starají a je možné je stáhnout či zakoupit v Unity obchodě. Jedním takovým je doplněk *Mirror* [26].

1.4 Podpůrné služby

Ačkoliv to nemusí být zřejmé, v dnešní době není téměř možné, aby hra fungovala bez podpůrných služeb. Hráči jsou již zvyklí na určitý standard dnešních her, kdy si mohou před začátkem samotné hry chatovat, tvořit společně skupiny, nakupovat různé herní předměty, vidět, kdo z jejich kamarádů aktuálně hraje, zaznamenávat si různé herní úspěchy a podobně. Pro zajištění tohoto standardu je nutné, aby hra využívala další podpůrné služby, které tyto funkcionality poskytují.

Následně budou představeny a vysvětleny zásadní služby, které se ve hrách využívají.

Login služba Zajišťuje přihlášení a ověření uživatele.

Friends služba Poskytuje hráčům možnost označit jiné hráče jako své kamarády a spravovat tak seznamy přátel. Na základě těchto seznamů hráči vidí, zda jejich kamarádi právě hrají a pomocí jiných služeb jim mohou pohodlně napsat nebo je pozvat do hry či skupiny.

Chat služba Jak název napovídá, jedná se o službu, která zprostředkovává chat mezi hráči. Dnes velmi často bývá součástí této služby i hlasový chat.

Party služba V případě týmových her chtějí hráči často před začátkem hry vytvořit skupinu tak, aby při hře tvořili jeden tým. Party služba se stará o vytváření skupin a jejich správu.

Matchmaking služba Některé hry seskupují hráče do zápasů podle počtu odehraných her, určitých statistik nebo metadat tak, aby schopnosti a zkušenosti hráčů byly podobné a herní souboj tak byl vyrovnaný. K tomu je využívána právě Matchmaking služba.

Achievement služba Tato služba zaznamenává a odemyká různé herní úspěchy a výzvy, které hráči mohou splnit. Jako výzvu si lze představit například to, že hráč musí vyhrát deset her po sobě nebo dojet závod bez jediného nárazu.

Statistics služba Slouží k definování a ukládání různých herních dat a statistik o hráčích a herních soubojích, které mohou být důležité například pro Matchmaking a Achievement službu.

Storage služba Jedná se o službu, která slouží k ukládání hráčských dat a souborů, jako jsou například profilové obrázky.

Provozovatelé distribučních platforem, které slouží k prodeji her, často podpůrné služby nabízejí zdarma v případě, že hra přes jejich platformu bude prodávána. Největší takovou platformou je *Steam* [27], jejíž služby jsou dostupné přes rozhraní *Steamworks* [28]. Dále existuje také platforma *Epic Games Store* [29] se svými službami *Epic Online Services* [30], které jsou zcela zdarma i v případě, že hra na jejich platformě nebude vůbec vydána. Obě platformy poskytují velmi podobné služby, například autentizační službu, chatovací službu, matchmaking službu, služby zprostředkovávající NAT Traversal atd.

Více informací o existujících podpůrných službách a jejich srovnání je možné nalézt v práci *Podpůrné služby pro online hru Inpemo* od Ing. Petra Nohejla [31], která se této tématice věnuje.

Kapitola 2

Výběr nástrojů a technologií

Z časových i finančních důvodů bude hra Inpemo zcela jistě vyvinuta v existujícím herním enginu. V takovém případě je většinou prospěšné vybrat herní engine ještě předtím, než se přistoupí k vytváření návrhu a architektury. K tomu, aby prostředky enginu byly využity co možná nejvíce, je dobré, aby byl návrh hry částečně postaven na základě architektury vybraného enginu.

Tato kapitola se bude zabývat výběrem nástrojů a technologií, které budou použity při vývoji hry Inpemo.

2.1 Herní engine

Na začátek je dobré zdůraznit, že oba představené enginy jsou pro vývoj hry Inpemo naprosto dostačující a svými nástroji a funkcionalitami naprosto převyšují požadavky této hry. Ovšem co se síťové komunikace týče, Unreal Engine 4 je na tom o poznání lépe a je i širší vývojářskou komunitou vnímán jako primární volba pro tvorbu online multiplayer hry. Navíc má oproti Unity vestavěná rozhraní, která slouží k integraci podpůrných služeb.

Z výše zmíněných důvodů a také toho, že autor práce vnímá využití Unreal Enginu jako výzvu a příležitost naučit se něco nového, jelikož s Unity má již pokročilou zkušenost, bude pro vývoj hry použit Unreal Engine verze 4.26.

2.2 Perforce - HelixCore

Perforce HelixCore je verzovací systém od společnosti *Perforce Software, Inc.* Tento verzovací systém je centralizovaný a oproti SVN nabízí pokročilejší možnosti mergování změn a větší výkon. Jelikož herní projekty obsahují velké množství binárních souborů, centralizovaný systém s možností jejich uzamčení je nutností. Z těchto důvodu je Perforce používán velkými herními společnostmi, jako *Ubisoft*, *CD Project Red*, *Crytek* nebo právě *Epic Games*. Unreal

Engine Editor má navíc integrovanou podporu pro tento verzovací systém zcela zdarma.

2.3 YouTrack

Jedná se o systém sloužící ke správě projektu, evidenci úkolů včetně zaznamenávání odpracovaného času a sledování chyb. Bude použit pro plánování a zaznamenávání práce, časové odhady a v pozdějších fázích vývoje bude sloužit jako dokumentace projektu.

Kapitola 3

Analýza podobných her

Tato kapitola se zabývá představením hry Inpemo a zároveň zkoumává podobné hry, které již existují. Ty byly vybrány na základě následujících parametrů:

- přítomnost na distribuční platformě Steam,
- počet instalací alespoň 100 000,
- 3D hra žánru MOBA[8] ,
- hra není založena na principu zničení hlavních budov (jako například Dota 2),
- maximální počet hráčů v rámci jedné hry je minimálně 4 a maximálně 10,
- top-down pohled,
- herní postavy ve hře bojují pomocí různých schopností a kouzel.

Podle těchto kritérií byly vybrány dvě hry - **Battlerite** a **Spellsworn**.

[8]Multiplayer Online Battle Arena - online hra pro více hráčů, kteří proti sobě bojují v aréně.

3.1 Představení hry Inpemo

Inpemo je online 3D akční hra pro více hráčů spadající do žánru MOBA. Hráči hru hrají ve formě zápasů, přičemž se mezi zápasy nepřenáší herní postup ze zápasů minulých. Zápas je pro dva až osm hráčů. Hra využívá pohled top-down. Každý hráč ovládá právě jednu postavu, kterou si vybírá před začátkem zápasu.

Každá postava má určitý počet bodu zdraví a unikátní soubor kouzel. Herní mapa je pomyslně rozdělena na dvě zóny - bezpečnou a smrtící. V bezpečné zóně se postava může volně pohybovat a není za to žádným způsobem postihována, kdežto pokud se postava nachází ve smrtící zóně, je jí opakovaně po uběhnutí určitého časového intervalu snižován počet bodů zdraví. V případě, že postavě klesne počet bodů zdraví na nulu, postava umírá a hráč je vyřazen ze zápasu.

Hlavní mechanikou hry je odstrkávání. Postavy mohou během zápasu sesílat kouzla, které mají různé efekty a dopady, nicméně jejich primární účel není ubírání bodů zdraví nepřátelským postavám. Kouzla slouží primárně k odstrkávání nepřátelských postav tak, aby byly odstrčeny do smrtící zóny, která, jak bylo vysvětleno, je hlavním zdrojem poškození v zápasu.

Cílem každého zápasu je přežít jako poslední a vystrčit všechny hráče ven z bezpečné zóny.

3.2 Battlerite [32]

Hodnocení na Steamu: **velmi pozitivní (80% - 94%)**
Počet instalací na Steamu: **6 600 000 [33]**

Hra jménem Battlerite byla vytvořena herním studiem *Stunlock Studios*. Jedná se o týmovou akční hru vycházející z herního žánru MOBA, která je považována za nástupce úspěšné hry Bloodline Champions, jež byla vytvořena totožným herním studiem. Hra byla v předběžném přístupu a free-to-play modelu vydána v srpnu roku 2016, oficiálně pak v listopadu 2017.

Battlerite je založena na principu zápasů. Zápas je rozdělen do více kol a účastní se ho dva týmy bojující proti sobě, přičemž každý tým je tvořen dvěma nebo třemi hráči. Před začátkem zápasu si každý hráč vybere unikátní postavu nazývanou *champion*, která je vybavena jedinečnými útočnými a obrannými schopnostmi. Tým, který porazí všechny postavy nepřátelského týmu, vyhrává kolo, přičemž tým, který dosáhne jako první tří vyhraných kol, vyhrává zápas. Kola mají přibližně dvouminutový časovač, po jehož uběhnutí se herní mapa začne pomalu zmenšovat a nutí tak hráče k většímu kontaktu. Herní mapy jsou různé a vybírají se náhodně před každým zápasem. V zápase nelze kupovat žádné předměty ani jakkoliv vylepšovat svou postavu.

Obrázek 3.1: Screenshot zachycující souboj dvou dvoučlenných týmů ve hře Battlerite

Hra byla vyvinuta v herním enginu Unity. Vývojáři použili client-server topologii, konkrétně variantu s dedikovanými servery, jelikož hra měla ambice stát se e-sportovým titulem. Z toho důvodu byl kladen velký důraz na stabilní latenci a zamezení jakéhokoliv podvádění. Hráči vidí herní scénu z pohledu top-down. Uživatelské rozhraní je minimalistické a umístěno k okrajům obrazovky tak, aby zabíralo co nejméně místa a nesnižovalo tak hráčovu viditelnost. Pohyb championa je do osmi směrů a je defaultně ovládán pomocí kláves WSAD. Schopnosti se zaměřují pomocí myši.

3.3 Spellsworn [34]

Hodnocení na Steamu: **velmi pozitivní (80% - 94%)**
Počet instalací na Steamu: **997 000 [35]**

Spellsworn je akční MOBA hra pro maximálně osm hráčů. Byla vyvinuta herním studiem *Frogsong Studios* a vyšla v předběžném přístupu koncem roku 2015. Původně využívala placený model, ale v březnu 2018 přešla na model free-to-play. Tím byla oficiálně vydána.

Herní mechaniky a principy vychází z herního módu Warlocks. Ten byl vytvořen fanoušky pro hru Warcraft 3. Spellsworn je také založen na principu zápasů, které se skládají z několika kol. Herní mapa se s ubíhajícím časem zmenšuje a je obklopena nebezpečnou zónou. Všichni hráči hrají za stejné

Obrázek 3.2: Scéna ze hry Spellsworn

postavy, jež jsou odlišeny barvou. Postavy na začátku zápasu nemají žádná kouzla ani schopnosti. Před každým kolem hráči dostanou určitý obnos herní měny, za kterou si mohou kouzla a schopnosti kupovat a vylepšovat ta, která jejich postava již má. Kouzla a schopnosti jsou různého charakteru, ale primárně slouží k obraně a udělování poškození ostatním hráčům či odstrkávání hráčů do nebezpečné zóny. Postavě, která se nachází v nebezpečné zóně, jsou postupně ubírány body zdraví. Cílem každého kola je zůstat naživu jako poslední. Zápas vyhrává hráč s nejvíce vyhranými koly.

Vývojáři pro vývoj této hry použili Unreal Engine 4, hra tedy využívá client-server topologii. Zvolena byla varianta listen serveru společně s použitím Steamworks podpůrných služeb, které mimo jiné poskytují NAT Traversal. Hráči hrají z pohledu top-down a pohybují se i míří pomocí myši.

Kapitola 4

Úvod do Unreal Engine 4

Jelikož Unreal Engine 4 obsahuje komplexní framework, který je opravdu specifický, je nutné některé jeho části představit a porozumět jim již před návrhem, aby bylo možné vytvořit takový návrh, který bude v souladu s jeho ekosystémem.

V této kapitole jsou představeny a vysvětleny fundamentální prvky a principy Unreal Enginu 4. Kapitola nepokrývá kompletně celý engine, nýbrž se soustředí pouze na části, které jsou relevantní z hlediska implementace prototypu hry Inpemo. Více informací je možné najít v oficiální dokumentaci [36], ze které tato kapitola vychází.

4.1 Základní prvky a jejich vztahy

Unreal Engine 4 má mnoho editorů a zabudovaných nástrojů. První, co uživatel při spuštění enginu uvidí, je *Unreal Editor*, jehož vzhled je možné vidět na obrázku 4.1. Ten poskytuje uživatelské rozhraní, díky kterému lze mimo jiné importovat a organizovat herní assety[9], upravovat a tvořit herní scénu, ale hlavně přistupovat k ostatním nástrojům enginu, jako je například vizuální skriptovací systém *Blueprints* nebo *Material Editor*, který slouží k tvorbě grafických povrchů a materiálů.

Níže jsou vysvětleny elementární pojmy týkající se tohoto enginu, které jsou v práci používány.

Object Základním stavebním kamenem enginu je *Object*. Lze říci, že prakticky vše v Unreal Enginu 4 dědí od Objektu. Objekt je v C++ reprezentován třídou *UObject*. Tato třída implementuje prvky jako je například garbage collection nebo podpora speciálních metadat pro třídní proměnné, díky kterým lze vystavit proměnné do Unreal Editoru nebo je serializovat a deserializovat.

[9] Jako assety se označují všechny soubory, které hra bude obsahovat. Mezi ně se řadí zvuky, 3D modely, textury, skripty a mnohé další.

Obrázek 4.1: Uživatelské rozhraní Unreal Editoru

Level Všechny objekty, které hráči ve hře vidí nebo s nimi interagují, jsou součástí této struktury. *Level*, přeloženo jako úroveň, je definovaná část herního světa, která obsahuje 3D modely prostředí, osvětlení, různé povrchy, postavy a mnoho dalšího, což dohromady tvoří prostředí herního světa.

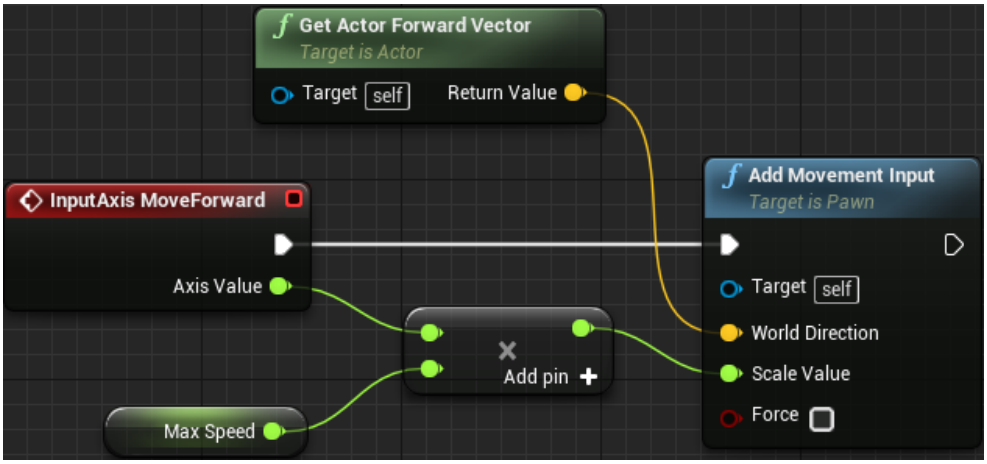
World *World* je reprezentace herního světa. Obsahuje seznam momentálně načtených úrovní a poskytuje přístup k důležitým částem programu.

Actor Pojem *Actor* označuje jakýkoliv herní objekt, který je možné umístit do úrovně. Jedná se o obecný objekt, který podporuje 3D transformace (translaci, rotaci, zvětšení) a může být dynamicky za běhu programu vytvořen i zničen. Každý *Actor* dědí od základní třídy *AActor*.

Component Komponenta je funkcionalita nebo její část, která může být připojena na instanci třídy *Actor*. Instance komponenty nemůže být vytvořena sama o sobě, nýbrž vždy musí být napojena na některou existující instanci třídy *Actor*, čímž je daný *Actor* rozšířen o funkcionalitu, kterou komponenta poskytuje. Příkladem takové komponenty může být například *MeshComponent*, která umožňuje přiřadit jejímu držiteli 3D model.

4.1.1 Blueprint Visual Scripting

Jedná se o komplexní skriptovací systém, pomocí kterého lze vytvářet herní prvky přímo uvnitř editoru. Je založený na principu uzlů a jejich spojování, což si lze představit jako spojování a vytváření krabiček s určitou funkcionalitou, které dohromady tvoří orientovaný graf (viz obrázek 4.2).



Obrázek 4.2: Uživatelské rozhraní Unreal Editoru

V systému lze vytvářet či upravovat chování a interakce objektů ve hře bez nutnosti psaní C++ kódu a jeho následného kompilování. Systém je velmi flexibilní a poměrně výkonný, jelikož poskytuje přístup k nástrojům, které jsou obvykle dostupné pouze pomocí kódu. Zároveň je možné vytvořit nové systémy a funkce přímo v C++ a pomocí speciálních maker je vystavit do tohoto systému, díky čemuž je možné je rozšířit nebo úplně přetížit. Systém mimo jiné také umožňuje vytváření nových proměnných a přenastavování inicializačních hodnot.

Objekty vytvořené pomocí tohoto systému jsou běžně označovány jako *Blueprints*.

4.1.2 Herní framework

Engine dává vývojářům k dispozici několik tříd, které obsahují mnoho již hotové generické herní logiky. Tyto třídy poskytují funkcionalitu pro reprezentaci hráčů a různých herních postav, včetně jejich ovládání pomocí uživatelských vstupů nebo umělé inteligence. Dále dávají k dispozici logiku pro nastavování herních pravidel a udržování stavu herního světa stejně tak jako stavu každého hráče. Nakonec se sem řadí i třídy starající se o herní kamery nebo uživatelské rozhraní.

Všechny následující třídy poskytují některou z výše zmíněných funkcionalit. Vycházejí z třídy *Actor*, mohou tedy být individuálně přidány do úrovně dle potřeby.

Pawn Tato třída je základní třídou pro všechny třídy *Actor*, jejichž instance mohou být ovládány hráčem nebo umělou inteligencí. *Pawn* je fyzická reprezentace herní postavy v rámci herního světa. Určuje tedy to, jak hráč nebo umělá inteligence vizuálně vypadá, ale taky způsob, jakým interaguje a koliduje se světem. Pro specifické hry, například karetní, není využití této třídy ihned zřejmé, jelikož v herním světě hráčská postava nemusí vůbec existovat.

I přesto je tato třída v takových hrách používána, protože je ideálním místem pro definování a zpracování uživatelských herních akcí.

Character Jedná se o podtřídu třídy *Pawn*, která navíc obsahuje *SkeletalMeshComponent* a *CharacterMovementComponent* a má primárně reprezentovat lidské postavy, nicméně zcela běžně je využívána i pro postavy výrazně odlišné. *SkeletalMeshComponent* se používá pro komplexnější 3D modely s animacemi, jejichž součástí je kostra. Neméně důležitou komponentou je *CharacterMovementComponent*. Ta poskytuje základní typy pohybu (chůze, běh, plavání, skok), přičemž může být rozšířena o libovolný počet vlastních variant. Z hlediska multiplayeru je důležité zmínit, že tato komponenta implementuje základní mechanismy pro plynulý pohyb po síti.

Controller Instance třídy *Controller* jsou objekty bez vizuální reprezentace, které mají zastávat roli hráče. *Controller* může ovládnout instanci třídy *Pawn* a převzít tak nad ní kontrolu. V terminologii enginu se tato akce nazývá „possess“. Podtřídy třídy *Controller* se dělí na dva typy - *PlayerController* a *AIController*.

