FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Multiplayer hra pro mobilní zařízení využívající cloudová řešení
Student:	Bc. Michal Popek
Vedoucí:	Ing. Miroslav Balík, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Navrhněte a implementujte hru pro mobilní zařízení. Ve hře hráč buduje vlastní pevnost, brání ji a útočí na pevnosti ostatních hráčů. Základní požadavky na hru jsou:

1. Hráč může zobrazit, upravovat a vylepšovat svojí pevnost.
2. Hráč může zobrazit, upravovat a vylepšovat svojí útočnou jednotku.
3. Hráč může prohlížet nepřátelské pevnosti zvolené na základě jeho zkušenosti ve hraní hry a útočit na ně.
4. Podle útoků a obran hráče je vypočítané skóre, které si hráč může zobrazit a porovnat s ostatními.

Postup práce:
1. Navrhněte podobu hry a přesně specifikujte její funkční požadavky.
2. Analyzujte možnosti cloudových řešení, jejich využití ve hrách a vyberte vhodné řešení.
3. Navrhněte a implementujte serverovou i klientskou část hry pro OS Android. Volba vhodných technologií je součástí práce.
4. Vydejte hru v uzavřené beta verzi a otestujte s několika uživateli.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 7. února 2018

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Multiplayer hra pro mobilní zařízení využívající cloudová řešení

Bc. Michal Popek

Vedoucí práce: Ing. Miroslav Balík, Ph. D.

7. ledna 2019

Poděkování

Chtěl bych poděkovat své rodině za podporu po celou dobu studia, a dále pak vedoucímu práce za pomoc s formální stránkou práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické 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 předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 7. ledna 2019 .....

Odkaz na tuto práci

Popek, Michal. *Multiplayer hra pro mobilní zařízení využívající cloudová řešení.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Práce se zabývá vývojem hry pro více hráčů pro operační systém Android, která využívá cloudová řešení. Součástí je analýza cloudových poskytovatelů a vybrání vhodného pro tento typ hry. Cílem práce je navrhnout, implementovat a otestovat takovou hru a připravit ji pro vydání.

Klíčová slova cloud, hra pro více hráčů, Android, Unity

Abstract

This thesis deals with development of multiplayer game for Android operating system, which will use cloud solutions. Part of analysis is dealing with selection process of proper cloud provider for this kind of game. The objective of this thesis is to design, implement and test such a game and to prepare it for release.

Keywords could, multiplayer game, Android, Unity

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V dnešní době skoro každý člověk vlastní mobilní zařízení. Slouží pro velké množství praktických úkonů, od telefonování, přes textové zprávy až k navigaci v mapách či vzdálenému ovládání různých zařízení. Používají se ovšem také pro zábavu ve volném čase, což je kategorie, do které spadají i mobilní hry.

Velké množství těchto her využívá připojení k Internetu, ať už pro uložení hráčova postupu, nebo pro hru více hráčů. Tyto hry potřebují výkonné servery, a poslední dobou je nejefektivnější pro ně využívat cloudové služby. Server vystavěný v cloudu může výrazně snížit náklady za údržbu a zajistit větší stabilitu než vlastní stroje.

Není tedy divu, že se většina vývojářů dnes snaží svoje servery do cloudu umístit, a herní vývojáři nejsou vyjímkou. Cílem této práce je využít cloudové služby pro vyvinutí mobilní hry pro více hráčů. Hra tedy bude mít dvě části: klient a server. Obě jsou součástí práce.

Práce sleduje typický proces softwarového vývoje. V rámci analýzy definuje podobu hry, zkoumá možnosti pro cloudové řešení a rozebírá konkurenci. V kapitolách návrh a implementace jsou poté podrobně rozepsány postupy vývoje obou částí hry a nakonec je provedeno testování a jeho vyhodnocení.

Kapitola 1

Analýza

1.1 Specifikace požadavků na aplikaci

V této části analýzy je specifikována podoba hry a její název. Dále jsou zde uvedeny jednotlivé funkční a nefunkční požadavky na obě části aplikace.

1.1.1 Slovní popis hry

V mojí hře bude mít hráč za úkol budovat si svojí pevnost. Jejím středem bude zlatý důl, generující zlaťáky a zbylá část bude tvořena obrannými vežemi. Věže budou stylizované do základních barev (červené, zelené a modré) a jejich kombinací. Věže lze stavět libovolně po herní ploše a blokovat pomocí nich cestu útočníků. Jediná vyjímka jsou záchytná místa, kterými útočník musí projít, ty věžemi zablokovat nejde. Hráč dále vlastní armádu, kterou může vylepšovat a útočit s ní na pevnosti ostatních hráčů, pokud prolomí jejich obranu, tak jim ukradne určité procento zlaťáků.

1.1.2 Název hry

Vymyslet správný název hry je velice důležité. Je to úplně první dojem, který hráč ze hry bude mít. Podle [1] je potřeba se při výběru zaměřit na několik věcí.

Na prvním místě jedinečnost jména. Pokud už stejné jméno někdo má, ať už pro hru, nebo něco jiného, je potřeba se zastavit a zamyslet nad změnou. Pokud někdo bude hledat informace o hře, musí dostat relevantní výsledky, jinak hledání brzy vzdá a ztratíme tak potenciálního hráče. Není ovšem dobré snažit se získat originální jméno za cenu použití zvláštních znaků, nebo těžce vyslovitelných slov.