PlayerController je používán lidskými hráči k ovládání tříd *Pawn*. *AIController* za tímto účelem implementuje umělou inteligenci. Defaultně je mezi třídou *Controller* a třídou *Pawn* vztah „one-to-one“, tedy každá instance třídy *Controller* má v danou situaci přiřazenou maximálně jednu instanci třídy *Pawn*, stejně tak jako instance třídy *Pawn* je ovládána maximálně jednou instancí třídy *Controller*.

GameInstance Instance této třídy je pouze jedna a je perzistentní po celou dobu, co je program spuštěn. Tedy i přesun mezi jednotlivými herními mapami zachová stejnou instanci této třídy. Třída především slouží k udržení herních dat dostupných napříč celou hrou (různé nastavení hry - např. rozlišení, uživatelské jméno hráče) a poskytuje důležité herní události pro zpracování síťových chyb.

GameMode *GameMode* lze považovat za srdce celého herního frameworku. Specifikuje, jaké konkrétní třídy budou v dané úrovni použity pro vytváření instancí důležitých herních objektů (*PlayerController*, *Pawn*, *GameState*, *PlayerState* atd.) a je určen pro definování a implementaci herních pravidel. *GameMode* je navržen tak, aby byl schopný pokrýt neomezené variace herních pravidel. Jako herní pravidla si lze představit například:

- počet hráčů potřebných k odstartování hry,
- způsob, jakým se hráči připojují do hry (výběr herní postavy, výběr místa, kde hráč začíná, výběr týmu atd.),
- kdy hra končí a jakým způsobem je vybrán vítěz,
- zda lze hra pozastavit a jak je pozastavení hry řešeno.

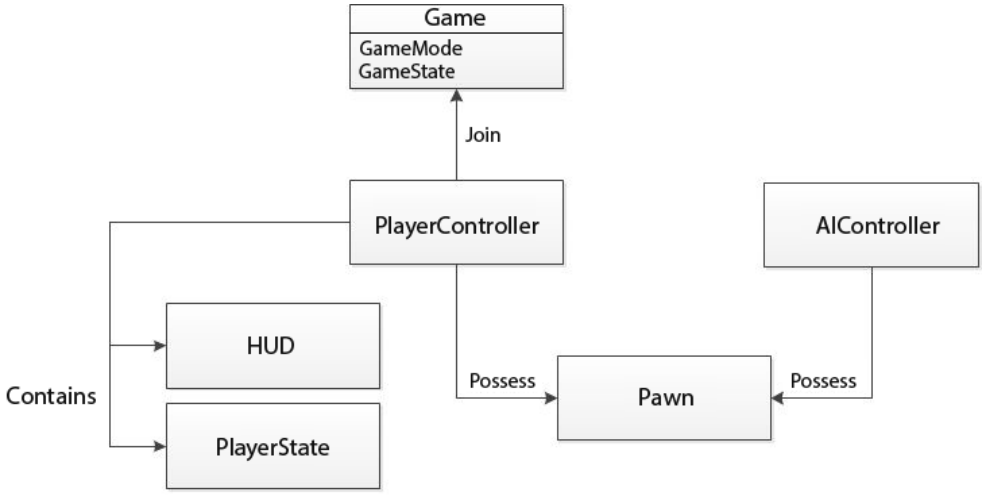
V případě síťového multiplayeru *GameMode* existuje pouze na serveru, tedy žádný klient nemá instanci této třídy.

GameState Tuto třídu lze považovat za kontejner pro data, která jsou společná pro všechny hráče v dané hře. Vzhledem k tomu, že *GameMode* existuje pouze na serveru (v síťových hrách), je potřeba, aby existovalo dostupné místo pro všechny klienty, kde budou uložena data, která popisují aktuální herní stav. Tím místem je právě *GameState*. Mezi taková data mimo jiné patří týmové skóre, uběhlý čas hry nebo seznam připojených hráčů.

PlayerState *PlayerState* je podobný jako *GameState* s tím rozdílem, že drží informace spojené s konkrétní instancí třídy *Controller*. Příkladem takových dat může být jméno hráče, skóre, počet smrtí nebo předměty, které vlastní. V kontextu síťového multiplayeru má každý klient všechny instance třídy *PlayerState* (svou instanci a instance všech ostatních hráčů).

HUD Třída *HUD* je základní třídou pro zobrazování uživatelského rozhraní. Každý *PlayerController* má právě jednu instanci této třídy. Dříve se třída používala přímo pro vykreslování jednotlivých elementů uživatelského rozhraní, jelikož poskytuje API pro vykreslování. S příchodem systému *UMG* [37] se *HUD* začal používat spíše jako kontejner pro ukládání a správu různých částí uživatelského rozhraní.

Vztahy mezi výše zmíněnými třídami je možné vidět na diagramu 4.3. Základ hry je složen z pravidel definovaných v *GameMode* a herního stavu uloženého v *GameState*. Hráči se připojují ke hře a jsou reprezentováni třídou *PlayerController*, pomocí které mohou ovládnout herní postavy vycházející ze třídy *Pawn*. *PlayerController* také obsahuje *HUD* starající se o uživatelské rozhraní a *PlayerState*, který slouží jako hlavní paměť pro stav hráče.



Obrázek 4.3: Vztahy mezi hlavními třídami herního frameworku [1]

4.2 Síťová komunikace a multiplayer

Unreal Engine 4 poskytuje robustní framework pro usnadnění vývoje online her, díky kterému se vývojáři nemusejí zabývat nízkoúrovňovou síťovou komunikací, ale mohou rovnou řešit herní kód a to, jestli funguje správně i po síti.

Vzhledem k tomu, že se jedná o multiplatformní engine, je nutné abstrahovat implementační detaily síťových socketů. To je zařízeno pomocí rozhraní *ISocketSubsystem*, které má několik implementací pro jednotlivé podporované platformy a je zodpovědné za vytváření a adresování socketů.

4.2.1 NetMode

NetMode udává, jaký vztah má daná herní instance vůči herní relaci[10]. Instance hry může nabývat jedné z následujících hodnot.

- **Standalone** Herní instance se chová jako server, přičemž neakceptuje žádná příchozí spojení od vzdálených klientů. Všichni hráči jsou striktně lokální a hra vyhodnocuje serverovou i klientskou logiku. Tento mód se používá pro *singleplayer* a lokální *multiplayer*.

- **Client** Herní instance je připojena k serveru a chová se jako klient. Nevyhodnocuje žádnou serverovou logiku.

[10]Doba, která začíná zahájením hry a končí poslední provedenou hráčskou akcí v rámci stejné hry. Obvykle se označuje jako „game session“.

- **Listen server** V tomto módu herní instance přijímá spojení od vzdálených klientů a chová se jako server, přičemž má i lokální hráče. Vyhodnocuje serverovou i klientskou logiku.

- **Dedicated server** Slouží jako vzdálený server pro herní relaci. Herní instance přijímá spojení od vzdálených klientů a nemá žádného lokálního hráče. Vyhodnocuje pouze serverovou logiku, takže zde nejsou přítomny některé nepotřebné moduly, jako například vykreslování grafiky, modul pro uživatelský vstup, zvukový modul a další.

4.2.2 NetDriver

Každá herní instance má jednu instanci třídy *NetDriver* (ve speciálních případech jich může mít více). Serverový *NetDriver* má seznam instancí třídy *NetConnection*, přičemž každá instance představuje jednoho hráče, který je k serveru připojený. Klient oproti tomu má pouze jednu instanci třídy *NetConnection* reprezentující připojení k serveru.

NetDriver se stará o vytváření nových spojení, přijímání packetů a jejich následné směrování do příslušného *NetConnection*. Packety nejsou zpracovány přímo touto třídou. Ta má množinu instancí třídy *Channel* (viz. obrázek 4.4) a data namísto zpracování přesměruje do příslušného kanálu, který se o zpracování dat postará. Níže jsou uvedeny základní typy kanálů.

- **Control Channel** slouží k zasílání řídících zpráv daného *NetConnection*, například zda má být spojení uzavřeno.

- **Voice Channel** se využívá k posílání hlasových dat.

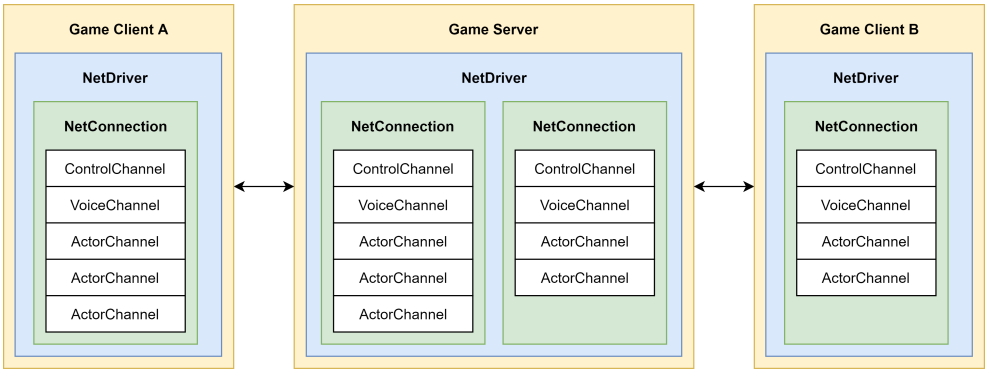
- **Actor Channel** existuje pro každou instanci třídy *Actor*, která je replikována.

Třídy *NetDriver* a *NetConnection* jsou nezávislé na síťovém protokolu. Nepředpokládají garanci doručení ani doručení ve správném pořadí. Místo toho engine implementuje vlastní protokol, který výše zmíněné garance zaručuje. Použité principy je možné najít v hlavičkovém souboru třídy *NetDriver* [38].

4.2.3 Actor Replication

Hlavní třídou, která má podporu pro replikaci, je třída *Actor*. Replikací se rozumí proces reprodukování herního stavu mezi herními instancemi v rámci herní relace. Při jejím správném používání jsou herní instance automaticky synchronizovány.

Replikační systém má několik funkcí, z nichž tři hlavní jsou správa životního cyklu, replikace proměnných a RPCs.



Obrázek 4.4: Vztahy mezi třídami NetDriver, NetConnection a Channel

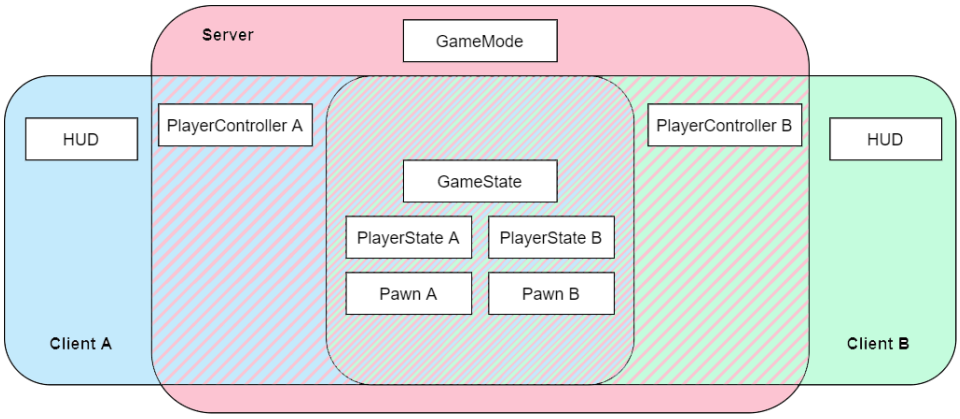
Správa životního cyklu Pokud server vytvoří instanci třídy *Actor*, je automaticky vytvořena stejná instance na všech klientech. Stejně tak pokud je na serveru zničena instance třídy *Actor*, je také zničena na všech klientech.

Proměnné Změny jakékoliv serverové proměnné, která je označena jako replikovaná, jsou automaticky propagovány na všechny klienty. Proměnné lze ještě označit speciálními podmínkami tak, aby byly replikovány například jen v určitém případě nebo pro konkrétní klienty. Více specifických informací lze nalézt v oficiální dokumentaci [39].

RPCs Jedná se o speciální funkce, které jsou volány lokálně, ale vyhodnoceny jsou na jiném stroji u některého z hráčů. *RPCs* jsou v základním nastavení nespolehlivé, tedy není garantováno, že budou vykonány. Nicméně je lze označit jako *Reliable*, v ten moment je jejich vykonání garantováno. Dělí se na tři základní typy [40].

- **Server** - funkce je vykonána na serveru, ať je zavolána ze serveru nebo z klienta.
- **Client** - funkce je vykonána na klientovi, který vlastní daného *Actora*.
- **NetMulticast** - funkce je vykonána na všech zařízeních v herní relaci, včetně serveru.

Jelikož všechny hlavní třídy herního frameworku dědí od třídy *Actor*, replikační systém pro ně funguje také. Pro jejich korektní používání je nutné vědět, komu jednotlivé instance tříd náleží a kde jsou tvořeny. Na obrázku 4.5 je zobrazen případ dedikovaného serveru a je na něm možné vidět, jaké konkrétní instance tříd herního frameworku klienti a server sdílí.



Obrázek 4.5: Sdílení instancí tříd herního frameworku (dedikovaný server)

Třída *GameMode* je pouze na serveru, klienti k ní nemají přístup. Každý klient má instanci třídy *PlayerController*, která ho reprezentuje jakožto hráče, přičemž server má všechny tyto instance. Účastníci herní relace spolu sdílejí *GameState*, všechny instance třídy *PlayerState* a případné herní postavy, které jsou instancemi třídy *Pawn*. Každý klient pak má vlastní *HUD*.

4.3 Online Subsystem

Online Subsystem je systém pro snadnou integraci herních podpůrných služeb. Používání tohoto systému je výhodné především při multiplatformním vývoji. Zaručuje, že při přechodu na jinou platformu není potřeba měnit zdrojový kód hry, stačí přidat integraci konkrétní platformy a udělat pouze konfigurační změny.

Systém podporuje nejrůznější podpůrné služby a je navržen primárně pro řešení asynchronních požadavků, jelikož dopředu není možné určit dobu, za jakou bude požadavek vyřízen. Z tohoto důvodu je celý systém postaven na delegátech. Po spuštění operace je garantováno, že příslušní delegáti budou po jejím dokončení zavoláni. Mimo to je zde také podpora pro reagování na výsledek požadavku nebo ukládání výsledků tak, aby se předešlo zbytečným duplicitním požadavkům.

Systém obsahuje několik rozhraní, která jsou specifická vůči určitému typu požadavků. Níže jsou uvedena některá z nich. Všechna podporovaná rozhraní je možné najít v dokumentaci [41].

- **Achievements** - slouží k získání seznamu všech úspěchů a jejich odemykání.

- **Friends** - poskytuje správu přátel (přidání, odebrání, blokaci) a vyhledání spoluhráčů z minulých her.
- **Identity** - umožňuje přihlášení a odhlášení uživatele.
- **Online User** - využívá se k získání metadat o uživatelích.
- **Presence** - slouží k nastavení uživatelského statusu a získání statusů ostatních hráčů.

4.4 Gameplay Ability System

Gameplay Ability System, zkráceně označován jako „GAS“, je velmi flexibilní framework pro tvorbu kouzel a schopností. S jeho pomocí lze vytvářet různorodé schopnosti s rozmanitými dopady, pasivní schopnosti nebo stavové efekty. Podporuje úrovně schopností, definování podmínek pro použití schopnosti, vizuální efekty, zvuky a mnoho dalšího. Dále také umožňuje vývojářům definovat vlastní atributy postav dle požadavků jejich hry. Velkou výhodou je fakt, že většina zmíněných funkcionalit je navržena pro maximální podporu síťového multiplayeru.

Systém se skládá z mnoha tříd. Mezi hlavní patří *GameplayAbility*, *Attribute*, *GameplayEffect*, *GameplayCue* a *AbilitySystemComponent*.

GameplayAbility třída představuje schopnost jako takovou. Definuje chování, dopady schopnosti a podmínky, za kterých může být schopnost použita.

Attribute je skupina čísel uložených ve speciální struktuře reprezentující specifickou vlastnost jeho držitele. Atribut může být například počet bodů zdraví, úroveň, útočná síla nebo výška výskoku.

GameplayEffect definuje změnu atributů. Efekty jsou čistě datové, neimplementují žádnou logiku a lze je aplikovat na jakoukoliv instanci třídy *Actor*, která má *AbilitySystemComponent* komponentu. Popisují, jakým způsobem se při jejich aplikaci mají atributy změnit.

GameplayCue slouží k zobrazování a vyhodnocování nekritických částí schopností, většinou zvuků a vizuálních efektů. Ty by měly být striktně odděleny od kritického kódu, který má zásadní dopad na stav herního světa.

AbilitySystemComponent je srdcem celého systému. Každý *Actor*, který má být do systému zahrnut, musí mít přiřazenou tuto komponentu. Ta se stará o aktivaci schopností, počítání a ukládání atributů, aplikaci efektů a mnoho dalšího.

Kapitola 5

Analýza a návrh hry Inpemo

Tato kapitola se věnuje analýze požadavků hry Inpemo a jejímu návrhu. Všechny uvedené informace vychází z herního návrhového dokumentu, který je součástí přiloženého USB disku. Návrhový dokument byl sepsán autorem práce, Ing. Petrem Nohejlem, Bc. Karímem Abu Nofalem a Bc. Tomášem Bohuslavem.

5.1 Funkční a nefunkční požadavky

Na základě návrhového dokumentu byly sepsány funkční a nefunkční požadavky. Z důvodu lepší přehlednosti byly seskupeny a rozděleny do několika podsekcí.

5.1.1 Obecné

Funkční požadavky

- **FG01: Hráči se budou moci přihlásit a odhlásit**

  Hra bude využívat herní účty. Na základě toho musí být uživatelé schopni se přihlásit a odhlásit.

- **FG02: K přihlášení a registraci půjde využít Steam a Epic účet**

  Autoři návrhového dokumentu rozhodli, že hra bude distribuována na platformě Steam a Epic Games Store. Obě dvě platformy poskytují SDK, pomocí kterého je možné získat přístupový token na základě přihlášené platformní aplikace v pozadí. Aby bylo uživatelům zajištěno co nejsnadnější přihlášení, bude využito zmíněné API.

- **FG03: Vestavěné uživatelské rozhraní Steam a Epic Games Store bude dostupné ze hry**

Obě dvě platformy mají integrované uživatelské rozhraní, které je dostupné skrze SDK. Uživatel bude moci toto rozhraní zobrazit přímo ze hry pomocí klávesových zkratek.

- **FG04: Maximální počet hráčů bude osm**

  V rámci jedné herní relace bude možné hrát maximálně v osmi hráčích, přičemž ke spuštění hry budou potřeba minimálně dva hráči.

Nefunkční požadavky

- **NG01: Hra bude podporovat OS Microsoft Windows 10**

  Jediný podporovaný operační systém bude Microsoft Windows 10.

- **NG02: Budou využity podpůrné služby od Ing. Petra Nohejla**

  Zadání práce klade jako požadavek na prototyp hry použití podpůrných služeb, které v rámci své diplomové práce vytvořil Ing. Petr Nohejl.

5.1.2 Organizace zápasů a sociální aspekty hry

Funkční požadavky

- **FMS01: Hráči se budou moci přihlásit do vyhledávání zápasu**

  Jelikož hru není možné hrát samostatně, je nutné pro zahájení hry seskupit hráče do skupin. O to se bude starat systém vyhledávání zápasu. Hráč musí být schopen se do tohoto systému přihlásit. Po úspěšném nalezení spoluhráčů bude hráč společně s nalezenými spoluhráči přesunut do herní relace.

- **FMS02: Hráči budou moci založit vlastní zápas a připojit se do něj**

  Aby hráči mohli hrát se svými přáteli, musí být schopni založit vlastní zápas mimo systém vyhledávání zápasů a připojit se do něj. Hra bude tuto funkcionalitu podporovat.