Pro výběr jména je pak vhodné zamyslet se, o čem hra bude, a s použitím několika slov zkusit shrnout její myšlenku. Můžeme klidně použít i obyčejná, často používaná slova. V tom případě je ovšem potřeba k nim přidat nějaký

dovětek. Jako příklad může být slovo „Magic“. Samo o sobě je široce používané a kdyby ho někdo vyhledával, asi by nenašel konkrétní hru. Když se k němu přidá ovšem další část, jako například „Might and Magic“ nebo „Magic: The Gathering“, dostaneme už názvy konkrétních her, které se s ničím nespletou.

Hra, kterou v rámci práce vyvíjím, je založená na žánru her Tower Defense, přesněji jeho odnoži, kde hráči bojují proti sobě. Tato odnož se nazývá Tower Wars. Vybrané jméno pouze přidává slovo Colorful, protože hra bude používat velice jednoduchou grafiku, obsahující pouze barevné kombinace a vylepšování věží bude probíhat jako míchání barev. Tím pádem název Colorful Tower Wars vystihuje dobře podstatu hry. Hráči budou válčit proti sobě a zároveň budou kombinovat věže podle barev.

1.1.3 Funkční požadavky

Tato sekce definuje funkční požadavky pro obě části aplikace.

Herní klient

- **Přihlásit hráče do hry na základě účtu** Uživatel není zatěžován vytvářením nového účtu a nutností zapamatovat si další přihlašovací údaje. Pro přihlášení bude použit jeho účet ze sociální sítě Facebook.
- **Zobrazit a editovat hráčovu pevnost** Hráč má možnost rozmístit věže tak, jak chce a vždy vidět jejich aktuální rozmístění.
- **Zobrazit a upravovat hráčovu armádu** Hráč má možnost prohlédnout si svojí armádu a dělat v ní změny. Může změnit pořadí v jakém jednotky útočí, nebo zadat povel k vylepšení jedné z jednotek.
- **Zadat povel k vylepšení věže, zobrazení průběhu a zrušení vylepšení** Hráč má možnost vybrat tři věže, které zkombinuje do jedné silnější, po zadání takového povelu uvidí průběh vylepšení a má možnost probíhající vylepšení zrušit.
- **Zobrazit pevnosti ostatních hráčů a dát povel na ně zaútočit** Hráč si zobrazí pevnost nepřítele a dostane vždy možnost provést útok, nebo zobrazit další. Poté, co se rozhodne zobrazit další se už nemůže vrátit zpět.
- **Zobrazit simulaci útoku na protivníkovu pevnost** Poté co se hráč rozhodne zaútočit na danou pevnost jiného hráče, zobrazí se simulace, jak se jeho armáda pokusí projít protivníkovou obranou.
- **Zobrazit hráčovo umístění v aktuálním žebříčku** Podle toho, jak hráč vyhrává nebo prohrává útoky, bude získávat hodnocení, které může porovnat s ostatními. Zobrazí se mu vždy jeho aktuální umístění a může

se podívat, jak jsou na tom ostatní hráči (posouvat si žebříček nahoru či dolů).

Server

- **Přihlásit hráče** Klient bude na server posílat přihlašovací údaje a potřebuje dostat informace o účtu, který je s nimi svázaný.
- **Vytvoření nového hráče** Pokud hráč v databázi serveru neexistuje, vytvoří nový účet.
- **Dát k dispozici informace o hráči, jeho pevnosti a armádě** Po přihlášení klient potřebuje tyto informace aby zobrazil hráči jeho účet.
- **Vyhodnotit příkaz pro vylepšení věže nebo jednotky** Když hráč zadá příkaz pro vylepšení věže nebo jednotky, server vyhodnotí, zda na to má právo a buď vylepšení zahájí, nebo dá vědět, že to nelze.
- **Dát k dispozici výběr možných protivníků a informace o jejich pevnostech** Klient potřebuje seznam protivníků, které může hráči ukázat. Ti budou vyhledaní tak, aby měli podobné umístění v žebříčku jako hráč.
- **Provést útok na daného hráče a dát k dispozici informace pro rekonstrukci simulace** Průběh útoku se bude simulovat na serveru a klient zpátky dostane jenom minimální nutné informace pro rekonstrukci. Hra běží na mobilních zařízeních, takže je nutné minimalizovat datový přenos.
- **Dát k dispozici žebříček hráčů** Klient potřebuje zobrazovat hráči jeho umístění oproti ostatním a to získává ze serveru.

1.1.4 Nefunkční požadavky

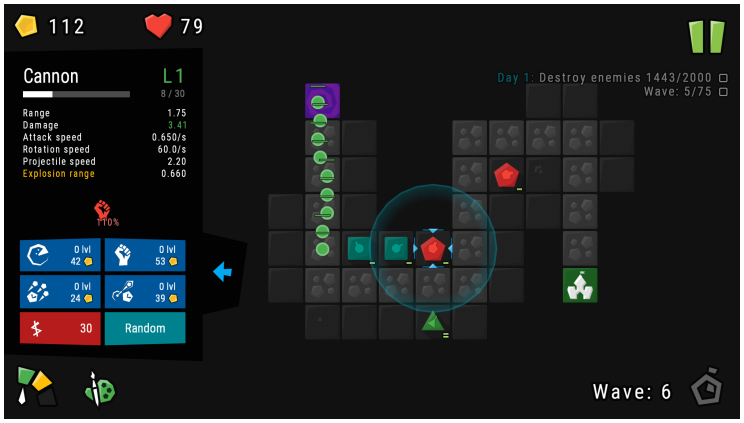
Tato sekce definuje nefunkční požadavky na aplikaci.

Herní klient Klient poběží na zařízeních s operačním systémem Android verze 5.1 nebo vyšší

Server Server poběží v cloudovém prostředí

1.2 Analýza konkurence

V žánru Tower Defense existuje nespočetné množství her, jak na mobilní zařízení, tak na stolní počítače. Jeho odnož pro více hráčů už ovšem tak zastoupená není. Na mobilních zařízeních, na které je v táto práci cíleno, byla



Obrázek 1.1: Rozehraná hra Infinitnode

nalezena pouze jedna, a i ta je kombinací několika žánrů. Většina vyzkoušených a hodnocených her jsou tedy klasické Tower Defense hry, které by mohly mít nějakou podobnost. Sekce se zaměřuje převážně na hodnocení hry pro více hráčů a herních mechanik při stavění věží.

1.2.1 Infinitnode

Infinitnode je klasická Tower Defense hra od společnosti Prineside. Na Google Play ukazuje kategorii 500 tisíc a více stáhnutí, patří tedy mezi úspěšné hry. Má docela jednoduchou 2D grafiku zaměřenou na geometrické tvary.

Herní mechaniky Hra podporuje jeden herní mód a to stavět dokud neprojde příliš mnoho jednotek a hráčovi dojdou životy. Používá přístup fixní cesty, na kterou nelze stavět věže. Na obrázku 1.1 lze vidět, jak taková cesta a věže kolem ní vypadají. Obrana tedy stojí mimo trasu, což trochu limituje hráčovu tvořivost. Věže mají velké množství vylepšení a několik možných typů útoku, zároveň také každá utočící jednotka má nějakou slabou a silnou stránku. Některé vlny nepřátel jsou i létající a většina věží do nich nemůže útočit a je nutné postavit věže speciální. Ve hře je důležitější, jak hráč věže vylepšuje, než kde přesně je postaví.

Ve hře se všechno točí okolo hlavního módu. Je možné vybrat těžší obtížnost a různé mapy s odlišnými cestami. Hráč si musí za získanou herní měnu odemykat nové věže a bonusy, které mu hru zjednoduší.

Multiplayer Hra nemá žádnou reálnou možnost, jak spolu mohou hráči interagovat. Mohou si jen porovnat skóre v žebříčku nejlepších výsledků, ale hra je prakticky singleplayer.

Porovnání Infinitnode představuje typickou Tower Defense hru, od které se hra vyvíjená v této práci z velké části liší. Oproti této hře budou v Colorful Tower Wars hráči útočit jeden na druhého, což by mohlo učinit hru atraktivnější. Dále mohou stavět věže i na cestu a tím ji změnit, nebude tedy nijak limitovaná hráčova tvořivost.

1.2.2 Defenders 2: Tower Defense CCG

Hra od společnosti Nival se pyšní na Google Play počtem instalací přes 1 milion. Oproti Infinitnode má tato hra grafiku výrazně komplikovanější. Kromě uživatelského rozhraní, je hra postavená z 3D modelů, na které herní kamera dívá ze shora pod lehkým úhlem.

Herní mechaniky Hráč má za úkol objevovat mapu, každý kousek mapy se zobrazí až poté, co hráč vyhraje příslušnou úroveň. Před každou bitvou hráč volí, jaké věže ze svých dostupných bude používat.

Přímo ve hře je nabízen zajímavý mix možností, kde veže stavět. Část herní plochy, v některých úrovních i celá, je fixně daná a hráč staví pouze podél cesty. V některých částích může umístěním věží cestu ovlivnit, avšak pouze velice limitovaně.

Každá úroveň má pouze několik vln nepřátel a poté co hráč vyhraje dostává odměnu ve formě nových věží, které jsou ukázany jako karty a odtud pochází část názvu hry CCG, což je zkratka pro Collectible Card Game. Hra sice není typická karetní, ale karty se objevují právě při sbírání nových věží, nebo jejich vylepšování. Pro každé vylepšení potřebuje hráč dostatečný počet karet dané věže viz 1.2, ty ovšem získává náhodně, což přináší do hry trochu nejistotu. Hráč neví, kdy bude mít dostatek karet pro vylepšení a může pouze doufat, že další úroveň mu dá to, co potřebuje.

Multiplayer Defenders 2 narozdíl od Infinitnode už pár prvků pro více hráčů přináší. Při prozkoumávání mapy může hráč narazit na budovy, které patří ostatním hráčům a když na ně zaútočí, získá bonusy té budovy a skóre do žebříčku pro více hráčů. Útok na cízí budovu se ovšem nijak výrazně neliší od útoků na počítačem kontrolované jednotky. Hráč, kterému budova patří nemá žádný vliv na to, jak bude probíhat útok.

Porovnání Hra vyvíjená v rámci této práce má hlavní výhodu ve hře pro více hráčů. Hráč v ní totiž aktivně ovládá, na co budou jeho protivníci útočit. Dále také stále nabízí větší volnost při stavění věží než tato hra. Defenders 2 sice dává lehce větší volnost při stavbě než Infinitnode, ale stále jsou možnosti hráče dosti omezené.