- **FMS03: Hráč uvidí své Steam a Epic přátele, kteří v daný moment hrají hru Inpemo**

  Hra bude obsahovat seznam přátel přihlášeného hráče, kteří aktuálně hrají hru Inpemo a které má přidané jako přátele na platformě Steam nebo Epic Games Store.

- **FMS04: Hráči, kteří jsou již připojeni do vlastního zápasu, budou moci do tohoto zápasu pozvat své přátele**

  Hráče ze seznamu přátel bude možné pozvat do vlastního zápasu.

- **FMS05: Pozvání bude možné poslat a přijmout skrze hru i skrze integrované uživatelské rozhraní platforem Steam a Epic Games Store**

  Platformy Steam a Epic Games Store poskytují jako součást uživatelského rozhraní i tlačítka na odeslání a přijmutí pozvání. Hra bude tato tlačítka integrovat.

- **FMS06: Hráči budou moci utvářet maximálně čtyřčlenné skupiny a společně se připojovat do systému vyhledávání zápasů a vlastních zápasů**

  Dnešní hry běžně umožňují hráčům tvořit skupiny a v rámci celé skupiny se pohybovat mezi zápasy. Hra Inpemo dovolí hráčům vytvářet maximálně čtyřčlenné skupiny. Ve skupinách se bude možné připojit do systému vyhledávání, který v takovém případě zajistí, že skupina bude do herní relace přesunuta jako celek. Podobně se bude možné připojit i do vlastních zápasů.

5.1.3 Hratelnost a herní módy

Funkční požadavky

- **FGG01: Hráč uvidí hru z pohledu top-down**
- **FGG02: Hráč se bude pohybovat pomocí kláves WSAD**

  Pohyb postavy bude do osmi směrů a bude ovládán pomocí kláves WSAD.

- **FGG03: První herní mód bude *Last Team Standing***

  Mód *Last Team Standing* spočívá v postupném zmenšování herní mapy, přičemž se hráči snaží, aby byli posledním přeživším. Používají své schopnosti k vystrčení ostatních hráčů mimo bezpečnou oblast.

- **FGG04: Hráči si budou vybírat postavu před začátkem zápasu**

Nefunkční požadavky

- **NGG01: Herní módy musí být dobře rozšiřitelné**

  Přestože herní návrhový dokument specifikuje pouze jeden herní mód, jeho autoři plánují přidání dalších herních módu. Herní módy by tedy měly být navrženy tak, aby bylo možné pohodlně a co možná nejsnadněji přidat další.

- **NGG02: Kamera by měla zabírat hráčovu postavu a umožnit hráči dobrý přehled o aktuální situaci**

  Vzhledem k tomu, že se jedná o velice rychlou a akční hru, je potřeba, aby hráč viděl tu část herního světa, která je pro něj v danou chvíli nejvíce relevantní. Kamerový systém by tento fakt měl reflektovat.

5.1.4 Avataři

Herní postavy ve hře Inpemo se nazývají avataři. Každý avatar má určitou energii, podle které se řadí do jednoho ze tří typů avatarů.

Funkční požadavky

- **FA01: Každý avatar bude mít čtyři schopnosti**

  Avataři mají čtyři schopnosti, přičemž tři jsou unikátní pro každého avatara. Čtvrtá schopnost je přiřazena dle energie, kterou avatar nabývá.

- **FA02: Avataři budou podporovat tři typy energií**

  Hra bude obsahovat tři unikátní typy energií.

Nefunkční požadavky

- **NA01: Pohyb avatarů by měl využívat techniky ke snížení pocitové latence**

  K zajištění co nejvíce příznivého uživatelského zážitku by měly být použity techniky pro snížení latence a predikování pohybu na straně klienta.

- **NA02: Avataři by měli být snadno rozšiřitelní**

  Přidání nového avatara by mělo být snadné a nemělo by to přímo vyžadovat zásah do herního kódu.

5.1.5 Schopnosti

Funkční požadavky

- **FS01: Schopnosti budou primárně odstrkovat nepřátelské avatary, ale měly by podporovat i jejich poškození**

  Primární mechanika hry je odstrkování, nicméně schopnosti by měly podporovat i udělování poškození, vzhledem k tomu, že další herní módy poškození mohou využívat.

- **FS02: Schopnosti budou mít různorodé vizuální efekty a dopady**

  Hra bude obsahovat různé typy schopností, které se budou lišit jak vizuálními efekty, tak i dopady.

- **FS02: Každá schopnost může mít jiný typ míření**

  Ve hře bude existovat více typů míření (oblastní, dovednostní atd.).

Nefunkční požadavky

- **NS01: Schopnosti by měly být snadno rozšiřitelné**

  Jelikož hra je postavena na schopnostech, zaintegrování nového způsobu míření i nové schopnosti by mělo být snadné.

5.2 Výběr síťového modelu

Vzhledem k výběru Unreal Engine 4 jakožto vývojového herního enginu je zřejmé, že hra musí využívat topologii client-server. Zůstává otázka, zda budou využity dedikované nebo listen servery.

Z podstaty hry by zcela jistě byly lepší servery dedikované. Serverové instance by měly být maximálně stabilní s co nejmenší odezvou, pro což se dedikované servery hodí výrazně více než listen servery. Jejich další výhodou je fakt, že herní relace není ohrožena opuštěním některého z hráčů. Pokud v případě listen serveru herní relaci opustí host, relace zaniká.

Dedikované servery musí být někde nasazeny. Pro tento účel je možné použít vlastní hardware, nicméně to je pro malá herní studia téměř nereálné. Vyžaduje to fyzicky dopravit hardware na různá místa po celé Evropě (v případě, že je žádoucí dostupnost serverů pouze v Evropě) a celou takovou infrastrukturu spravovat, což je finančně i časově velmi náročné. Alternativou k vlastnímu hardwaru je nasazení serverů do cloudu. Tento přístup je výrazně méně pracný. Například společnost Amazon nabízí přímo pro hostování herních dedikovaných serverů produkt *Gamelift* [42].

Gamelift má dva druhy ceníků - levnější *spot pricing* a dražší *on-demand*. Levnější varianta má ovšem velkou nevýhodu. V případě, že infrastruktura Amazonu potřebuje více prostředků pro ostatní procesy, může dané instance vypnout. Jinými slovy se může stát, že herní servery budou v průběhu hraní vypnuty a hráčům tak zanikne herní relace. Z tohoto důvodu se pro hru způsob *on-demand* hodí více. Na základě ukázkového příkladu [43] je možné si spočítat přibližnou měsíční cenu. Při výpočtu je brána instance „c4.xlarge“ (4 vCPU a 7.5 GiB paměti) s odhadem, že zvládne pět herních serverů po šesti hráčích[11]. Tento odhad může být nepřesný, jelikož záleží na implementačních detailech hry. Také pro zjednodušení výpočtu bude počítáno s tím, že počet hráčů bude konstantní a pouze v čase 10:00 - 0:00. Za těchto předpokladů vychází měsíční cena pro 120 hráčů přibližně 300 USD.

Z finančních důvodu autora práce není možné dedikované servery použít. Proto budou využity listen server i přes některé jejich špatné vlastnosti. Nicméně při vývoji bude kladen důraz na psaní kódu tak, aby v budoucnosti bylo možné v případě potřeby přejít na servery dedikované s minimálním zásahem do kódu.

[11] Herní relace nemusejí být vždy zaplněné na maximální počet osmi hráčů.

5.3 Výběr podpůrných služeb

Zadání práce (nefunkční požadavek **NG02**) udává povinnost použití podpůrných služeb od Ing. Petra Nohejla. Tyto služby budou dále označovány jako sada služeb „BVision“. V době vytváření zadání bylo plánováno, že služby BVision budou použity společně se službami *Epic Online Services*.

Většinu *Epic Online Services* služeb je možné používat bez Epic účtů. Stačí přihlášení pomocí *Connect Interface* [44], které umožňuje využít různé poskytovatele identity (Google, Steam, Discord atd.), včetně vlastní OpenID autentizační/autorizační služby. Ovšem nachází se mezi nimi i pár služeb, které striktně vyžadují přihlášení přes Epic účet. Mezi ně patří služba spravující seznamy přátel.

Pro splnění funkčních požadavku **FMS03**, **FMS04** a **FMS05** je nutné integrovat službu spravující seznamy přátel z *Epic Online Services*, která ovšem vyžaduje přihlášení skrze Epic účet. V tento moment by platforma BVision musela poskytovat možnost přihlášení pomocí Epic účtu, jelikož je nežádoucí, aby se hráči museli přihlašovat pomocí dvou různých účtů (Epic a BVision). Také by bylo nutné mapovat uživatelské identifikátory BVision na Epic a naopak.

Služby BVision výše zmíněnou možnost přihlášení neposkytují. Navíc v nedávné době vývojářský tým *Epic Online Services* vydal nové OpenID API [45], pomocí kterého je možné ověřit Epic přístupový token. Je tedy možné uživatele ověřit jakoukoliv vlastní externí službou.

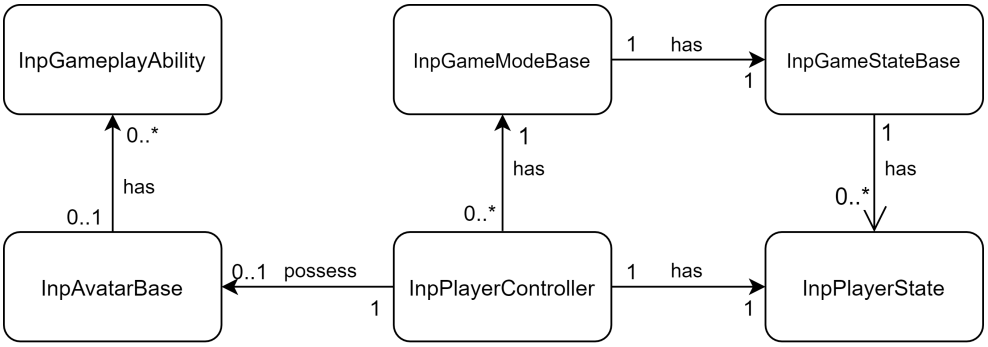
Ověření Epic přístupového tokenu externí službou nebylo v době vytváření zadání možné. V aktuální situaci použití BVision poskytovatele identity bohužel nedává smysl a to i za předpokladu, že by do platformy BVision byla přidána možnost přihlášení pomocí Epic účtu.

Po konzultaci s vedoucím práce bylo rozhodnuto, že služby BVision nebudou ve výsledném prototypu použity a místo nich budou použity pouze služby *Epic Online Services*. V době vytváření zadání neexistovalo výše zmíněné API a zadání práce tak nereflektuje aktuální situaci. Nicméně pro jeho splnění bude vytvořena integrace služeb BVision, jejíž zdrojové kódy budou součástí přiloženého USB disku.

5.4 Návrh základních tříd

Na základě enginem poskytovaných tříd byly navrženy vlastní hlavní třídy, které jsou ve většině případů jejich potomky. Všechny nové třídy začínají předponou „Inp“, aby se předešlo případným kolizím s názvy ostatních vestavěných tříd, které jsou již součástí enginu.

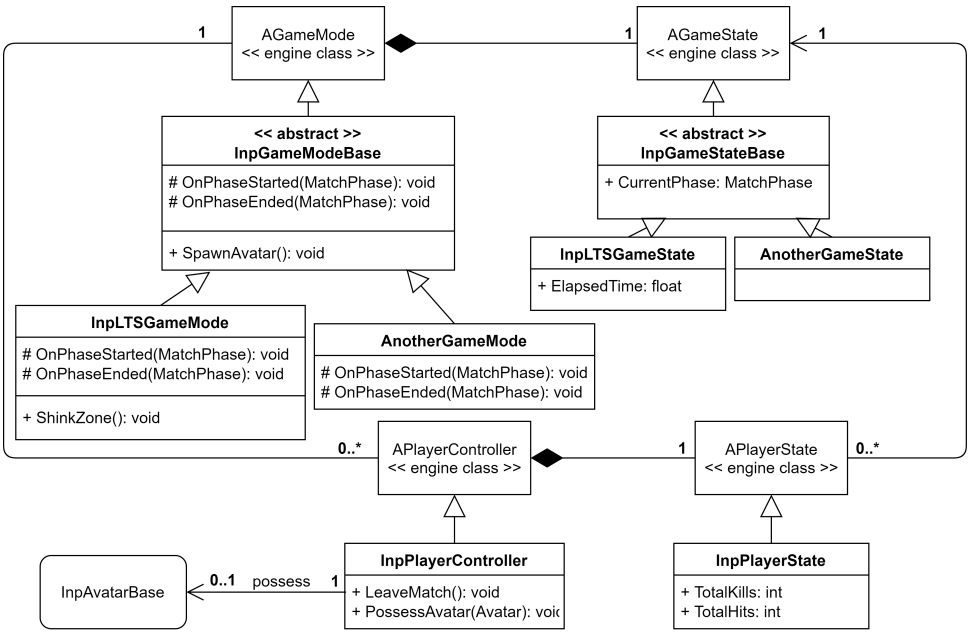
Zjednodušený pohled na hlavní herní systém zobrazuje diagram 5.1. Dílčí části tohoto systému budou rozkresleny a vysvětleny v podsekcích níže.



Obrázek 5.1: Zjednodušený pohled na hlavní herní třídy a jejich vztahy

5.4.1 Herní módy

Dle požadavku **NGG01** je kladen důraz na snadné přidání případných nových herních módů. Základ pro všechny herní módy tvoří třída *InpGameModeBase*, která se stará o společnou logiku všech herních módů. Do této společné logiky se řadí zejména rozfázování zápasu do několika částí a řízení některých z nich.



Obrázek 5.2: Diagram tříd popisující herní módy

Fází zápasu je celkem pět, přičemž aktuální fáze bude uložena ve třídě *InpGameStateBase* tak, aby k ní měli všichni klienti přístup. Každý herní mód bude moci přetížit případnou výchozí logiku konkrétních fází.

- **WaitingForAllPlayers** V této fázi se zápas nachází ihned po načtení mapy. Jejím účelem je počkat na všechny hráče, jelikož doba potřebná pro načtení mapy se pro každého z nich může lišit.
- **InProgress** Při přechodu do fáze *InProgress* je zajištěno, že všichni hráči mají již načtenou mapu a zápas je zahájen. Tato fáze nemá žádnou výchozí logiku. Předpokládá se, že logika herních módů bude velmi specifická a proto zde není snaha ji jakkoliv generalizovat. Herní módy jsou také zodpovědné za přechod do fáze *MapResult*.
- **MapResults** Fáze reprezentující konec zápasu a vyhlášení výsledků.
- **LeavingMap** Poslední fáze zápasu, dává možnost herním módům vykonat poslední logiku před opuštěním mapy.
- **Aborted** Fáze značící fatální chybu v zápase. Dává herním módům šanci na chybu reagovat. Na konci této fáze je opuštěna herní mapa.

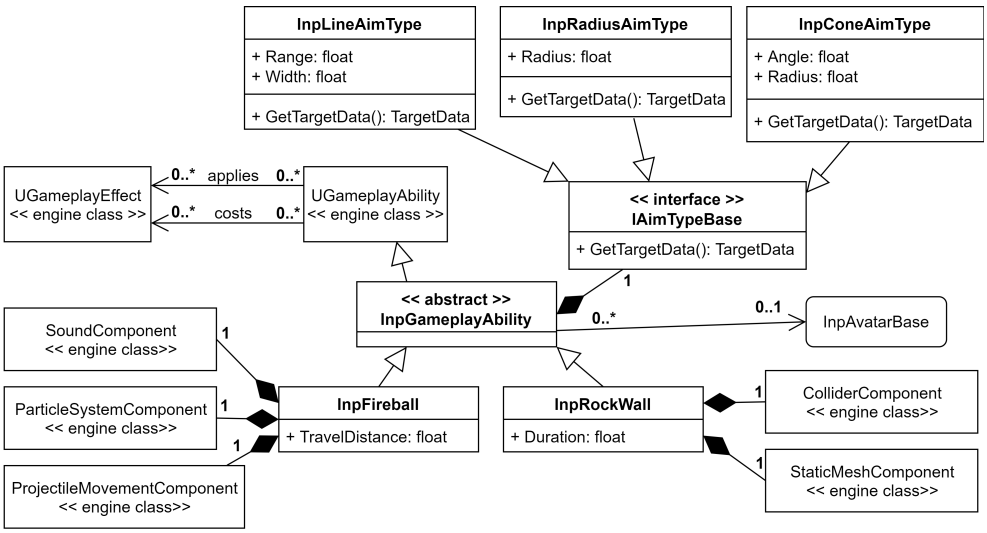
Vzhledem k hernímu návrhovému dokumentu a konzultaci s jeho autory se nepočítá s tím, že by byly potřeba rozdílné třídy *PlayerController* a ukládaná hráčská data pro různé herní módy. Z tohoto důvodu jsou navrženy třídy *InpPlayerController* a *InpPlayerState*, které mohou být přímo využity všemi herními módy. V případě specifických potřeb nového herního módu je možné od těchto tříd dědit a definovat tak dedikované chování pro konkrétní herní mód, nicméně takový případ by měl být ojedinělý.

5.4.2 Schopnosti

Celý ekosystém schopností je postaven na *Gameplay Ability* systému. Výchozí abstraktní třídou všech schopností je *InpGameplayAbility*. Ta v momentální situaci neobsahuje žádnou funkcionalitu navíc oproti třídě *UGameplayAbilty*, nicméně je zde připravena pro případ, že v budoucnu bude potřeba generická funkcionalita společná pro všechny schopnosti.

Dopady schopností se definují pomocí třídy *GameplayEffect*. Každá schopnost může mít neomezené množství instancí této třídy, přičemž každá instance reprezentuje jeden dopadový efekt (například odstrčení nebo zpomalení). Stejným způsobem se třída používá k definování ceny kouzla (které atributy a jaké hodnoty jsou potřeba k použití schopnosti).

Hra poskytuje několik způsobů míření, ty jsou zobecněny pomocí rozhraní *IAimTypeBase*. Každý způsob míření musí toto rozhraní implementovat. Všechny schopnosti mají právě jeden způsob míření.



Obrázek 5.3: Diagram tříd popisující schopnosti

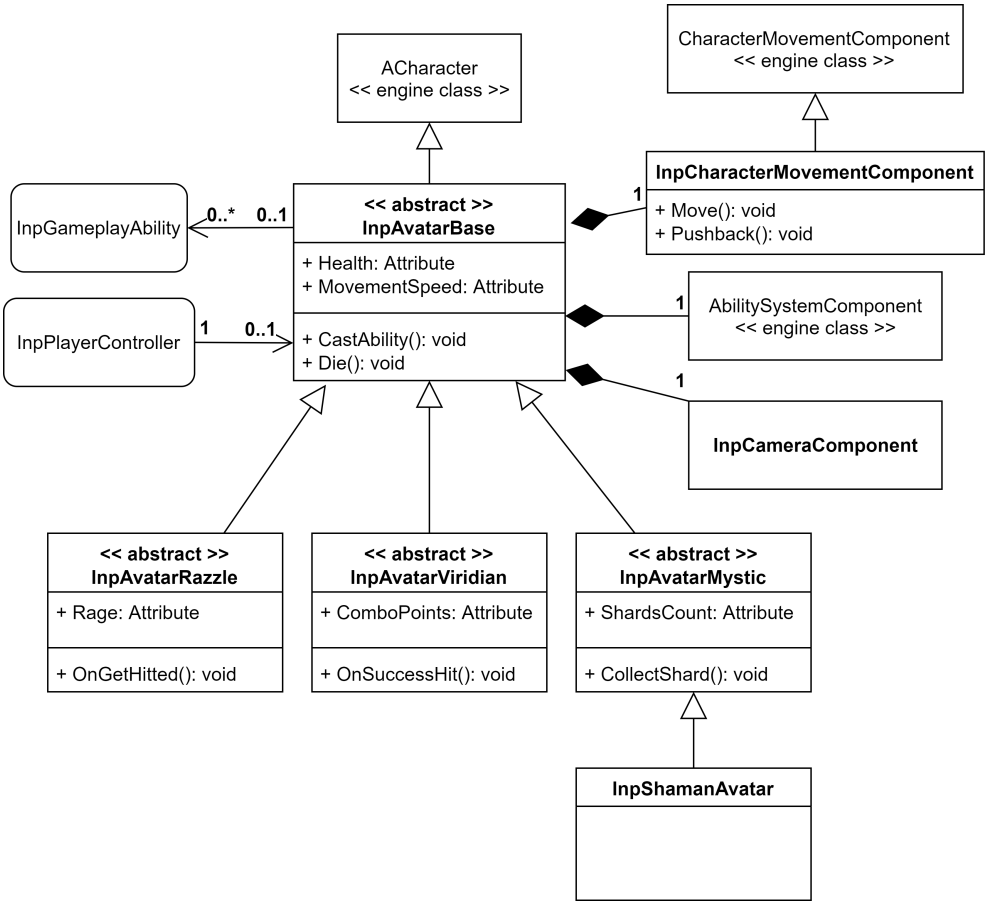
Samotné schopnosti jsou pak rozšiřovány o různé komponenty dle jejich povahy. Například schopnost „Fireball“ (ohnivá koule letící vybraným směrem) potřebuje komponentu pro přehrávání zvuků, komponentu pro zobrazování vizuálních efektů (jiskřiček a plamenů) a komponentu, která se bude starat o pohyb. Díky komponentovému návrhu je možné vytvářet nepřeberné množství schopností, které na sobě nebudou závislé, a zároveň při jejich vývoji používat funkcionality (komponenty), které již byly vytvořeny dříve.