Obrázek 1.2: Vylepšení věže kombinováním dvou karet v Defenders 2

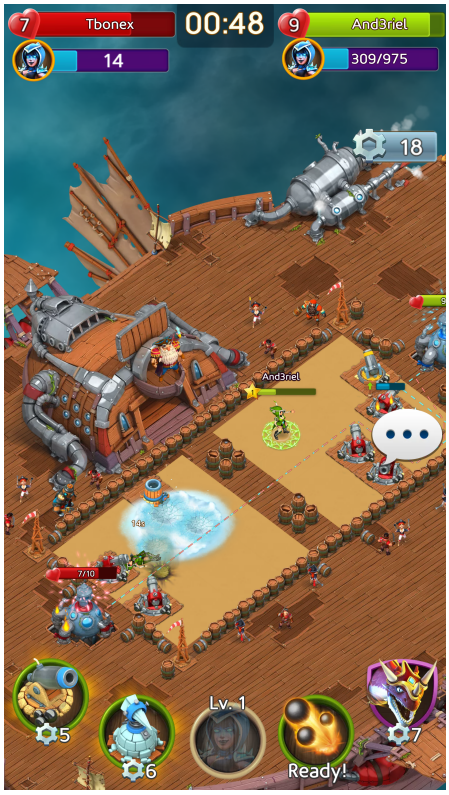
1.2.3 Survival Arena

Hra od společnosti Game Insight stojí obdobně jako Defenders 2 na hranici 1 milion instalací. Staví na podobné 3D grafice jako Defenders a na velkém množství efektů.

Herní mechaniky Narozdíl od předchozích her, Survival Arena nechává hráčovi plnou volnost stavět po celé ploše a vytvářet tak bludiště, kterým musí jednotky projít. Podobně jako v Defenders 2 má hráč k dispozici pouze několik věží ze začátku a další musí sbírat po jednotlivých výhrách a odemknutých odměnách.

Navíc ještě přidává speciální prvek. Kromě stavění věží hráč má i jednu jednotku, nazývanou hrdina, kterou může ovládat a zabíjet s ní nepřátele. Jednotka může chodit po mapě a ve hře pro více hráčů lze i s ní útočit na věže druhého hráče.

Multiplayer Survival Arena je primárně hra pro více hráčů a tomu také odpovídá velké množství možností, jak hrát proti ostatním.



Obrázek 1.3: Herní mód Duel ve hře Survival Arena

Jeden z módů je podobný předchozím hrám, je zde nazýván Turnaje, kde hráč hraje proti počítači a jeho nejlepší výsledek se bere jako jeho pozice v žebříčku. Tyto turnaje mají omezený čas a na konci hráči dostanou odměnu podle umístění.

Další zajímavý mód je Duel. Hrají zde dva hráči proti sobě. Oběma hráčům periodicky chodí nepřátelé, prohrává ten hráč, který první útoku podlehne. Hráči můžou posílat i jednotky navíc, nebo útočit na protivníkovu obranu pomocí hrdiny či kouzel. Herní mód je vidět na obrázku 1.3. Za výhry hráči dostávají odměny ve formě karet věží a hodnocení do žebříčku.

Porovnání Survival Arena má proti vyvíjené hře rozšířenější multiplayer, je zde více možností, jak soupeřit s ostatními hráči. Její slabší stránkou ovšem je grafika. Survival Arena má velice stylizovanou grafiku, která je cílená pravděpodobně na mladší generaci a části hráčů bude připadat dětinská. Oproti tomu Colorful Tower Wars bude mít grafické zpracování extrémě jednoduché a tím může Survival Aréně konkurovat.

1.3 Cloudové služby

Podle [2] se za cloudové služby označují všechny IT služby, které dává k dispozici cloudový poskytovatel a jsou přístupné přes Internet. Nahrazují infrastrukturu, která by normálně běžela na našem vlastním serveru, například aplikace nebo databáze. S jejich použitím nemusíme řešit instalaci, údržbu a zabezpečení hardwaru, to řeší poskytovatel za nás. Dále nám dávají vysokou flexibilitu, můžeme podle potřeby zvyšovat nebo snižovat zdroje, které potřebujeme. Dělí se na tři základní sekce podle [3].

Software jako služba Zkráceně SaaS je služba, která poskytuje uživatelům přístup k softwaru, který beží u dodavatele. Uživatelé nic neinstalují a používají buď webové rozhraní, nebo API. Hlavní výhody pro uživatele:

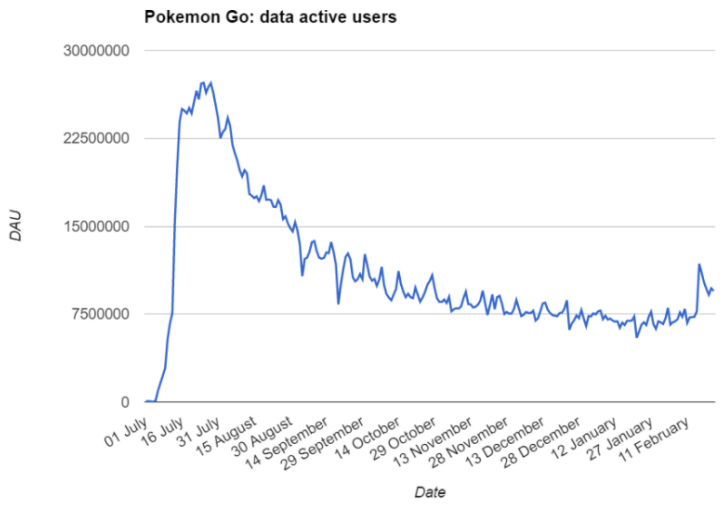
- Není potřeba nic instalovat ani nastavovat.
- Data jsou zabezpečená v cloudu a selhání zařízení nevede ke ztrátě dat.
- Aplikace jsou dostupné z jakéhokoliv zařízení připojeného k Internetu.