5.4.3 Avataři

Hlavní vlastností všech avatarů je jejich energie. Avatar může oplývat vždy pouze jednou energií. Ta částečně specifikuje jeho herní styl. Ve hře se nacházejí tři různé energie.

- **Razzle** Energie je čerpána z obdrženého poškození nebo odhození.
- **Viridian** Za každou úspěšně trefenou schopnost je obdržen jeden bod energie.
- **Mystic** Po úspěšném zasažení schopností jsou v okolí avatara vygenerovány fragmenty, které může sebrat a tím tak obnovovat svou energii.

Společné chování a vlastnosti všech avatarů definuje abstraktní třída *InpAvatarBase*. Od této třídy dědí další tři abstraktní třídy, které reprezentují jednotlivé energie a obsahují vlastnosti a logiku pro danou energii. Potomci těchto tříd jsou již konkrétní avataři.



Obrázek 5.4: Diagram tříd popisující avatary

Základní mechanikou avatarů je pohyb. Komponenta *CharacterMovementComponent* obsahuje velmi robustní řešení pohybu, ovšem neposkytuje některé vyžadované pohyby. Z toho důvodu je vytvořená nová komponenta *InpCharacterMovementComponent*, která z ní vychází a dodefinovává potřebné funkcionality.

Druhou důležitou mechanikou je používání schopností. Avataři dle návrhového dokumentu mají čtyři schopnosti, nicméně z hlediska návrhu tříd je počítáno s neomezeným počtem. Pro správné zaintegrování avatarů do *Gameplay Ability* systému je potřeba, aby vlastnili komponentu *AbilitySystemComponent*.

Požadavek **NGG02** pojednává o tom, že by herní kamera měla zabírat herní postavu. Na základě toho se autor práce domnívá, že ideálním řešením je umístění kamerové komponenty přímo na avatara, díky čemuž bude stále v záběru.

5.5 Návrh systému pro vytváření zápasů

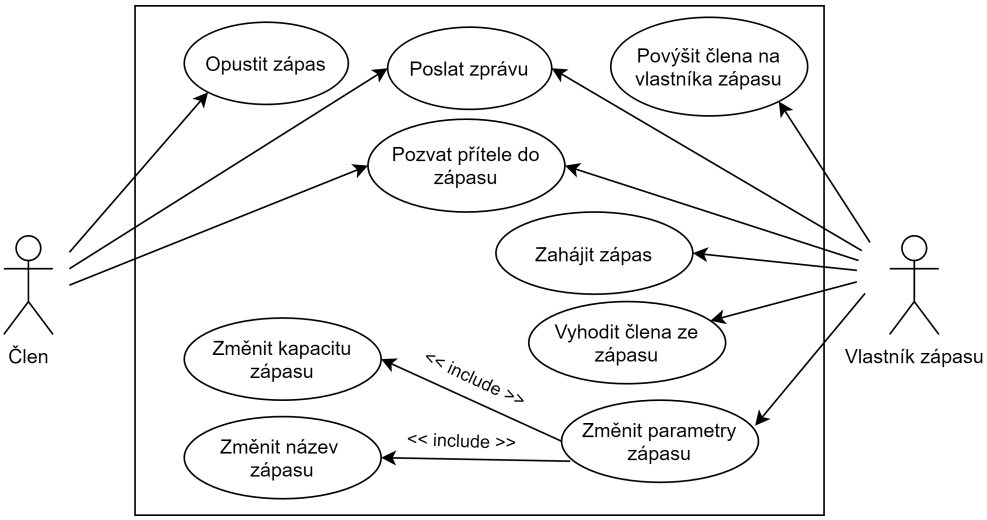
Systém pro vytváření a vyhledávání zápasů musí využívat podpůrnou službu, která bude zápasy a hráče registrovat. Pro tento účel budou využity služby *Epic Online Services*, konkrétně *Lobby Interface* [46]. Toto rozhraní funguje na principu místností. Hráči mohou místnosti zakládat a připojovat se do nich. Zároveň místnostem i hráčům v nich mohou být přiřazeny libovolné parametry, které je možné následně i editovat.

Požadavky **FMS01** a **FMS02** žádají dva různé přístupy vytváření zápasu. Oba budou detailněji rozebrány v podsekcích níže.

5.5.1 Vlastní zápasy

Tento způsob tvoření zápasu je prakticky identický s tím, jak funguje *Lobby Interface*. Zápas lze chápat jako místnost, do které se připojují hráči a která může mít přiřazené jednotlivé parametry (např. název, kapacitu, herní mapu).

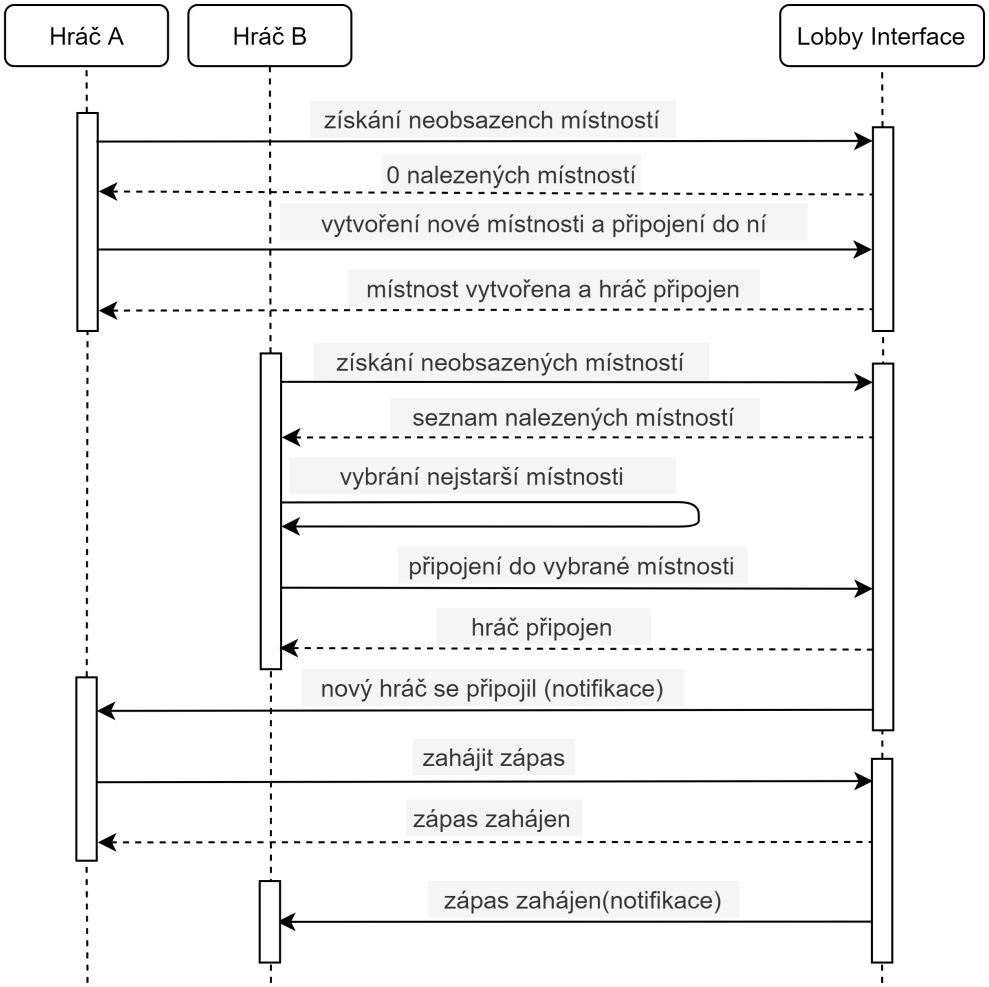
Hráči se mohou libovolně připojovat do ještě zcela neobsazených místností, případně do nich mohou být pozváni jakýmkoliv hráčem, který v místnosti je již připojen. Po připojení do zápasu hráč zastává roli člena (*member*). Oproti tomu hráč, který zápas vytvořil, zastává roli vlastníka zápasu (*lobby owner*). Ten má navíc některé pravomoci, například vyhození člena ze zápasu nebo změnu parametrů zápasů. Vlastník také může předat svou roli některému členovi, čímž se sám stane členem. Přehled základních případů užití vyjadřuje diagram 5.5.



Obrázek 5.5: Diagram případů užití pro role ve vlastním zápasu

5.5.2 Systém vyhledávání

Při vytváření návrhového dokumentu bylo počítáno s tím, že systém vyhledávání bude využívat podpůrnou službu od Ing. Petra Nohejla, která bude hře přizpůsobena na míru. To bohužel vzhledem k již zmíněným okolnostem není možné. Nicméně při využití jisté logiky bežící na pozadí všech instancí hry je možné použít *Lobby Interface* k dosažení podobné funkcionality.



Obrázek 5.6: Sekvenční diagram systému vyhledávání - zjednodušený případ pro dva hráče

Diagram 5.6 popisuje logiku bežící na pozadí (diagram je zjednodušen, zápas je spuštěn pouze se dvěma hráči). Celý proces je založen na stejném principu jako tvoření vlastních zápasů s tím, že celá logika vytváření zápasu a připojení do něj je před hráčem skryta a automatizována na pozadí. Každý

hráč při spuštění vyhledávání požádá *Lobby Interface* o všechny místnosti, které nejsou zcela naplněny. Pokud žádná taková místnost není nalezena, hráč požádá o vytvoření nové místnosti a stane se jejím vlastníkem. Vlastníkova zodpovědnost je kontrolovat naplnění místnosti a v momentě, kdy je místnost plná, zápas odstartovat. Pokud nějaká taková místnost existuje, hráč se připojí do té s nejstarším datem založení a pokud je takových kandidátů více, vybere tu, která má nejvíce hráčů. Díky tomu by měly být prioritně vybírány místnosti, které čekají nejdéle.

Tento mechanismus bude funkční v případě, že do vybírání vhodného zápasu nebudou zahrnuty další parametry odvíjející se od statistik hráčů (například počet odehraných her). Pokud by hra v budoucnu potřebovala, aby systém vyhledávání zápasů takové údaje využíval, bude potřeba centrální autorita, která za vybírání vhodných spoluhráčů bude zodpovídat. Bude tedy nutné vytvořit vlastní podpůrnou službu a integrovat ji s *Epic Online Services* službami. Aktuální systém nechává veškerou zodpovědnost na jednotlivých herních zařízeních a to by v případě použití statistik nemuselo být bezpečné.

5.6 Systém pro správu skupin

Tento systém je principem velmi podobný vlastním zápasům. Hráč může založit skupinu, čímž se stane jejím vlastníkem a může do ní pozvat další hráče. Problém nastává v momentě, kdy se celá skupina chce připojit do zápasu. Je potřeba zaručit, že všichni členové skupiny budou do zápasu připojeni v rámci jedné transakce, tedy že budou připojeni jako celek pomocí jednoho požadavku.

V momentální situaci toho není možné docílit čistě pomocí podpůrných služeb *Epic Online Services*. Ty zatím neposkytují žádné rozhraní, které by řešilo hráčské skupiny. Zároveň ani *Lobby Interface* nepodporuje připojení více hráčů do jedné místnosti v rámci jedné transakce. To by se ale mělo změnit v momentě, kdy vývojáři vydají oficiální podporu pro skupiny, která je již na cestě. Dle veřejného vývojového plánu týmu vyvíjejícího tyto podpůrné služby [47] se aktuálně systém pro skupiny nachází v návrhové fázi a v blízké době by měl být přidán.

Skupiny nejsou prioritním požadavkem ani kritickou částí hry, díky čemuž mohou být snadno do hry dodány, aniž by bylo nutné přepracovat zásadní části hry. Z těchto důvodů autor práce vyčká na oficiální podporu skupin službami *Epic Online Services*. Z hlediska této práce to znamená, že funkcionalita skupin nebude ve výsledném prototypu přítomna.

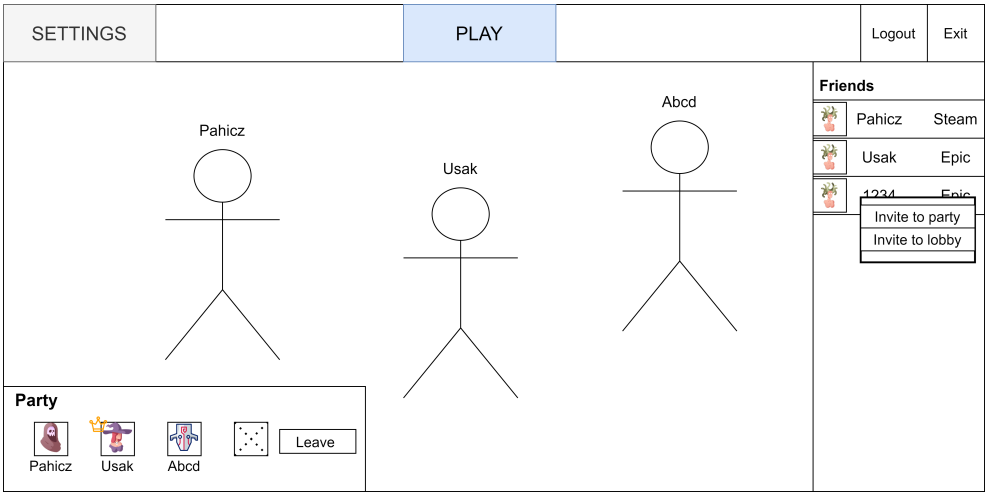
5.7 Uživatelské rozhraní

Hra se skládá z několika různých obrazovek, přičemž mezi hlavní tři patří hlavní menu, výběr avatara a pak samotná herní obrazovka.

Hlavní menu (obrázek 5.7) obsahuje čtyři hlavní tlačítka. Pomocí tlačítka *Play* se hráč dostane do rozcestníku, kde může vybrat, zda chce vyhledat zápas nebo přejít do přehledu vlastních her. Tlačítko *Settings* hráče přesune do obrazovky, kde může upravovat jednotlivá nastavení hry (zvuky, grafika atd.). Poslední dvě tlačítka *Logout* a *Exit* slouží k odhlášení hráče a vypnutí hry. Při vypnutí hry není vynucené odhlášení. Hra si drží potřebné obnovovací tokeny tak, aby hráč mohl být při příštím spuštění hry automaticky přihlášen. Odhlášení herní program nevypne, nicméně vynutí smazání veškerých přístupových i obnovovacích tokenů.

Dále se zde nachází přehled přátel. U každého přítele je vidět jeho jméno a označení, zda se jedná o přítele z platformy *Steam* nebo *Epic Games Store*. Po kliknutí pravým tlačítkem na přítele se otevře kontextové menu, ve kterém se nachází tlačítko na pozvání přítele do skupiny a v případě, že je hráč připojen ve vlastním zápase, je zde také tlačítko na pozvání přítele do vlastního zápasu.

Poslední důležitou částí hlavního menu je přehled skupiny. Ten je dostupný pouze v případě, že se hráč aktuálně nachází ve skupině. Kromě jednotlivých členů se v přehledu nachází také tlačítko *Leave*, pomocí kterého je možné skupinu opustit.

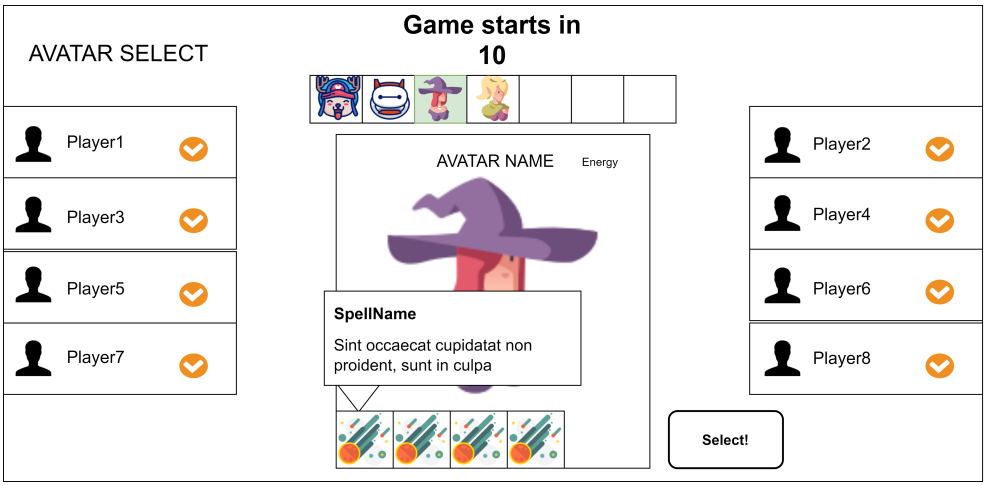


Obrázek 5.7: Návrh obrazovky hlavního menu

Obrazovka výběru avatara (obrázek 5.8) má v horní části informativní text s odpočtem, který hráče informuje o aktuálním dění. Hned pod ním se nachází seznam avatarů. Po kliknutí na jednoho z avatarů je uprostřed obrazovky zobrazen jeho detail. V detailu je vidět avatarovo jméno, energie a vizuální reprezentace. Jsou zde také jeho schopnosti, které při přejetí kurzorem zobrazí svůj detail s popisem.

Po obou stranách jsou vypsány kartičky všech hráčů. Kartička obsahuje

hráčovo jméno a stav připojení, který indikuje, zda je hráč úspěšně připojen do výběru. Poté, co si hráč vybere svého avatara pomocí tlačítka *Select*, se miniatura vybraného avatara zobrazí vedle hráčova jména.



Obrázek 5.8: Návrh obrazovky výběru avatara

Na obrázku 5.9 je vidět návrh hlavní herní obrazovky. Při jejím návrhu byl kladen důraz na to, aby uživatelské rozhraní zabíralo co nejméně místa. Nejdůležitější částí je spodní lišta obsahující všechny důležité informace o avatarovi, za kterého hráč hraje. Je zde možné vidět avatarovy aktuální body zdraví, body energie a také jeho schopnosti. Pokud hráč čeká na obnovení schopnosti, je přes ní zobrazen čas, který zbývá do obnovy. Po stranách jsou zobrazeni avataři spoluhráčů. Jejich přeškrtnutí indikuje, že jsou již vyřazeni ze zápasu.