Platforma jako služba Zkráceně PaaS je služba, která dává k dispozici prostředí, v kterém mohou uživatelé vyvíjet, spravovat a dodávat aplikace. Prostředí obsahuje sadu předpřipravených nástrojů pro vývoj, úpravu a testování. Výhody pro uživatele:

- Testování, vývoj i běh aplikace probíhá v jednom prostředí.
- Dává možnost zaměřit se pouze na vývoj a nestarat se o infrastrukturu.
- Usnadňuje spolupráci týmů, které pracují vzdáleně.

Infrastruktura jako služba Zkráceně IaaS je služba, v rámci které jsou poskytovány výpočetní prostředky, například servery, systémy pro ukládání dat nebo síťové prostředí. Na tyto prostředky si pak uživatel nahraje svoji vlastní aplikaci nebo platformu. Hlavní výhody pro uživatele:

- Není potřeba nakupovat a instalovat hardware a platí se pouze za ten objem výpočetní síly, kterou uživatel použije.
- Lze rychle infrastrukturu přizpůsobovat podle aktuálních požadavků.
- Data jsou v cloudu, neexistují kritické body selhání.



Obrázek 1.4: Vývoj počtu hráčů v Pokemon Go [6]

1.3.1 Využití v mobilních hrách

Jak mobilní, tak ostatní hry mohou správným použitím cloudových technologií mnohé získat. Hlavní problém, který se ve hrách pro více hráčů řeší, je správně nadimenzovaný server. Před vydáním hry se špatně hodnotí, jak populární bude a tím pádem kolik výkonu bude potřeba. Málo výpočetní síly může znamenat neočekávané pády hry, což je jednoduchá cesta ke ztrátě hráčů. Pokud jí bude moc, tak vývojář zbytečně utratil peníze za servery navíc. Mobilní hry jsou známé tím, že jejich popularita může být dost nepředvídatelná.[4]

Například když SEGA vydala mobilní verzi hry Crazy Taxi, během prvního týdne získala 8 milionu uživatelů, na což zdaleka nebyla připravena. Měli vlastní servery ve vlastním data centru a neměli možnost takto rychle reagovat na nové uživatele, což způsobilo problémy s přihlašováním do hry a nefukčnost některých částí hry. Pro další hru už vývojáři zvolili server v cloudu. [5]

Hlavní výhoda použití cloudu pro mobilní hry spočívá v možnosti rychle reagovat na změny popularity. Pokud při vydání hru začne hrát velké množství lidí, dočasně zvýšíme počet serverů, a až populace hráčů začne klesat tak se postupně počet sníží. Důležité je, že stroje, které přidáváme a odebíráme v cloudu, nejsou naše a nemusíme řešit co s těmi, které už nepotřebujeme. Když se třeba podíváme na graf s počtem hráčů hry Pokemon Go 1.4, vidíme že kdyby měli servery nadimenzované na počet hráčů v prvním měsíci, tak během pár měsíců, kdy se čísla ustálí, tak víc než 50% jejich výpočetního výkonu bude zbytečných.

1.3.2 Výběr poskytovatele

Poskytovatelů cloudových služeb existuje velké množství, pro vybrání toho správného pro daný projekt je potřeba se zamyslet nad poměrem cena/výkon a hlavně nad tím, co přesně bychom od daného poskytovatele využili. Pro hru vyvíjenou v rámci této práce bude potřeba databáze, REST API a co nejmenší cena. Mezi nejvýznamější poskytovatele patří Amazon Web Services, Google Cloud Platform a Windows Azure. Právě u nich jsem s výběrem začal.

1.3.2.1 Amazon Web Services

Jeden z nejznámější cloudových poskytovatelů od Amazonu, zkráceně AWS. Jeho počátky jsou už v roce 2002, kdy vstoupil do první beta fáze s nabídkou SOAP a XML specifikací pro katalog produktů Amazon. V roce 2004 pak formou blogu začala firma dávat najevo, že chystá sadu služeb pro infrastrukturu a v roce 2006 pak vyšla první verze AWS, která zahrnovala tři služby : Simple Storage Service (S3), Elastic Compute Cloud (EC2) a Simple Queue Service (SQS).[7]

V dnešní době už zahrnuje služeb desítky, pro tuto práci jsou zajímavé hlavně EC2, Lambda a služby pro databázi.

AWS Elastic Compute Cloud EC2 je jedna z hlavních služeb AWS, zákazníkovi dává možnost vytvářet si libovolné instance virtuálních strojů přesně podle potřeby. Jako operační systém nabízí různé verze Microsoft Windows Server a několik desítek distribucí Linuxu. Jedna z hlavních předností je rychlost vytváření nových instancí v případě potřeby, nebo změny výpočetního výkonu aktuálních. Na to navazuje i služba EC2 Auto Scaling, která umožňuje nastavit automaticky podle aktuálních požadavků na výpočetní výkon, jak se mají servery zvětšovat či zmenšovat.

AWS Lambda Lambda je služba pro serverless computing, což je pojem pro službu, která zaručí provedení kódu a uživatel se vůbec nemusí starat o to, kde kód běží. Prostředí pro běh nastavuje a spouští poskytovatel. Do této služby tedy nahrajeme kód, který pak lze z různých míst spouštět. Ostatní služby AWS mají možnost spustit funkci v Lambdě a lze také spouštět přímo z kódu mobilních aplikací za pomoci knihovny, kterou AWS dává k dispozici. V tuto chvíli Lambda podporuje 5 programovacích jazyků : Python, Java, C#, Node.Js a Go.

AWS Database Services Amazon nabízí široký výběr databází, v čele se službou Relational Database Services, která dává k dispozici klasické relační databáze : MySQL, Oracle, SQL Server a PostgreSQL. Dále ještě nabízí vlastní relační databázi Aurora. Ta je optimalizovaná pro cloud a slibuje větší rychlost než MySQL či PostgreSQL. Z NoSQL databází nabízí svojí vlastní databázi

DynamoDB, která může být jak dokumentová databáze, tak key-value, a dále ještě grafovou databázi Neptune.

1.3.2.2 Google Cloud Platform

Cloudový poskytovatel od společnosti Google vydal první verzi svých služeb o pár let později než Amazon. První verzi zvanou Google App Engine vydal v roce 2008 jako ukázku pro prvních 10 tisíc vývojářů, jeho služby zahrnovaly dynamické webové stránky, persistentní úložiště a API pro autorizaci uživatelů. Od té doby průběžně přidává novější služby a jejich repertoárem se bez problému vyrovná Amazonu. [8]

Pro svoji práci se zaměřím hlavně na moduly Compute engine, Cloud functions a databáze.

Compute engine Svou nabídku virtuálních strojů nebízí Google ve službě nazývané Compute Engine. Podobně jako AWS nabízí výběr mezi distribucemi Linuxu, nebo Microsoft Windows Server. Nemá sice službu, která přidává stroje automaticky podle potřeby, ale slibuje velice rychlé spouštění dalších instancí a nabízí load balancing přes nastavené sady instancí. Zajímavé jsou také slevy, které Google dává za využití alokovaných strojů a za přislíbení delší spolupráce.

Cloud functions Google používá tuto službu pro nabídku serverless computing. Obdobně jako AWS Lambda, stačí pouze nahrát svůj kód a poté ho spustit. Nic dalšího z infrastruktury se neřeší. Nabízí možnost volat tyto funkce z ostatních svých služeb a zároveň z platformy pro mobilní zařízení Firebase. Jednu velkou nevýhodu Cloud functions ovšem vidím v omezení na programovací jazyky. K dispozici je pouze Node.Js.

Databáze Google nabízí několik různých služeb:

- Cloud SQL zahrnuje MySQL a PostgreSQL.
- Cloud Storage dává možnost uchovávat nestrukturovaná data.
- Cloud Datastore je dokumentová databáze.
- Cloud Bigtable, Cloud BigQuery jsou řešení pro BigData.

1.3.2.3 Microsoft Azure

Microsoft se s vývojem vlastního cloudu ještě o trochu opozdil, Microsoft Azure byl oznámen až v roce 2008 a první služby byly spuštěny pro veřejnost v roce 2010. Začínal pouze s nabídkou .NET a SQL služeb. Postupně se začal více soustředit na integraci služeb, na což Azure zavedl takzvané Biztalk

Services. Tyto služby dávaly možnost propojovat ostatní do logických celků a řetězit jejich volání. Dnes už Azure dohnal Google i Amazon a nabízené služby jsou podobné.[9]

Azure Virtual Machines Jak už název napovídá, toto je služba, kterou Azure používá pro svojí nabídku virtuálních strojů. Nabízí obdobné operační systémy jako konkurence, tedy Windows Server a různé distribuce linuxu. Narozdíl od ostatních poskytovatelů ovšem nabízí širokou nabídku obrazů systémů, které jsou předpřipravené pro určité činnosti.

Azure Functions Takto Microsoft nazývá svojí serverless implementaci. Nabízí obdobné možnosti volání těchto funkcí, buď z ostatních služeb poskytovatele, nebo z externích aplikací. Dává k dispozici jazyky : Javascript, C#, Java, Python, PHP, Bash, Batch a PowerShell.

Databáze Azure také nabízí větší množství databází na výběr:

- SQL Database, Azure Database for PostrgreSQL/MySql - několik základních možností pro klasickou SQL databázi,
- SQL Data Warehouse - výkonější databáze optimalizovaná pro analýzu dat,
- SQL Server Stretch Database - databáze se škálováním,
- Azure Cosmos - NoSQl databáze s globální distribucí a horizontálním škálováním,
- Table Storage - NoSQL Key-value databáze.

1.3.2.4 Porovnání

Všichni zmínění poskytovatelé nabízejí obdobné služby, při jejich porovnání je tedy hlavní faktor cena těchto služeb za stejný výkon.

Cena První věc, na kterou je potřeba se při porovnání ceny podívat je výpočetní výkon, který poskytovatelé dávají zadarmo. Uživatel si pak může služby zdarma zkusit, nebo případně nemusí platit nic v průběhu vývoje, kdy nepotřebuje velký výkon.