Obrázek 5.9: Návrh hlavní herní obrazovky

Kapitola 6

Implementace

6.1 Integrace online služeb

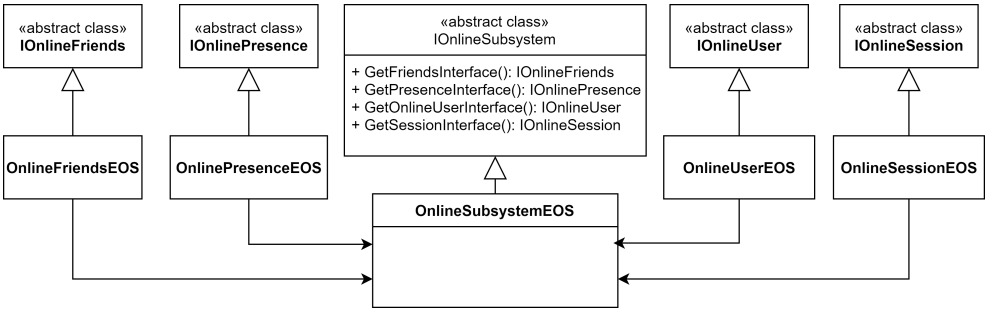
Jak již bylo zmíněno dříve, podpůrné služby by se do hry vyvíjené v UE4 měly integrovat pomocí rozhraní, která jsou součástí systému *OnlineSubsystem*. Tento postup bude při implementaci hry dodržen.

Pro zaintegrování nové sady služeb pomocí tohoto systému je potřeba vytvořit nové třídy implementující všechna *IOnline* rozhraní[12], která budou hrou využívána. Hlavní z nich jsou *IOnlineFrieds*, *IOnlinePresence*, *IOnlineUser* a *IOnlineSession*, která slouží ke správě seznamů přátel, informacím o statusu hráče, získání uživatelských metadat a správě herních relací. Dále je nutné vytvořit třídu, jež bude představovat samotný *OnlineSubsystem*. Ta musí být potomkem třídy *FOnlineSubsystemImpl* a její hlavní zodpovědností je správa a poskytování instancí právě těchto nových tříd implementující *IOnline* rozhraní. Vzhledem k tomu, že se jedná o služby *Epic Online Services*, dává smysl tento systém pojmenovat *OnlineSubsystemEOS*.

Autor práce v rámci počátečního prozkoumávání enginu a podpůrných služeb zkusil vytvořit částečnou integraci serveru *Nakama* [48]. Byla vytvořena implementace rozhraní *IOnlineIdentity* a *IOnlineChat*, která slouží k přihlášení/odhlášení uživatele a zasílání zpráv. Tato rozhraní nejsou nijak významně rozsáhlá, spíš se řadí mezi ta jednodušší. Přesto jejich implementace zabrala přibližně dvanáct člověkodní (96 hodin). Na základě toho je možné přibližně odhadnout, že implementace systému s potřebnými službami, kterých je nejméně osm, by zabrala minimálně 400 hodin, jelikož zbylá rozhraní jsou ve většině případů výrazně složitější.

S touto znalostí se autor práce rozhodl zakoupit plugin *EOS Online Subsystem* od společnosti *RedpointGames* [49]. Cena pluginu byla 120 USD. Plugin je

[12] Ve skutečnosti se jedná o abstraktní třídy. Všude jsou ovšem označovány jako „interfaces“, tak je toto názvosloví dodržováno i v této práci.



Obrázek 6.1: Hierarchie *OnlineSubsystem* rozhraní

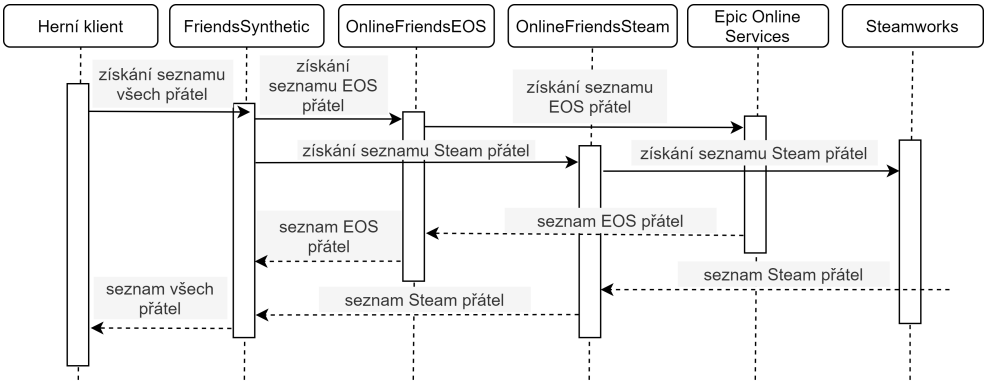
dostupný i ve volně dostupné verzi, bohužel ta obsahuje pouze binární soubory bez zdrojových kódů.

6.1.1 EOS Online Subsystem a OnlineSubsystemSteam

EOS Online Subsystem plugin obsahuje implementaci některých *IOnline* rozhraní a poskytuje tak integraci většiny EOS služeb. Pro získání přístupu k EOS službám je potřeba vytvořit nový produkt v portálu *EOS Developer Portal*, nastavit vyžadované poskytovatele identit (Epic a Steam) a vytvořit v produktu aplikaci reprezentující herního klienta. Na základě toho portál poskytne důvěryhodné informace, pomocí kterých je nutné inicializovat EOS SDK při každém spuštění hry. Plugin se o inicializaci SDK postará, stačí všechny potřebné informace zadat ve správném formátu do konfiguračního souboru *DefaultEngine.ini*.

Pokud by hra využívala pouze služby EOS, výše zmíněné nastavení stačí. Ovšem hra Inpemo má využívat společně s nimi i služby *Steamworks*. *OnlineSubsystemSteam* je plugin integrující služby *Steamworks* a je zabudován přímo v UE4. Ten pro svou inicializaci vyžaduje *AppId*, které představuje identifikátor aplikace uvnitř platformy Steam. Vzhledem k tomu, že platforma Steam vyžaduje pro získání vlastního *AppId* registraci aplikace, pro což musí být registrujícím subjektem firma a zaplatit poplatek 200 USD, bude prozatím použito *AppId* 480, které slouží pro testovací účely.

Jelikož vývojáři UE4 se soustředí na vytvoření vlastního pluginu pro integraci služeb EOS (viz veřejný EOS plán [47]), *OnlineSubsystemSteam* není moc dobře udržovaný a spravovaný. V kompilované verzi má problémy s inicializací, konkrétně při vypnutí hry si velmi často smaže konfigurační soubor „steam_appid.txt“ obsahující *AppId*. Kvůli tomu nemůže být při příštím spuštění úspěšně nastartován. Pro vyřešení tohoto problému byl vytvořen nový modul *EarlyConfigWorkaround*, který načte *AppId* z konfiguračního souboru *DefaultEngine.ini* a ještě před samotnou inicializací systému *OnlineSubsystemSteam* soubor vytvoří.



Obrázek 6.2: Sekvenční diagram získání přátel pomocí rozhraní *FriendsSynthetic*

EOS Online Subsystem podporuje využití více systémů *OnlineSubsystem* najednou. Chová se jako primární systém a ostatní jsou jím využívány. Celý princip je postaven na vytvoření nového nadřazeného integračního rozhraních. Jako příklad bude bráno rozhraní *IOnlineFriends* a požadavek na získání všech přátel přihlášeného uživatele. Herní klient použije nadřazené integrační rozhraní *FriendsSynthetic* k získání všech přátel. To pak využije obě implementace systémů *OnlineSubsystem* pro přístup k příslušným implementacím rozhraní *IOnlineFriends*, konkrétně *OnlineFriendsEOS* a *OnlineFriendsSteam*. Tato rozhraní použije k odeslání požadavku pro jednotlivé platformy. Poté, co platformy vyřídí požadavky, *FriendsSynthetic* výsledky sloučí a vrátí je hernímu klientovi.

Plugin také dává k dispozici funkci *AutoLogin* poskytující vývojářům funkci automatického přihlášení. Přihlášení lze přepnout do jednoho ze tří módů [50].

- **EASRequired** Přihlášení striktně pouze pomocí Epic účtu, uživatel musí mít Epic účet nebo si ho musí vytvořit. Plugin zkusí přihlášení pomocí Epic Game Store služby na pozadí a v případě neúspěchu otevře webový prohlížeč s přihlášením.
- **EASOptional** Uživatel nemusí mít Epic účet, plugin nejdříve zkusí přihlášení pomocí Epic Game Store služby, poté pomocí všech dostupných platforem na pozadí (Steam, Discord atd.) a v případě neúspěchu otevře webový prohlížeč s přihlášením.
- **NoEAS** Uživatelé se nemohou přihlásit pomocí Epic účtu. Plugin zkusí přihlášení pomocí všech dostupných platforem (kromě Epic Games Store) na pozadí. Pokud žádná není dostupná, uživatel není přihlášen.

Celá tato logika je definována ve třídě *AuthenticationGraph*, která by se vzdáleně dala přirovnat k *Behavior Tree* (strom chování). Obsahuje speciální

strukturu uzlů a podmíněných hran, přičemž se prochází od kořene až do doby, než je uživatel úspěšně přihlášen nebo do doby, kdy již nezbývají žádné nenavštívené uzly.

Hra Inpemo bude využívat přihlašovací mód *EASRequired*, protože potřebuje přístup ke všem službám EOS (uživatel musí být vždy přihlášen přes Epic účet).

6.1.2 Služby od Ing. Petra Nohejla

Služby BVision musí spolupracovat s EOS službami. Spojení těchto dvou sad služeb není předmětem této práce a je popsáno v diplomové práci Ing. Petra Nohejla [31]. Nicméně povrchově lze nastínit, že EOS autorizační služba využívá OpenID API poskytovatele identity BVision k ověření přístupového tokenu. Herní klient tedy pro přihlášení k EOS službám musí zaslat přístupový token BVision.

V rámci *EOS Online Subsystem* pluginu byl vytvořen nový modul *OnlineSubsystemEOSPlatformBVision* představující platformu BVision. Uvnitř tohoto modulu byla vytvořena nová třída *BvisionAuthenticationGraph*, jež je potomkem třídy *AuthenticationGraph*. I přes to, že bylo možné přepsat přímo zdrojové kódy pluginu a změnit tak *AuthenticationGraph* přímo v nich, autor práce raději zvolil vytvoření nových tříd obsahujících nový graf, jelikož by to potenciálně v budoucnosti mohlo přinést výrazné ulehčení práce s přechodem na nové verze pluginu.

Nový graf obsahuje pět uzlů (viz výpis kódu 6.1). Kořenový uzel *FAuthenticationGraphNodeUntil_Forever* nese funkci nekonečné smyčky. Tedy pokud po průchodu všech uzlů není uživatel přihlášen, celý proces je spuštěn znovu. Druhým v pořadí je uzel *FBVisionWebLoginNode*, který otevře uvnitř hry webový prohlížeč s přihlášením do platformy BVision. Bylo nutné použít vestavěný prohlížeč UE4 dostupný ve formě experimentálního pluginu *Web Browser* [51], protože poskytovatel identity BVision poskytuje přihlášení pouze pomocí webového formuláře a neumožňuje po dokončení přihlášení zaslat *authorization_code* pomocí WebSocketu příslušné aplikaci tak, jak to dělá například poskytovatel identity EOS. Webový formulář byl tedy vyplněn přímo v aplikaci a po dokončení přihlášení byl *authorization_code* získán z adresy prohlížeče, konkrétně *code* parametru.

```
1 TSharedRef<FAuthenticationGraphNode> FBVisionAuthenticationGraph
2   ::CreateGraph(TSharedRef<FAuthenticationGraphState> InitialState)
3 {
4   return MakeShared<FAuthenticationGraphNodeUntil_Forever>()
5     ->Add(MakeShared<FBVisionWebLoginNode>())
6     ->Add(MakeShared<FBVisionGetTokenFromCodeNode>())
7     ->Add(MakeShared<FBVisionLoginWithEOSNode>())
8     ->Add(MakeShared<FBVisionCreateIdentityStateNode>());
9 }
```

Výpis kódu 6.1: BVisionAuthetication Graph

Třetí uzel *FBVisionGetTokenFromCodeNode* pomocí *authorization_code* požádá poskytovatele identity BVision o přístupový token. S tím je již možné se přihlásit do služeb EOS. O to se postará čtvrtý uzel *FBVisionLoginWithEOSNode*. Pátý uzel *FBVisionCreateIdentityStateNode* už pouze uloží všechny obdržené tokeny a vytvoří *IdentityState* objekt, který se stará o obnovování BVision a EOS přístupových tokenů po celou dobu běhu programu.

Sada služeb BVision obsahuje dvě podpůrné služby. *Notification* služba je zodpovědná za odesílání jakýchkoliv notifikací. *Matchmaking* službu lze použít ke spárování hráčů. Původně měla být použita namísto navrženého systému vyhledávání. Integrace obou služeb byly provedeny pomocí třídy *GameInstanceSubsystem*. Jedná se o speciální třídu, jejíž instance jsou unikátní napříč programem a mají stejnou životnost a dostupnost jako instance třídy *GameInstance* [52]. Třídy *BVisionNotificationSubsystem* a *BVisionMatchmakingSubsystem* jsou tedy potomky třídy *GameInstanceSubsystem* a díky tomu jsou dostupné napříč celým programem a vytvořeny automaticky při spuštění programu.

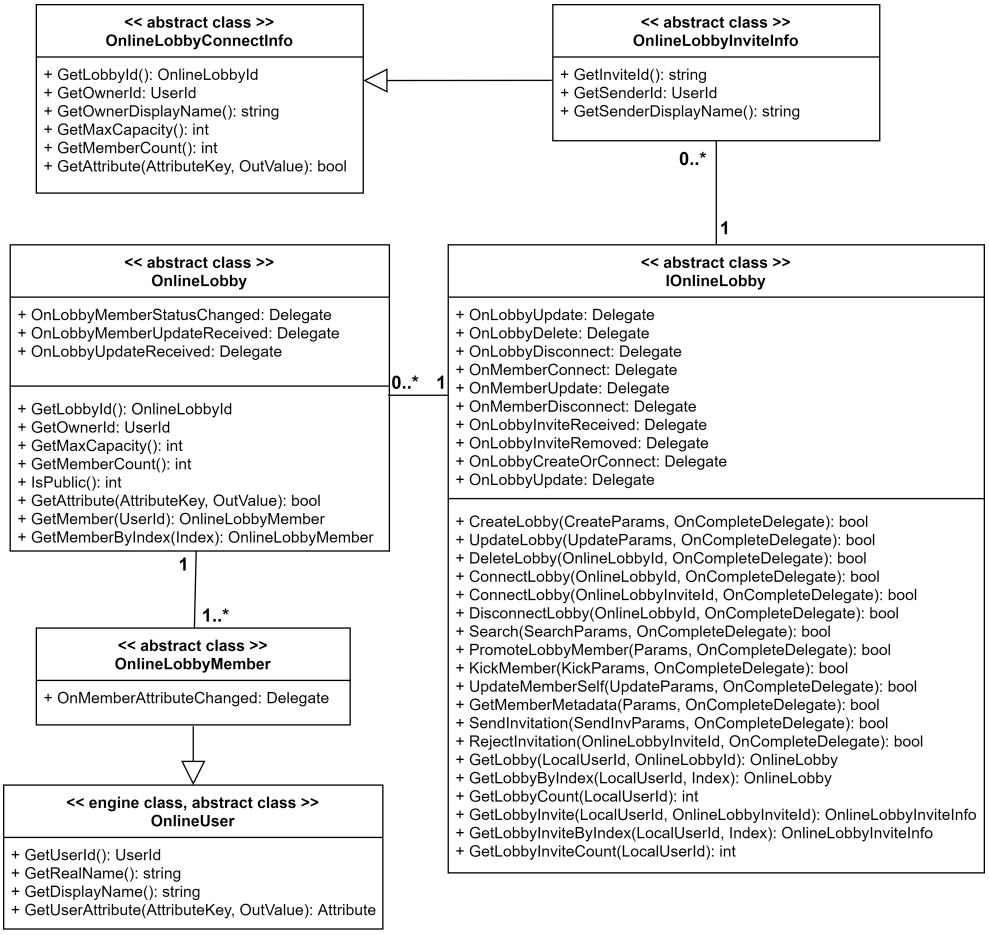
Řešení pomocí třídy *GameInstanceSubsystem* není ideální. Lepší by bylo vytvořit pro služby BVision vlastní implementaci systému *OnlineSubsystem* a propojit ji s *EOS Online Subsystem* pluginem stejně, jako je propojený systém *OnlineSubsystemSteam*. Ovšem vzhledem k tomu, že integrace *BVision* služeb nebude ve výsledné hře použita a byla vytvořena pouze pro testovací účely a splnění zadání, se autor práce rozhodl z časových důvodů pro toto řešení. Všechny zdrojové kódy modulu *OnlineSubsystemEOSPlatformBVision* jsou nahrány na přiloženém USB disku.

6.2 OnlineLobbyInterface

Systém *OnlineSubsystem* a celkově UE4 neobsahuje žádné rozhraní, které by alespoň částečně pokrývalo funkcionality EOS *LobbyInterface* rozhraní. Na základě toho bylo navrženo nové rozhraní *IOnlineLobby*, které je absolutně nezávislé na potřebách hry Inpemo a *EOS Online Subsystem* pluginu. Dodržuje veškeré standardy a postupy společné pro všechna *IOnline* rozhraní. Mezi tyto základní standardy patří především podpora pro více lokálních hráčů na jednom zařízení.

Zajištění podpory pro více lokálních hráčů není nijak složité. UE4 umožňuje hrát maximálně čtyřem uživatelům na jednom zařízení. Každý z nich je identifikován pomocí indexu v rozmezí nula až tři (včetně). Stačí tedy, aby všechna uživatelská data byla uložena vůči indexu hráče. Při přistupování k těmto datům pak musí být vždy specifikován index hráče.

Hlavní abstraktní třídou je *IOnlineLobby*. Ta obsahuje všechny důležité funkce pro správu místností a poskytuje události (delegáty), kterým je možné přiřadit různé funkce. Pokud daná událost nastane, funkce jsou zavolány. Místnost reprezentuje třída *OnlineLobby*, která má základní funkce pro zís-



Obrázek 6.3: Diagram tříd popisující *IOnlineLobby* rozhraní

kání informací o místnosti a události oznamující změny parametrů nebo členů místnosti. Místnost může mít neomezené množství členů, minimálně však jednoho. Členové jsou tvořeni třídou *OnlineLobbyMember*.

Dále se zde nacházejí ještě dvě třídy. *OnlineLobbyConnectInfo* představuje jakýsi kontextový objekt pro připojení k místnosti. Pokud uživatel vyhledává místnosti, je nežádoucí, aby si při získání jejich seznamu ihned vytvářel instance podtříd *OnlineLobby*, napojoval se tak na všechny události a získal informace o všech členech. Pro tento účel je navržena právě třída *OnlineLobbyConnectInfo*, jež neobsahuje žádné události a drží pouze takové informace, které by uživatele při vyhledávání mohly zajímat. Druhou třídou je *OnlineLobbyInviteInfo* reprezentující zaslané pozvání do místnosti od jiného uživatele. Jedná se o podtřídu třídy *OnlineLobbyConnectInfo*, která má navíc pouze informace o odesílateli pozvánky.

Diagram 6.3 z důvodu zjednodušení neobsahuje konkrétní definice delegátů. Ty ovšem mají pevně specifikované parametry. Definice některých z nich je možné vidět ve výpisu kódu 6.2.

```
DECLARE_MULTICAST_DELEGATE_TwoParams(
FOnLobbyUpdate,
const FUniqueNetId& /* UserId */,
const FOnlineLobbyId& /* LobbyId */);

DECLARE_MULTICAST_DELEGATE_TwoParams(
FOnLobbyInviteReceived,
const FUniqueNetId& /* LocalUserId */,
const FString& /* InviteId */);

DECLARE_DELEGATE_ThreeParams(
FOnLobbySearchComplete,
const FOnlineError& /* Error */,
const FUniqueNetId& /* UserId */,
const TArray<TSharedRef<FOnlineLobbyConnectInfo>>& /* Result */);
```

Výpis kódu 6.2: Definice delegátů *OnLobbyUpdate*, *OnLobbyInviteReceived* a *OnLobbySearchComplete*

Na základě třídy *IOnlineLobby* byla uvnitř *EOS Online Subsystem* pluginu vytvořena třída *OnlineLobbyInterfaceEOS*, která je pomocí EOS SDK napojena na EOS *LobbyInterface*.

6.3 Přechod do herní úrovně

Po úspěšném seskupení hráčů je potřeba založit server a všichni hráči se musí na server napojit. Serverem bude vždy jeden z hráčů, jelikož byla zvolena varianta listen serverů. Roli serveru bude vždy zastupovat vlastník zápasu. Přesun hráčů do herní úrovně lze rozdělit do několika částí.

1. Oznámení o spuštění zápasu, hráči se připojují na server
2. Všichni hráči jsou úspěšně připojeni na serveru, začíná výběr avatarů
3. Výběr avatarů byl ukončen, začíná přesun do herní úrovně
4. Všichni hráči jsou úspěšně připojeni k serveru a mají načtenou herní úroveň

UE4 nabízí dva způsoby načítání úrovní - *non-seamless* a *seamless* [53]. Hlavní rozdíl mezi nimi je ten, že *non-seamless* je blokující operace, kdežto *seamless* je neblokující. Zároveň při *non-seamless* načítání je klient vždy odpojen od serveru (zaniká instance třídy *NetDriver*) a po načtení je potřeba, aby se k serveru opět připojil. Je výrazně doporučeno používat *seamless* načítání, ovšem při zakládání serveru a prvním připojování k němu lze použít pouze *non-seamless*.

Na první problémy narazil autor práce hned při implementaci první části tohoto procesu, tedy oznámení o spuštění zápasu. EOS *LobbyInterface* nemá žádnou metodu na spuštění zápasu. Zároveň k tomu, aby se klienti mohli napojit na server, potřebují EOS *peer-to-peer* adresu serveru a port, na kterém hra poběží. UE4 používá výchozí port 7777, přičemž když není k dispozici, je inkrementován až do doby, než je nalezený port volný. Port je tedy bezpečně známý až po založení serveru, nikoli dopředu. Na základě těchto skutečností byl zvolen následující postup.

1. Vlastník vytvoří místnosti nový parametr s klíčem „State“ a nastaví mu hodnotu na „InProgress“. Stejnojmenný parametr se používá při získávání dostupných místností. Místnosti s hodnotou „InProgress“ nejsou nabízeny k připojení.

2. Vlastník zápasu zahájí *non-seamless* načítání úrovně s názvem „Avatar-Select“ a parametrem „listen=true“, který značí, že úroveň bude načtena v módu listen serveru.

3. Poté, co vlastník úspěšně dokončí načítání úrovně, vytvoří nový parametr místnosti s klíčem „ServerAddress“ a nastaví mu hodnotu ve formátu „[peer-to-peer adresa]:[port]“.

4. Hráči obdrží notifikaci o změně parametru „ServerAddress“ a na základě jeho hodnoty se připojí k serveru.

5. Poté, co se všichni hráči úspěšně připojí na server, server zahájí výběr avatarů.

6. Po skončení výběru avatarů je u všech hráčů zahájeno *seamless* načítání a pomocí toho jsou přesunuti do herní úrovně.

7. Vlastník zápasu (server) odešle požadavek na smazání místnosti.

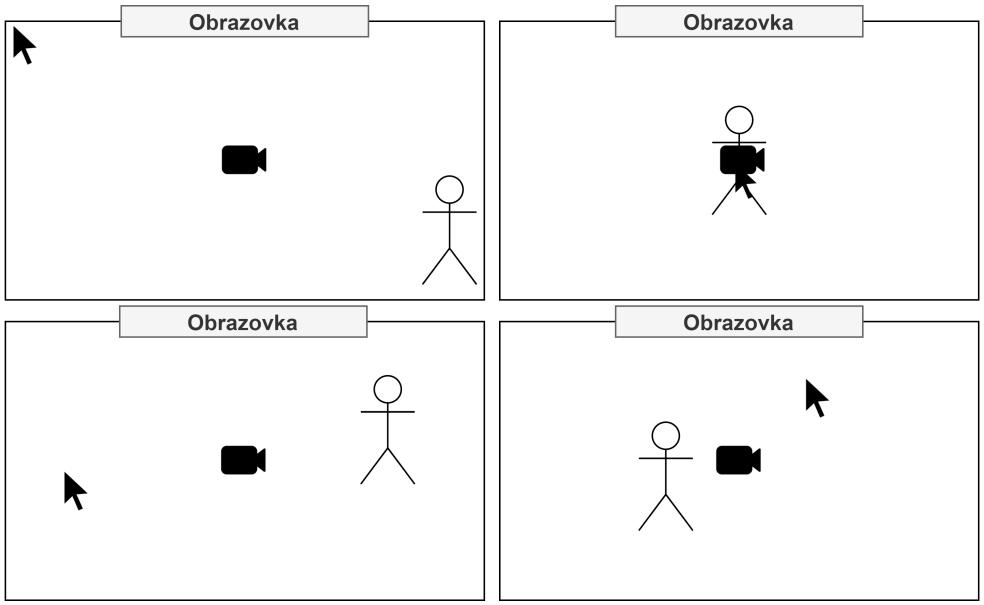
V celém tomto procesu je nutné řešit chybové stavy. Pokud se některý hráč nepřipojí do výběru avatarů nebo hru opustí v jeho průběhu, všichni hráči spustí *non-seamless* načítání úrovně hlavního menu a opět se přesunou do stejné obrazovky jako před spuštěním zápasu. Vlastník zápasu v tento moment odstraní parametry „State“ a „ServerAddress“, čímž opět umožní připojení nových spoluhráčů. Kvůli tomuto řešení chybových stavů je místnost smazána až na úplném konci celého připojovacího a výběrového procesu, aby stále existoval bezpečný stav, kam je možné se v případě chyby vrátit.

6.4 Kamera

Při návrhu bylo rozhodnuto, že kamera bude připevněna přímo na herní postavu. Poté, co byla herní postavě přidána kamerová komponenta, se vyskytly dva hlavní problémy.

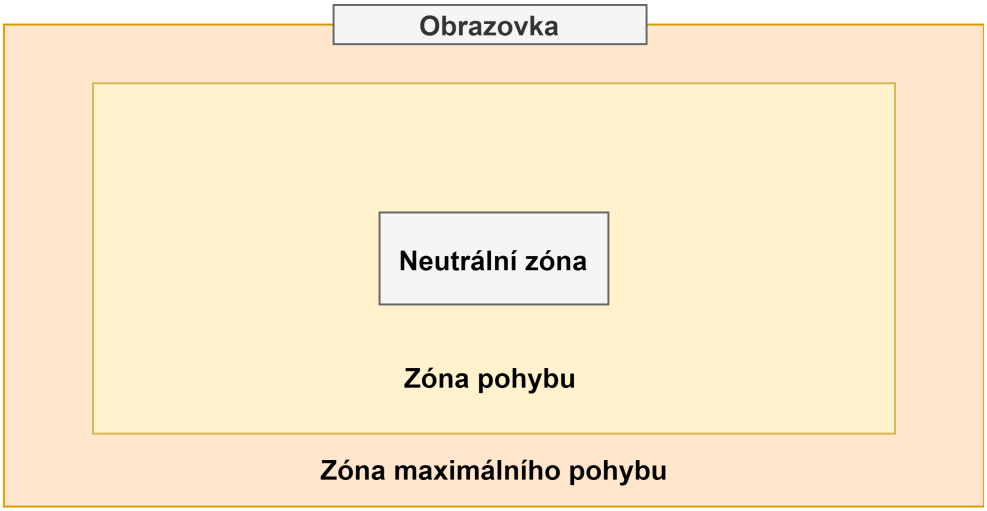
První z nich souvisí s rotací kamery. Všechny komponenty připojené na instanci třídy *Actor* automaticky kopírují její rotaci, tedy se s ní otáčejí. Takové chování není v tomto případě žádoucí. Kamera by měla mít po celou dobu zápasu stejnou rotaci, nezávislou na rotaci herní postavy. Měnit by se měla pouze lokace kamery.

Druhým z problémů bylo umístění herní postavy přímo ve středu zorného pole nezávisle na herní situaci. Již při testování v rámci vývoje byla statičnost kamery uživatelsky velmi nepřívětivá, jelikož ve hře Inpemo není hlavním bodem hráčova zájmu vždy jeho postava. Hráč potřebuje vidět i ostatní hráče, pohybovat se vůči jejich pozici, mířit schopnosti a vyhýbat se schopnostem ostatních. U velké části mechanik se hráčova pozornost často soustředí na místa v okolí kurzoru myši. Z toho autor práce usuzuje, že další body hráčova zájmu jsou právě v okolí kurzoru myši. Na základě této úvahy byl navržen nový způsob pohybu kamery. Kamera bude vždy zabírat herní postavu, nicméně ta nebude pouze ve středu zorného pole. Lokace kamery bude vypočítána na základě pozice herní postavy a pozice kurzoru v okně aplikace. Zjednodušeně lze říci, že kamera se bude pohybovat za kurzorem, ale maximálně tak, aby herní postava zůstala vždy v záběru kamery. Pozice kamery bude ve středu mezi kurzorem myši a herní postavou. Pro lepší představu ji znázorňuje obrázek 6.4 při různých umístěných kurzoru.



Obrázek 6.4: Znázornění pozice kamery vůči herní postavě a kurzoru

Zároveň bude možné pomocí klávesové zkratky pohyb kamery přepnout do volného módu, ve kterém se kamera naprosto odloučí od herní postavy



Obrázek 6.5: Příklad zón zorného pole při přepnutí pohybu kamery do volného módu

a bude se pohybovat čistě za kurzorem. Tento mód slouží k rozhlédnutí po herní mapě tak, aby se hráč mohl podívat i na místa, od kterých je jeho herní postava daleko. Pro tento účel je zorné pole rozděleno do tří zón z pohledu pozice kurzoru (viz obrázek 6.5). Zóna nejblíže středu je neutrální zóna. Pokud se kurzor nachází v této zóně, kamera se nepohybuje. V prostřední zóně (zóna pohybu) se kamera pohybuje směrem ke kurzoru, přičemž rychlost pohybu je lineárně závislá na vzdálenosti od neutrální zóny (čím dále, tím rychleji). Pokud je kurzor umístěn v poslední zóně (zóna maximálního pohybu), kamera se opět pohybuje směrem ke kurzoru, akorát rychlost pohybu je maximálně možná, specifikovaná konstantou.

Pro zajištění veškeré výše zmíněné funkcionality byla vytvořena nová komponenta *CameraArmComponent*, která je prostředníkem mezi kamerou a herní postavou. Herní postava má na sobě připojenou instanci *CameraArmComponent* komponenty a ta na sobě má připevněnou komponentu kamery. Uvnitř první verze komponenty *CameraArmComponent* byla implementována logika pro negování rotace herní postavy za účelem zabránění rotace kamery a také logika pro oba způsoby pohybu kamery včetně definice parametrů, pomocí kterých je možné ovládat rychlost pohybu kamery nebo velikost zmíněných zón. Bohužel nebylo možné plynule míchat jednotlivé módy pohybu a přecházet tak mezi nimi. Logika volného pohybu byla z komponenty vyjmuta a přesunuta do samostatné třídy *FreeCamera*, která je podtřídou třídy *Actor*. Díky této změně je možné využít odladěnou zabudovanou funkcionalitu přechodu mezi různými kamerami a přechod je tak plynulý bez citelného třesení.

6.5 Problém s replikací instancí třídy *Actor*

Při vývoji různých částí hry autor práce narazil na komplikace, které byly způsobené replikací instancí třídy *Actor*. Každá replikovaná instance třídy *Actor* musí být vytvořena na serveru a následně je po síti odeslána všem klientům. Konkrétně je odeslán požadavek, aby si klienti vytvořili novou instanci dané třídy a přiřadili jí specifikované *NetworkId* a inicializační parametry. Na základě *NetworkId* se klientské verze instance párují s verzí serverovou. Díky tomuto chování mohou při špatném zacházení s objekty vzniknout někdy až fatální chyby, které vedou k pádům programu. Nejčastější chybou je snaha použít instanci objektu, která ještě nebyla vytvořena.

Části hry, při jejichž vývoji se autor práce s těmito problémy potýkal, byly zejména herní módy a výběr avatarů. Poté, co se hráči připojí na server, musí server pro každého z nich vytvořit instanci třídy *PlayerState* a odeslat všem informace o všech replikovaných instancích třídy *Actor*, mezi které se řadí mimo jiné instance třídy *PlayerState* a *GameState*. Těchto instancí může být několik. Informace jsou tedy typicky rozděleny do několika zpráv, u kterých nelze předem určit pořadí ani dobu, za jakou budou doručeny. Velmi často se tedy stává, že klient si na základě přijaté zprávy od serveru vytvoří kopii serverové instance, jejíž inicializační logika využívá ukazatele na instance jiných objektů, které ale ještě nebyly vytvořeny, protože pokyny pro jejich vytvoření byly poslány až v pozdějších zprávách.

Jako příklad lze uvést třídu *HUD* sloužící k zobrazení uživatelského rozhraní. Ve výběru avatarů jsou součástí uživatelského rozhraní hráčské kartičky, které obsahují mimo jiné jména hráčů. Klient přijme instrukci, na jejímž základě si instanci třídy *HUD* vytvoří. V rámci inicializace a zobrazení hráčských kartiček je potřeba mít seznam všech instancí třídy *PlayerState* a získat z nich jméno hráče. Bohužel v době inicializace instance třídy *HUD* nemusí být instance třídy *PlayerState* ještě dostupné nebo mohou být dostupné jenom některé, protože instrukce k jejich vytvoření dorazí až později.

Řešením tohoto obecného problému je hojné využívání událostí a delegátů, které jsou vyvolány až po vytvoření nebo změně specifických instancí. Konkrétně pro řešení uvedeného příkladu je možné vytvořit událost *OnAllPlayerStatesInitialized*, která bude zavolána až v momentě, kdy budou existovat všechny instance třídy *PlayerState*. Instance třídy *HUD* se na tuto událost napojí a inicializaci uživatelského rozhraní provede až na jejím základě. Ovšem je důležité si dát pozor na to, že samotná událost může nastat ještě dříve, než se na ní všichni zainteresované objekty připojí. Instance třídy *HUD* může být klidně vytvořena až poté, co budou vytvořeny všechny instance třídy *PlayerState* a událost tak nastane dříve. Je tedy potřeba zajistit, že při napojení na již proběhlou událost dá sama událost připojovanému objektu vědět, že již proběhla. Případně si to musí připojovaný objekt ověřit sám.

6.6 Schopnosti, výpočet poškození a míření

Jak již bylo několikrát zmíněno, schopnosti jsou postaveny v systému *Gameplay Ability System*. Tento systém nebylo potřeba žádným způsobem rozšiřovat nebo modifikovat, je velmi robustní a pokrývá veškeré potřeby hry Inpemo. Finální schopnosti byly skriptovány v systému *Blueprints*.

Schopnosti mnohdy vytvářejí nové objekty, které následně putují herním světem. Jednou takovou schopností je například schopnost „Fireball“. Po použití schopnosti je vytvořena ohnivá koule letící vybraným směrem. Ovšem povrch herní mapy není zcela rovný. Pokud je ohnivá koule vytvořena v nejnižším místě mapy a letí směrem k vyšším místům, může částečně nebo zcela úplně zaletět pod povrch. K vyřešení tohoto problému musela být vytvořena nová komponenta *MissileMovementComponent*. Ta pomocí procesu *raycasting*[13] zajistí, že létající objekty s touto komponentou kopírují povrch. Je třeba pouze nastavit komponentě požadovanou výšku, kterou má objekt od povrchu udržovat.

Dále byly všem avatarům přidány dva speciální atributy *Damage* (poškození) a *Healing* (léčení). Ty slouží pouze jako schránky pro dočasné hodnoty, které mají měnit jiné atributy. Může se totiž stát, že avatar v rámci jednoho snímku obdrží více různého poškození a léčení zároveň, přičemž je na místě, aby v takovém případě nemělo pořadí efektů vliv na celkový výsledek. Problém je více vysvětlen na následujícím příkladě. Jednotlivé body jdou chronologicky za sebou.

1. Avatar má 200 bodů zdraví (*Health*) a začíná snímek.
2. Avatar obdrží poškození za 250 bodů zdraví.
3. Avatar obdrží léčení za 125 bodů zdraví.
4. Avatar obdrží poškození za 25 bodů zdraví.
5. Avatar Snímek končí a začíná nový.

Pokud by bylo ponecháno výchozí počítání atributů, avatar by po prvním obdrženém poškození zemřel. Je třeba vzít v potaz fakt, že snímek na průměrném stroji bude trvat přibližně 16,66 milisekundy. Z hlediska vnímání hráčů se tato doba dá považovat za jeden okamžik. Z navrženého rozhodnutí není žádoucí, aby hráč, který obdržel ve stejný okamžik poškození a k tomu adekvátní léčení, zemřel. Proto případné poškození a léčení nemodifikuje přímo atribut *Health* (body zdraví), nýbrž atributy *Damage* a *Healing*. Na konci snímku jsou tyto atributy sloučeny do jedné hodnoty, aplikovány na atribut *Health* a vynulovány (viz výpis kódu 6.3).

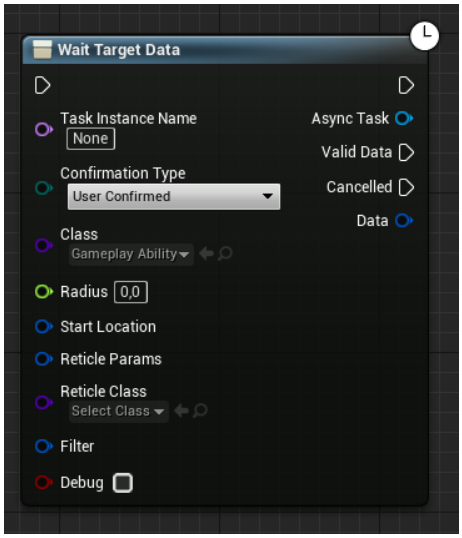
[13] Proces provádějící kolizní trasování podle specifikované trajektorie, který vrátí první zasažený objekt.

```
void PostGameplayEffectExecute(...) {
    ...
    const float NewHealth = Health + Healing - Damage;
    SetHealing(0.0f);
    SetDamage(0.0f);
    SetHealth(FMath::Clamp(NewHealth, 0.0f, MaxHealth));
    if(Health < 0.1f) {
        //avatar dies
    }
    //end of frame
}
```

Výpis kódu 6.3: Výpočet atributu *Health*

Použití tohoto postupu na uvedeném příkladě má za výsledek přežití avatara s 50 body zdraví, což je požadované chování. Pro tento konkrétní případ by šel použít pouze jeden dočasný atribut. Nicméně hra Inpemo může do budoucna obsahovat například i efekt snížení léčení nebo poškození po určitou dobu. Z tohoto důvodu jsou atributy *Damage* a *Healing* rozděleny na dva, aby případné vytváření podobných efektů bylo snadnější.

Zaměřovací logika využívá již hotový uzel *WaitTargetData*. Ten má několik vstupních parametrů. Nejdůležitějším z nich je parametr *Class*, pomocí kterého lze specifikovat zaměřovací typ (kruh okolo hráče, linka, kruhová oblast apod.). Dle hodnoty parametru *class* uzel nabídne parametry pro vybraný zaměřovací typ.