Tabulka 1.1 ukazuje přehled virutálních strojů, které dávají poskytovatelé zadarmo. Pojmem Bursting procesor se označuje systém přidělování CPU, který může dát až plný výkon procesoru na kratší dobu. Řídí se tím, že každou hodinu instance dostane přidělené množství času na procesoru, pokud ho nevyužije, tak se uloží do zásoby pro chvíli, kdy bude potřeba. Google Cloud platform dává k dispozici nejslabší stroj z těchto tří, avšak nemá omezenou

dobu, po kterou k dispozici bude, zbylí dva poskytovatelé dávají instanci pouze první rok po registraci.

-	Prvních 12 měsíců	Navždy zadarmo
Amazon Web Services	1 1 GB RAM, 3.3 GHz Bursting procesor	-
Google Cloud Platform	-	0.6 GB RAM, Bursting procesor
Microsoft Azure	1 GB RAM, Bursting procesor	-

Tabulka 1.1: Porovnání výkonu zdarma pro virtuální stroje

Pro serverless výpočty vidíme dle tabulky 1.2, že všichni poskytovatelé dávají nastálo 1 milion požadavků a 400 tisíc GB-vteřin. GB-vteřina je jedna vteřina výpočtu, který používá 1GB paměti. Google navíc dává ještě 1 milion požadavků, ale zároveň počítá i výpočetní sílu procesoru na GHz-vteřiny, pro kteoru dává limit taktéž 400 tisíc.

-	Prvních 12 měsíců	Navždy zadarmo
Amazon Web Services	-	1 Milion požadavků, 400 tisíc GB-seconds
Google Cloud Platform	-	2 Miliony požadavků, 400 tisíc GB-seconds, 400 tisíc GHz-seconds
Microsoft Azure	-	1 Milion požadavků, 400 tisíc GB-seconds

Tabulka 1.2: Porovnání výkonu zdarma pro serverless výpočty

Jak je možné vidět v tabulce 1.3, v nabídce databází zdarma vychází nejhůře Google s NoSQL úložištěm o velikosti pouze 1GB. Microsoft Azure na druhou stranu dává k dispozici velké úložiště o 250 GB, ale limituje ho na počet souběžných transakcí. AWS stojí někde uprostřed, limituje instanci výkonem spíš než přesným množstvím transakcí a dává větší úložiště než Google, navíc ještě oproti zbylým dvěma dává napořád 25GB ve svojí NoSQL databázi DynamoDB.

Funkcionalita Google, AWS i Microsoft nabízejí obdobné funkcionality co se týče databází a virtuálních strojů. Relační databáze, která je pro práci

-	Prvních 12 měsíců	Navždy zadarmo
Amazon Web Services	1 GB Ram, 1 CPU, 20 GB Storage	25GB DynamoDB
Google Cloud Platform	-	1 GB Cloud Storage NoSQL DB
Microsoft Azure	250 GB SQL databáze, 10 transakcí max	-

Tabulka 1.3: Porovnání výkonu zdarma pro databáze

potřeba je dostupná u všech tří a provedení serveru bude serverless. Důležité tedy je hlavně porovnání této funkcionality.

Google Cloud Platform pokulhává za ostatními v nabídce jazyků pro implementaci. Možnost pouze jednoho jazyka je vysoce limitující. Nejlépe na tom je naopak Microsoft Azure s osmi jazyky a poté AWS s pěti.

1.3.2.5 Výběr poskytovatele

Hlavní volba byla mezi AWS a Microsoft Azure. Google Cloud Platform byl z výběru vyřazen vzhledem k nízké flexibilitě jeho serverless implementace a nabídky pouze NoSQL databáze.

V kategorii služeb serverless vychází AWS i Azure skoro nastejno, nabízí shodné množství výpočetního výkonu zadarmo a nemají příliš limitovaný výběr programovacích jazyků. Rozdíl mezi nimi je až ve službách databází. Azure sice nabízí více úložného prostoru, ale v případě AWS je možné po uplynutí prvních 12 měsíců přejít na databází DynamoDB a pokračovat bez placení.

Zvolený poskytovatel pro hru implementovanou v rámci práce tedy je Amazon Web Services.

Kapitola 2

Návrh

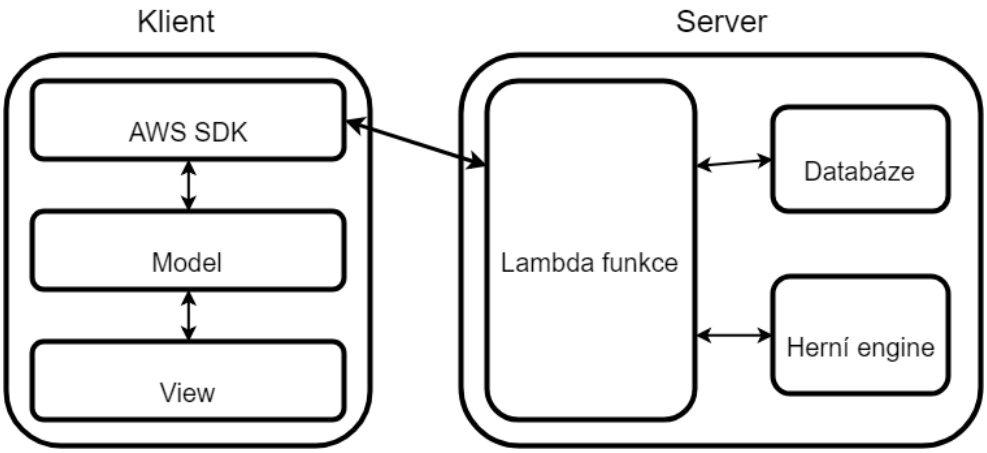
Systém se bude skládat ze dvou částí: Klient a Server. Každá z nich je dále rozdělena na menší samostatné celky, jak je možné vidět na obrázku 2.1.