Obrázek 6.6: Uzel *WaitTargetData* v systému *Blueprints*

Na obrázku 6.6 je vybrán zaměřovací typ *TargetActor_Radius* (kruh okolo hráče), který má parametr *radius* (poloměr) a *start location* (střed). V případě dokončení míření uzel vrací data, která obsahují informace o vybrané oblasti a seznam zasažených herních postav.

6.7 Pohyb avatarů

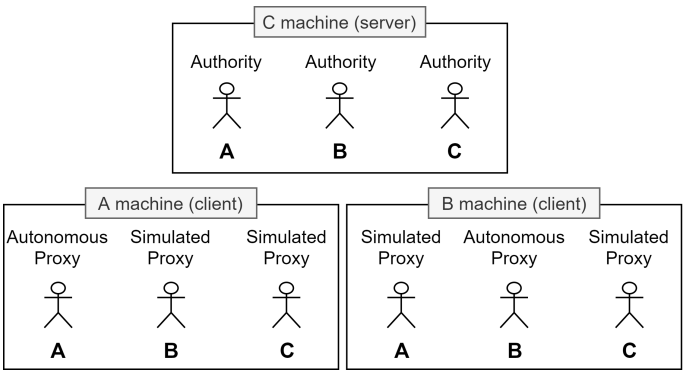
Pohyb obstarává komponenta *InpCharacterMovementComponent*, která je potomkem vestavěné komponenty *CharacterMovementComponent*. Vestavěná komponenta poskytuje pohybový systém s běžnými režimy pohybu pro humanoidní postavy (chůze, pád, plavání a létaní), které jsou všechny navrženy tak, aby plně podporovaly síťové hry. Komponenta také obsahuje podporu pro přidání vlastních režimů pohybu.

Během každého snímku komponenta *CharacterMovementComponent* volá funkci *PerformMovement*. Ta je zodpovědná za fyzické přemístění herní postavy v úrovni a je volána jinými funkcemi, které jsou různé pro server a klienty. *PerformMovement* volá různé *Phys* funkce dle aktuálního pohybového režimu, například funkci *PhysWalking* obsahující logiku chůze nebo *PhysSwimming* definující logiku plavání.

Komponenta *CharacterMovementComponent* používá síťovou roli své herní postavy k vybrání správné logiky replikace. Herní postava může nabývat jedné ze tří rolí.

- **Authority** Jedná se o instanci herní postavy, která je na serveru.
- **Autonomous Proxy** Instance herní postavy se nachází na klientovi, který jí lokálně ovládá.
- **Simulated Proxy** V tomto případě je instance herní postavy na některém z klientů, který ji lokálně neovládá. Může se jednat o postavu jiného než lokálního hráče nebo postavu, která je ovládána umělou inteligencí.

Obrázek 6.7 vyobrazuje síťové role herních postav v jednotlivých herních instancích, přičemž je zde brán případ listen serveru a dvou klientů.



Obrázek 6.7: Síťové role herních postav ve hře o třech hráčích (listen server)

Pokud hráč *A* ve své herní instanci zavolá funkci *GetNetworkRole* na své postavě, výsledkem bude role *Autonomous Proxy*, jelikož danou postavu lokálně ovládá. Pokud to stejné udělá hráč *B*, tedy zavolá ve své herní instanci

funkci *GetNetworkRole* na postavě hráče *A*, výsledkem bude role *SimulatedProxy*, protože postavu neovládá a z hlediska jeho herní instance se jedná pouze o postavu, která je simulována na základě informací ze serveru. Hráč *C* je server, z toho důvodu mají všechny jeho instance roli *Authority*.

Logika vyhodnocování a vyhlazování pohybu se liší podle síťové role. *Autonomous Proxy* používá strategii tahů (popsána v sekci 1.2.5.3), kdežto *Simulated Proxy* používá interpolaci. Níže jsou chronologicky popsány všechny zásadní kroky, které *CharacterMovementComponent* dělá (podle síťové role).

- **Autonomous Proxy (lokální hráč)**
  1. Lokální klient ovládá svou postavu. Je zavolána funkce *PerformMovement*.
  2. Výsledek funkce *PerformMovement* je uložen do struktury *FSavedMove* společně se všemi daty, která byla potřeba pro vyhodnocení pohybu. Tato struktura je následně zařazena do fronty *SavedMoves*.
  3. Podobné struktury *FSavedMove* ve frontě jsou sloučeny za účelem zmenšení velikosti dat a jsou poslány na server.
- **Authority (server)**
  4. Server obdrží frontu *FSavedMove* struktur a zreprodukuje pohyb klienta pomocí funkce *PerformMovement*.
  5. Proběhne kontrola, zda se výsledná pozice (výsledek funkce *PerformMovement*) neliší na klientovi a serveru.
  6. Pokud se neliší, je klientovi (*Autonomous Proxy*) odesláno potvrzení o validitě pohybů. V opačném případě je odeslána oprava pomocí vzdáleného volání funkce *ClientAdjustPosition*.
  7. Server odešle všem ostatním klientům (všem *Simulated Proxies*) pozici, rotaci a stav pohybu pomocí replikované struktury *ReplicatedMovement*.
- **Autonomous Proxy (lokální hráč)**
  8. Pokud klient obdrží volání funkce *ClientAdjustPosition*, zreprodukuje pohyb serveru a použije frontu *SavedMoves* k přepočítání pohybů a dosazení nového korektního stavu.
  9. Až po poslední potvrzený/opravený pohyb (včetně) jsou všechny pohyby z fronty *SavedMoves* odstraněny.
- **Simulated Proxy (ostatní klienti)**
  10. Zreplikuje pohyb postavy na základě doručených informací z *ReplicatedMovement* struktury, přičemž pro vyhlazení pohybu použije interpolaci.

CharacterMovementComponent ve výchozím stavu nabízí lineární a exponenciální interpolaci pro vyhlazení pohybu u *Simulated Proxies*. Typ interpolace je specifikován parametrem *NetworkSmoothingMode*. Také je možné komponentě dodat zcela vlastní interpolační logiku. Hra Inpemo využívá interpolaci lineární.

Komponenta *InpCharacterMovementComponent* rozšiřuje základní pohybové režimy o režim *Pushbacking*. V tomto pohybovém módu se bude hráč nacházet v době letu při odstrčení. Výpočet rychlosti postavy je možné vidět ve výpisu kódu 6.4. Rychlost postavy se snižuje na základě tření specifikovaného parametrem *Friction*. Pokud rychlost postavy při odstrčení dosáhne určité hranice, postava začne zpomalovat konstantně. Poté, co se postava zcela zastaví, je nastavena chůze jako nový režim pohybu.

```
1  void PhysPushbacking(float DeltaTime, int32 Iterations)
2  {
3    ....
4    FVector OldVelocity = Velocity;
5    Velocity = Velocity * (1.f-FMath::Min(Friction * DeltaTime, 1.f));
6    if (Velocity.Size() < DECELERATION_THRESHOLD)
7    {
8      float Deceleration = -1 * DECELERATION_COEF * DeltaTime;
9      Velocity = Velocity + Velocity.GetSafeNormal2D() * Deceleration;
10   }
11   //vector operator | does dot product
12   if (Velocity.IsNearlyZero(50) || (Velocity | OldVelocity) <= 0.f)
13   {
14     SetMovementMode(MOVE_Walking);
15     Velocity = FVector::ZeroVector;
16   }
17   ....
18 }
```

Výpis kódu 6.4: Výpočet rychlosti herní postavy při odstrčení

6.8 Grafické a zvukové assety

Assety použité v prototypu pocházejí z různých zdrojů. Grafické assety uživatelského rozhraní byly zakoupeny v obchodě *GameDev Market* za 7,26 USD [54]. Assety prostředí pro tvorbu úrovní zdarma vytvořil a dodal Marek Baláž. Postavu včetně některých animací zdarma vytvořila Dominika Draesslerová. Zvuky pochází z volně dostupných databank. Vizuální efekty, materiály (např. efekty schopností, materiál vody) a zbylé potřebné animace vytvořil autor práce sám.

Kapitola 7

Testování

7.1 Testovací a vývojové zařízení

Pro vývoj a automatické testování byl použit stolní počítač a notebook. Konfigurace obou zařízení je popsána v tabulce 7.1. Obě zařízení mají nainstalovaný operační systém Microsoft Windows 10 64bit.

Zařízení	**CPU**	**GPU**	**RAM**
Stolní počítač	AMD Ryzen 9 3950x	Nvidia GeForce RTX 2070 Super	64 GB DDR4
Notebook	Intel Core i7-8750H	Nvidia GeForce GTX 1050 Ti	16 GB DDR4

Tabulka 7.1: Konfigurace vývojových zařízení

7.2 Testování programátorem

Testování her pro více hráčů během vývoje je výrazně obtížnější než testování her pro jednoho hráče. Naštěstí UE4 se snaží vývojářům poskytnout velké množství nástrojů k jeho zlehčení. Velmi důležitou funkcionalitou je možnost spuštění více různých instancí hry, přičemž se engine sám postará o jejich propojení, čímž simuluje síťové spojení. V tomto režimu testování je možné pomocí příkazové konzole simulovat různé parametry, jako je například latence nebo výpadky zpráv. Engine také poskytuje nástroje pro sledování vytíženosti CPU a GPU nebo monitorování síťové komunikace. Pomocí těchto nástrojů je možné identifikovat problémové části kódu.

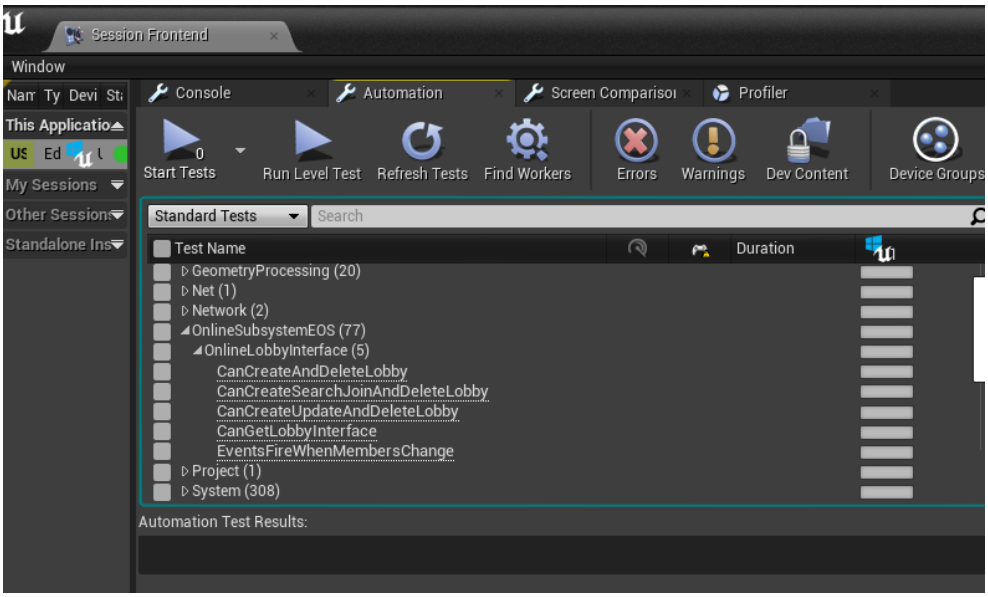
Všechny zmíněné nástroje byly při testování během vývoje použity. Důležité funkcionality a ty, které nešly zmíněnými nástroji testovat (například integrace *Steamworks*), byly otestovány autorem práce pomocí zařízení uvedených v tabulce 7.1.

7.3 Automatizované testování

UE4 nabízí dva různé systémy pro automatizované testování. *Automation System* slouží pro nízkoúrovňové testy, testy editoru a testy podpůrných systémů, které nesouvisí přímo s logikou a pravidly hry. Tyto testy se píší pomocí C++ a dělí se do následujících kategorií.

- **Unit** Testy na úrovni API jednotlivých systémů.
- **Smoke** Jedná se o rychlé nekomplexní testy. Každý takový test by měl být opravdu rychlý, aby bylo možné tyto testy spouštět před každým spuštěním editoru nebo hry.
- **Screenshot Comparison** Test, který umožňuje vytváření snímků a jejich následné porovnávání s nastavitelným stupněm volnosti.

Druhý systém *Functional Testing Framework* je navržen pro komplexnější testování herních úrovní, herních pravidel a celkově celé hratelnosti. Takové testy se dají psát v C++, nicméně více preferovanou variantou je tvoření testů pomocí systému *Blueprints*. Každý test vytvořený v tomto systému musí mít vlastní úroveň, jejíž název začíná prefixem „FTEST“, jinak ho engine nerozpozná.

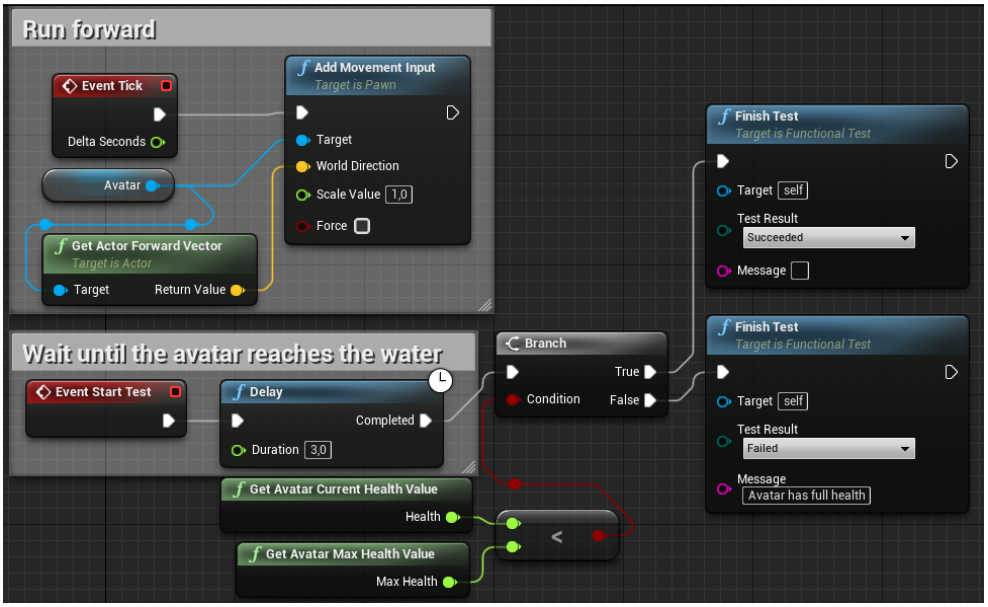


Obrázek 7.1: Prostředí pro spouštění testů v UE4

Pomocí prvního zmíněného systému bylo vytvořeno několik unit testů, které se soustředí na implementaci *IOnlineLobby* rozhraní. Například test

„CanCreateAndDeleteLobby“ zkusí vytvořit a následně smazat místnost. Názvy zbylých testů je možné vidět na obrázku 7.1, který zároveň zobrazuje i prostředí, ze kterého jsou testy v UE4 spouštěny.

Ve druhém systému byly vytvořeny testy testující herní logiku a pravidla. Jedním takovým testem je test „ZoneDamage“, jehož implementace je vidět na obrázku 7.2. Úroveň testu obsahuje pouze vodu (ta reprezentuje smrtelnou zónu), ostrov a jednoho avatara, který je umístěn blízko břehu ostrova (na pevnině) a je otočen čelem k vodě. Při průběhu testu je simulován uživatelský vstup tak, aby avatar běžel směrem do vody. Po určitém časovém úseku je ověřeno, zda avatar obdržel od vody poškození. Pokud ano, test je vyhodnocen jako úspěšný.



Obrázek 7.2: Implementace testu „ZoneDamage“

7.4 Testování s hráči

Testování se zúčastnilo šest hráčů, přičemž jedním z nich byl autor práce. *Epic Online Services* služby bohužel ještě neumožňují ověření organizace. To má za následek to, že se mohou přihlásit pouze ti hráči, kteří jsou členy organizace vlastnící hru v portálu *EOS Developer Portal*. Tedy každý hráč, který chce hru hrát, musí být v momentální situaci vlastníkem produktu manuálně registrován a přidán, čímž je z hlediska portálu považován za člena vývojového týmu. Z toho důvodu bylo zvoleno testování v menším počtu hráčů, kteří jsou zároveň blízcí známí autora práce.

Všichni zúčastnění hráči jsou věku 23-26 let, hrají z hlediska herního výkonu na průměrných až nadprůměrných zařízeních, mají pokročilejší zkušenost s *MOBA* hrami a jsou seznámeni s problémy online multiplayer her. Před začátkem testování byli hráči požádáni, aby si změřili rychlost připojení. Nejpomalejší naměřená rychlost napříč všemi byla 13,1 Mb za sekundu pro stahování a 2,4 Mb za sekundu pro nahrávání. Testování se primárně soustředilo na síťovou komunikaci, stabilitu serveru a případné chyby, které díky reálné síťové komunikaci mohly vznikat. Všichni hráči byli seznámeni s herními mechanikami a byl jim puštěn krátký video-záznam z testování autorem práce tak, aby hru lépe pochopili. Také byli instruováni, aby se soustředili především na špatnou odezvu herní postavy vzhledem k uživatelskému vstupu, falešně trefené schopnosti, vysokou latenci a jakékoliv jiné projevy, které mohou být spojeny se síťovou komunikací. S těmito poznatky hráči odehráli šest zápasů, přičemž se všichni prostřídali v roli serveru.

Po odehrání prvních šesti zápasů byly odehrány ještě další dva s tím, že host byl hráč s nejrychlejším připojením a všichni byly instruováni, aby se soustředili na celkový herní zážitek a hru jako celek.

7.4.1 Výsledky

Testování obecně dopadlo úspěšně. Nevyskytly se žádné zásadní problémy a hráči hru po grafické i herní stránce chválili. Celkově hráči nahlásili šest problémů, přičemž první dva souvisí se síťovou komunikací.

1. **Výpadek serveru** Při čtvrtém zápase měl host krátký výpadek připojení, ze kterého se hra nedokázala vzpamatovat. Zápas tedy skončil síťovou chybou a všichni hráči byli přesunuti do hlavního menu.

2. **Odezva při použití schopnosti** V zápase, kde serverem byl hráč s nejslabším připojením, se dvěma hráčům stalo, že při použití schopnosti pocítili výrazné zpozdění.

3. **Problém při založení nové místnosti** Po výpadku serveru ve čtvrtém zápase nemohl další hráč v pořadí založit novou místnost. Bylo potřeba restartovat hru.

4. **Avatar může vyběhnout mimo herní mapu** Během páté hry se jeden hráč rozhodl běžet co nejdále ve vodě a vyběhl z herní mapy.

5. **Problém se schopností teleportace** Při použití schopnosti teleportace se hráč zasekl uvnitř palmy.

Nahlášené chyby s označením tři, čtyři a pět byly následně opraveny. Zbylé problémy není možné aktuálně vyřešit, protože vyžadují komplexnější řešení. Návrhy pro jejich řešení jsou nastíněny v kapitole 8.

Kapitola 8

Náměty k budoucímu rozšíření

8.1 Host migration

Jedná se o techniku, pomocí které lze přenést funkci serveru na jednoho z klientů. Díky použití této techniky lze při výpadku serveru zachránit herní relaci a hráči tak mohou pokračovat v zápase. UE4 svojí architekturou bohužel nepodporuje použití této techniky, jelikož při prvním připojení na server je vždy nutné použít *non-seamless* načítání, které neumožňuje zachovat aktuální stav úrovně. Jediným řešením je vytvoření komplexního systému pro uložení a načtení stavu zápasu. Nový potenciální server poté před opuštěním úrovně uloží stav zápasu, založí server a načte stav zápasu z uložených dat.

8.2 Prediktivní chování schopností

V jistých situacích by bylo dobré umožnit klientům použití schopnosti ještě před potvrzením od serveru. Tím by se zlepšila odezva hry na uživatelské vstupy a případná vyšší latence by nebyla pro hráče tolik citelná. Zejména by pomohlo prediktivní vytváření instancí třídy *Actor* s tím, že by se následně instance spárovaly se svými serverovými protějšky.

8.3 Tutoriál

Ve hře se nenachází žádná úvodní úroveň, která by hráče provedla hrou a představila mu základní herní mechaniky. Z hlediska prototypu to autor práce nevnímá jako problém, ale pokud by hra měla vyjít v plné verzi, zcela jistě by něco takového musela obsahovat.

Závěr

Hlavním cílem této práce bylo vytvořit funkční prototyp hry Inpemo pro operační systém Microsoft Windows 10, který vychází z již existujícího herního návrhového dokumentu. V rámci toho měly být prozkoumány vhodné technologie, síťové topologie a principy používané při vývoji online her pro více hráčů. Výsledný prototyp měl být otestován a používat podpůrné služby *BVision*, jejichž autor je Ing. Petr Nohejl.

Začátek práce se věnuje typům her pro více hráčů a jejich problémům z hlediska síťové komunikace. Poté jsou představeny hlavní protokoly a síťové topologie včetně rozebrání jejich použitelnosti pro online hry. Také jsou zde vysvětleny důležité principy a techniky používané pro řešení a zamaskování problémů spojených se síťovou komunikací. Byly prozkoumány dva v současnosti nejpoužívanější herní enginy, přičemž pro vývoj prototypu byl použit Unreal Engine 4, který byl vybrán zejména pro své robustní vestavěné nástroje sloužící právě k vývoji online her pro více hráčů. Díky této volbě a finančním možnostem byla vybrána síťová topologie listen server. Na základě návrhového dokumentu byly specifikovány funkční a nefunkční požadavky a poté navrženy důležité části hry. Následně jsou pak popsány zajímavé části implementace a problémy, které při ní vznikly. Poslední část práce se zabývá testováním vytvořeného prototypu a návrhům pro budoucí rozšíření.

Všechny požadavky a cíle práce byly splněny. Výsledkem je funkční prototyp, který je dle úsudku autora spíše stabilním základem, který je možné snadno rozšířit a udělat z něj tak plnou verzi hry. Podpůrné služby od Ing. Petra Nohejla sice nejsou ve výsledném prototypu z mnoha důvodů použity, ale kvůli splnění zadání byla přesto vytvořena jejich integrace.

Literatura

[1] Epic Games: Gameplay Framework Quick Reference. [online], [cit. 2021-03-20]. Dostupné z: https://docs.unrealengine.com/en-US/InteractiveExperiences/Framework/QuickReference/index.html

[2] Barton, M.; Loguidice, B.: The History Of Pong: Avoid Missing Game to Start Industry. *Gamasutra - The Art & Business of Making Games*, 2009, [cit. 2021-03-08]. Dostupné z: https://www.gamasutra.com/view/feature/132293/the_history_of_pong_avoid_missing_.php

[3] Riot Games: Legue of Legends. [online], [cit. 2021-03-05]. Dostupné z: https://euw.leagueoflegends.com/en-gb/

[4] CD Project RED: Cyberpunk 2077. [online], [cit. 2021-03-05]. Dostupné z: https://www.cyberpunk.net/cz/en/

[5] Gulati, S.: 12 Most Popular Video Games In 2020 You Can Play Right Away. [online], Prosinec 2020, [cit. 2021-03-03]. Dostupné z: https://fossbytes.com/most-popular-video-games/

[6] IBM: *TCP/IP protocols [online]*. [cit. 2021-03-08]. Dostupné z: https://www.ibm.com/support/knowledgecenter/SSGMCP_5.3.0/com.ibm.cics.ts.internet.doc/topics/dfhtl23.html

[7] Information Sciences Institute University of Southern California: RFC 793 - Transmission Control Protocol. [online], Září 1981, [cit. 2021-03-05]. Dostupné z: https://tools.ietf.org/html/rfc793

[8] Postel, J.: RFC 768 - User Datagram Protocol. [online], Srpen 1980, [cit. 2021-03-05]. Dostupné z: https://tools.ietf.org/html/rfc768

[9] Information Sciences Institute University of Southern California: RFC 791 - Internet Protocol. [online], Září 1981, [cit. 2021-03-05]. Dostupné z: https://tools.ietf.org/html/rfc791

[10] Nagle, J.: Congestion Control in IP/TCP Internetworks. [online], Leden 1984, [cit. 2021-03-05]. Dostupné z: https://tools.ietf.org/html/rfc896

[11] Eyeball Networks Inc: NAT & Firewall Traversal Technologies for VoIP & Video Telephony using STUN, TURN, & ICE. [online], 2013, [cit. 2021-03-05]. Dostupné z: https://web.archive.org/web/20131019201448/http://www.nattraversal.com/

[12] Huynh, C. P.; Hunt, R.; McKenzie, A.: NAT Traversal Techniques in Peer-to-Peer Networks. Leden 2008.

[13] interpolation: Interpolation – Wikipedia, The Free Encyclopedia. [cit. 2021-03-15]. Dostupné z: https://en.wikipedia.org/wiki/Interpolation

[14] Linear interpolation: Linear interpolation – Wikipedia, The Free Encyclopedia. [cit. 2021-03-15]. Dostupné z: https://en.wikipedia.org/wiki/Linear_interpolation

[15] Extrapolation: Extrapolation – Wikipedia, The Free Encyclopedia. [cit. 2021-03-15]. Dostupné z: https://en.wikipedia.org/wiki/Extrapolation

[16] Aronson, J.: Dead Reckoning: Latency Hiding for Networked Games. *Gamasutra - The Art & Business of Making Games*, 1997, [cit. 2021-03-12]. Dostupné z: https://www.gamasutra.com/view/feature/131638/dead_reckoning_latency_hiding_for_.php

[17] Christopoulou, E.; Xinogalos, S.: Overview and Comparative Analysis of Game Engines for Desktop and Mobile Devices. *International Journal of Serious Games*, ročník 4, 12 2017, doi:10.17083/ijsg.v4i4.194.

[18] Thomsen, M.: History of the Unreal Engine. *Gamasutra - The Art & Business of Making Games*, 2012, [cit. 2021-03-12]. Dostupné z: https://www.ign.com/articles/2010/02/23/history-of-the-unreal-engine

[19] Epic Games: FREQUENTLY ASKED QUESTIONS (FAQ). [online], [cit. 2021-03-17]. Dostupné z: https://www.unrealengine.com/en-US/faq

[20] Epic Games: What is GitHub? [online], [cit. 2021-03-17]. Dostupné z: https://www.unrealengine.com/en-US/ue4-on-github

[21] Games, E.: Unreal Engine End User License Agreement For Publishing. [online], [cit. 2021-03-16]. Dostupné z: https://www.unrealengine.com/en-US/eula/publishing

[22] Unity (game engine): Unity (game engine) – Wikipedia, The Free Encyclopedia. [cit. 2021-03-16]. Dostupné z: https://en.wikipedia.org/wiki/Unity_(game_engine)

[23] Technologies, U.: Choose the plan that is right for you. [online], [cit. 2021-03-16]. Dostupné z: https://store.unity.com/compare-plans

[24] Technologies, U.: Getting Started with MLAPI. [online], [cit. 2021-03-16]. Dostupné z: https://docs-multiplayer.unity3d.com/docs/getting-started/about-mlapi

[25] Technologies, U.: UNet Deprecation FAQ. [online], [cit. 2021-03-16]. Dostupné z: https://support.unity.com/hc/en-us/articles/360001252086-UNet-Deprecation-FAQ

[26] vis2k: Mirror - Unity Asset Store. [cit. 2021-03-17]. Dostupné z: https://assetstore.unity.com/packages/tools/network/mirror-129321?_ga=2.188590957.458639248.1616622749-655583969.1615994301

[27] Corporation, V.: Welcome to Steam. [online], [cit. 2021-03-18]. Dostupné z: https://store.steampowered.com/

[28] Corporation, V.: Steamworks. [online], [cit. 2021-03-18]. Dostupné z: https://partner.steamgames.com/

[29] Games, E.: Epic Games Store. [online], [cit. 2021-03-18]. Dostupné z: https://www.epicgames.com/store/en-US/

[30] Games, E.: Epic Online Services. [online], [cit. 2021-03-18]. Dostupné z: https://dev.epicgames.com/en-US/services

[31] Nohejl, P.: Podpůrné služby pro online hru Inpemo. Prosinec 2021, [cit. 2021-03-18]. Dostupné z: https://dspace.cvut.cz/handle/10467/92940

[32] Stunlock Studios: Battlerite on Steam. [online], [cit. 2021-03-17]. Dostupné z: https://store.steampowered.com/app/504370/Battlerite/

[33] PlayTracker: Battlerite stats by Playtracker Insight. [online], [cit. 2021-03-17]. Dostupné z: https://playtracker.net/insight/game/1068

[34] Frogsong Studios: Spellsworn on Steam. [online], [cit. 2021-03-17]. Dostupné z: https://store.steampowered.com/app/360620/Spellsworn/

[35] PlayTracker: Spellsworn stats by Playtracker Insight. [online], [cit. 2021-03-17]. Dostupné z: https://playtracker.net/insight/game/4861

[36] Epic Games: Unreal Engine 4 Documentation. [online], [cit. 2021-03-17]. Dostupné z: https://docs.unrealengine.com/en-US/index.html

[37] Games, E.: UMG UI Designer. [online], [cit. 2021-03-18]. Dostupné z: https://docs.unrealengine.com/en-US/InteractiveExperiences/UMG/index.html

[38] Epic Games: Netdriver.h. [online], [cit. 2021-03-20]. Dostupné z: https://github.com/EpicGames/UnrealEngine/blob/release/Engine/Source/Runtime/Engine/Classes/Engine/NetDriver.h

[39] Epic Games: Property Replication. [online], [cit. 2021-03-20]. Dostupné z: https://docs.unrealengine.com/en-US/InteractiveExperiences/Networking/Actors/Properties/index.html

[40] Epic Games: RPCs. [online], [cit. 2021-03-20]. Dostupné z: https://docs.unrealengine.com/en-US/InteractiveExperiences/Networking/Actors/RPCs/index.html

[41] Epic Games: Online Subsystem. [online], [cit. 2021-03-21]. Dostupné z: https://docs.unrealengine.com/en-US/ProgrammingAndScripting/Online/index.html

[42] Amazon Web Services: Amazon GameLift. [online], [cit. 2021-03-22]. Dostupné z: https://aws.amazon.com/gamelift/

[43] Amazon Web Services: Amazon GameLift Pricing. [online], [cit. 2021-03-22]. Dostupné z: https://aws.amazon.com/gamelift/pricing/

[44] Epic Games: Connect Interface. [online], [cit. 2021-03-22]. Dostupné z: https://dev.epicgames.com/docs/services/en-US/Interfaces/Connect/index.html

[45] Epic Games: OpenID Configuration. [online], [cit. 2021-03-22]. Dostupné z: https://api.epicgames.dev/epic/oauth/v1/.well-known/openid-configuration

[46] Epic Games: Lobby Interface. [online], [cit. 2021-03-22]. Dostupné z: https://dev.epicgames.com/docs/services/en-US/Interfaces/Lobby/index.html

[47] Epic Games: EOS Public Roadmap. [online], [cit. 2021-03-22]. Dostupné z: https://trello.com/b/rLvzFJfE/eos-public-roadmap

[48] Heroic Labs: Nakama: Realtime, social competitive game server. [online], [cit. 2021-03-24]. Dostupné z: https://heroiclabs.com/nakama-opensource/

[49] Redpoint Games: EOS Online Subsystem. [online], [cit. 2021-03-24]. Dostupné z: https://www.unrealengine.com/marketplace/en-US/product/eos-online-subsystem

[50] Redpoint Games: Configuring authentication. [online], [cit. 2021-03-24]. Dostupné z: https://redpointgames.gitlab.io/eos-online-subsystem/docs/identity_configuration

[51] Epic Games: Web Browser. [online], [cit. 2021-03-25]. Dostupné z: https://docs.unrealengine.com/en-US/InteractiveExperiences/UMG/UserGuide/WidgetTypeReference/WebBrowser/index.html

[52] Epic Games: Programming Subsystems. [online], [cit. 2021-03-25]. Dostupné z: https://docs.unrealengine.com/en-US/ProgrammingAndScripting/Subsystems/index.html

[53] Epic Games: Travelling in Multiplayer. [online], [cit. 2021-03-26]. Dostupné z: https://docs.unrealengine.com/en-US/InteractiveExperiences/Networking/Travelling/index.html

[54] GameDev Market: Wooden UI. [online], [cit. 2021-03-28]. Dostupné z: https://www.gamedevmarket.net/asset/wooden-ui-9146/

Příloha A

Seznam použitých zkratek

2D Two-dimensional

3D Three-dimensional

API Application programming interface

EOS Epic Online Services

FPS First person shooter

HUD Head-up display

ICE Interactive Connectivity Establishment

IP Internet Protocol

LAN Local area network

MLAPI Mid level API

MMORPG Massively multiplayer online role-playing game

MOBA Multiplayer online battle arena

NAT Network address translation

RFC Request For Comments

RPCs Remote procedure calls

RTS Real-time strategy

RTT Round trip time

STUN Session Traversal Utilities for NAT

SVN Apache Subversion

TCP Transmission Control Protocol

TURN Traversal Using Relays around NAT

UE4 Unreal Engine 4

UDP User Datagram Protocol

vCPU virtual central processing unit

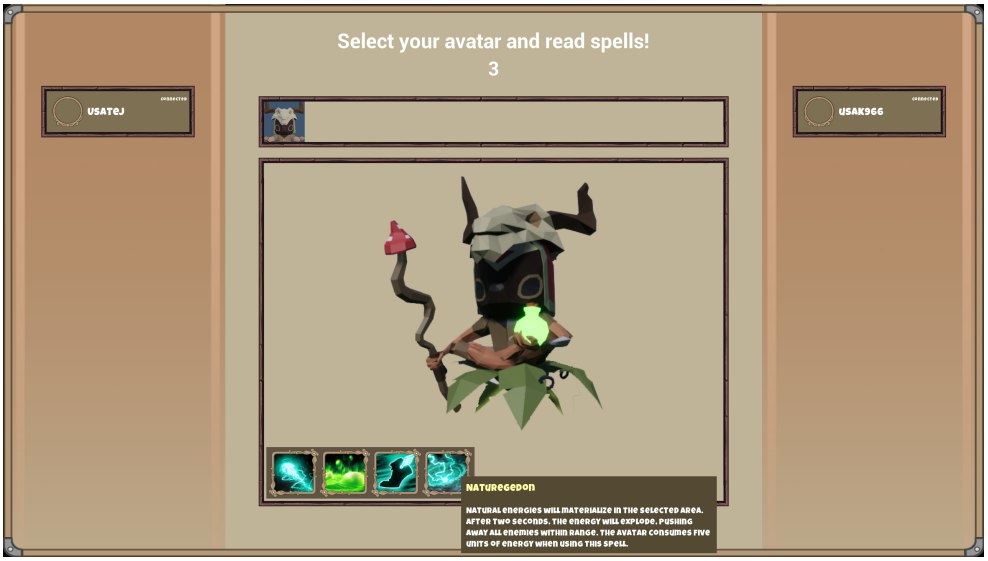
Příloha B

Obrázky ze hry

Obrázek B.1: Hlavní menu

Obrázek B.2: Přehled vlastních zápasů

Obrázek B.3: Obrazovka vlastního zápasu



Obrázek B.4: Výběr avatara

Obrázek B.5: Míření schopnosti „Venombolt“

Obrázek B.6: Schopnost „Naturegedon“

Obrázek B.7: Pohled na celou herní mapu. Tento pohled vidí hráč po úmrtí svého avatara.

Příloha C

Uživatelská příručka

C.1 Před spuštěním hry

Pro úspěšné spuštění hry je potřeba mít operační systém Microsoft Windows 10 a funkční internetové připojení. Spustitelný program je možné nalézt na přiloženém USB médiu ve složce „Binary“.

Každý uživatel musí vlastnit Epic účet a mít na něm zapnuté dvoufázové ověření. V případě, že ho uživatel nemá, může si ho vytvořit na adrese „https://www.epicgames.com/id/register“. Poté je nutné kontaktovat autora práce na emailu „sveigmic@cvut.cz“ s požadavkem o přidání do organizace (bez tohoto kroku se hráči nepůjde přihlásit). Není možné do organizace přidat uživatele, který nemá zapnuté dvoufázové ověření.

Uživatel vlastnící účet Steam má možnost propojení tohoto účtu s Epic účtem. Tím si může usnadnit přihlašování a získá ve hře přístup ke svým Steam přátelům.

C.2 Po spuštění hry

Pokud má uživatel spárovaný Steam účet a na pozadí spuštěný program Steam, uživatel je přihlášen automaticky. Jinak musí kliknout na tlačítko „Login“, čímž se mu otevře okno prohlížeče. Zde se přihlásí ke svému Epic účtu. Po dokončení přihlášení se může vrátit zpět do hry.

Na základě úspěšného přihlášení se zobrazí hlavní menu hry. V něm si uživatel může nastavit například rozlišení nebo obnovovací frekvenci. Tlačítkem „Play“ se otevře výběr se dvěma způsoby hledání zápasu. Způsob „Find Match“ slouží k hledání zápasu automaticky. V momentální situaci není doporučeno tento systém používat, jelikož systém zápas spustí až po úplném naplnění kapacity (osm hráčů). Preferovaným způsobem vyhledávání zápasu jsou vlastní zápasy („Custom Match“). Po přechodu do vlastních zápasů si může každý uživatel vytvořit vlastní zápas nebo se do již existujícího připojit. Vlast-

ník zápasu poté může kdykoliv zápas zahájit. Pokud je uživatel již připojen do místnosti, může na jakéhokoliv svého přítele kliknout pravým tlačítkem myši a vybrat možnost „Invite to lobby“.

Po zahájení zápasu jsou všichni jeho účastníci přesunuti do výběru avatara. Prototyp obsahuje pouze jednoho avatara, proto je avatar vybrán automaticky. V této obrazovce si hráči mohou prozatím přečíst pouze schopnosti tohoto avatara.

Poté, co skončí výběr avatara, jsou hráči přesunuti do hlavní herní úrovně. Cílem každého hráče je zůstat naživu jako poslední. Ovládání v herní úrovni je následující:

- **W, S, A, D** - klasický pohyb do osmi směrů,
- **Pohyb myší** - avatar se automaticky otáčí za kurzorem,
- **Q, E, R, F** - pomocí těchto čtyř kláves se používají schopnosti,
- **Levé tlačítko myši** - potvrzení míření schopnosti,
- **Pravé tlačítko myši** - zrušení míření schopnosti,
- **Držení levého ctrl** - přepnutí kamery do módu volného pohybu.

Příloha **D**

Obsah přiloženého USB

- `readme.txt` ..... stručný popis obsahu USB
- `Binary` ..... spustitelná forma prototypu
- `Source`
  - `Inpemo` ..... zdrojové kódy prototypu
  - `BVisionIntegration` ..... zdrojové kódy integrace služeb BVision
  - `Thesis` ..... zdrojová forma práce ve formátu LaTeX
- `Text`
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `gdc-inpemo.pdf` ..... herní návrhový dokument hry Inpemo
- `Screenshots` ..... obrázky ze hry
- `Video`
  - `abilities-demontration.mp4` ..... video ukázka schopností postavy
  - `camera-demonstration.mp4` ..... video ukázka kamerového systému
  - `gameplay-no-sound.mp4` ..... video ukázka celého zápasu (tři hráči)