2.1 Server

Tato sekce popisuje strukturu serverové části aplikace. Struktura se dá rozdělit na tři celky: databázi, lambda funkce a herní engine.

2.1.1 Lambda funkce

Serverless prostředí AWS Lambda funguje na pricipu funkcí. Uživatel v administraci vytvoří funkci, dá jí název, nahraje pro ni zdrojový kód a poté ji pod daným názvem může volat. Každá funkce má vstup i výstup ve formě



Obrázek 2.1: Architektura systému

JSON souboru. Implementace serveru je postavená na těchto funkcích, které jsou pak volány v klientovi pomocí AWS SDK. Celkově je potřeba pro běh hry následujících 12 funkcí.

1. **GetOpponents**

   Tato funkce slouží k vyhledání protivníků, na které může hráč útočit.

2. **SkipOpponent**

   Funkce SkipOpponent je používána ve chvíli, kdy se hráč rozhodne vynechat jednoho z nabízených protivníků.

3. **AttackOpponent**

   Poté co hráč vybral protivníka pomocí předchozích funkcí, tak se použije tato funkce pro provedení útoku.

4. **UpgradeCreep**

   Tato funkce zajišťuje vylepšování hráčových jednotek.

5. **GetRankings**

   Pomocí funkce GetRankings klient zjišťuje aktuální umístění hráče v žebříčku.

6. **StartTowerUpgrade**

   Tato funkce je použita pro spuštění vylepšování věží.

7. **FinishUpgrade**

   Funkce se použije ve chvíli, kdy je vylepšení hotovo a klient potvrzuje, že je výsledek zobrazen.

8. **UpdateCreepPositions**

   Tato funkce změní pořadí hráčových jednotek podle jeho požadavků.

9. **AuthenticateUser**

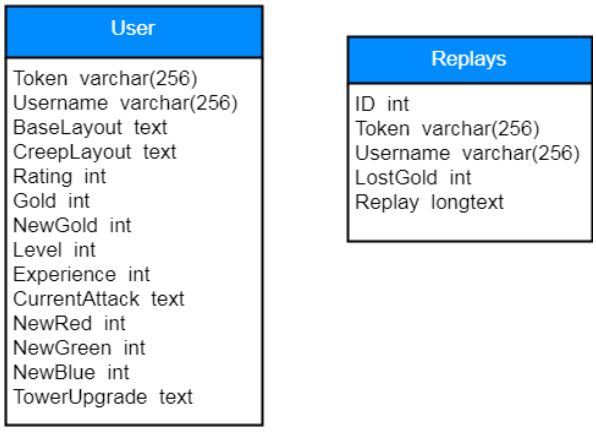
   Při vstupu do hry je potřeba hráče přihlásit, pro což slouží tato funkce. Dá klientovi všechny potřebné informace pro zobrazení hráčovy pevnosti.

10. **GiveGold**

    Tato funkce je použita pro navyšování množství zlaťáků, které hráči mají.

11. **SetUsername**

    Tato funkce se použije při prvním přihlášení hráče pro nastavení jeho jména.



Obrázek 2.2: Schéma databáze

12. **SaveLayoutChanges**

    Poté co hráč dokončí upravování své pevnosti, tak je použita tato funkce pro uložení změn.

2.1.2 Databáze

Server bude uchovávat data v rámci relační databáze typu MySQL, která běží v rámci služby Amazon RDS. Rozhodnutí použít MySQL databázi bylo na základě zkušeností právě s tímto typem databáze, a z dostupných služeb na Amazonu byla nejjednodušší na použití. Na obrázku 2.2 lze vidět, že schéma je velice jednoduché. Hra využívá pouze dvě tabulky, pro ukládání informací o hráči a pro záznamy útoků mezi hráči.

Users Tabulka uchovávající informace o hráčích hry. Každý hráč je identifikovaný řetězcem Token. Kromě základních informací, jako jsou zkušenosti, zlaťáky, nebo třeba úroveň, jsou v tabulce uloženy složitější hodnoty, které nejsou potřeba v dotazech na databázi a jsou tedy uloženy pouze jako reprezentace ve formátu JSON, místo jednotlivých hodnot.

BaseLayout Tato hodnota představuje rozložení hráčovy pevnosti. Má následující formát:

```
{
  "towers" : [{
      "x" : 0,
      "y" : 1,
      "id" : 0,
      "colorType" : 1,
      "level" : 1
    }, ...
  ],
  "path" : [{"x" : x1,"y" : y1},...]
}
```

Informace o pevnosti tedy drží dva hlavní údaje, jaké věže hráč má a jak vypadá cesta mezi nimi. Pole `towers` obsahuje informace o jednotlivých věžích. Každá vež má souřadnice, identifikátor, barvu a úroveň. Pole `path` obsahuje souřadnice dlaždic, kterými prochází cesta.

CreepLayout Zde je ukládána informace o jednotkách, které hráč má a v jakém jsou pořadí. Každá jednotka má informaci o životech a rychlosti. Formát JSON řetězce:

```
{
  "creeps" : [{
      "id" : 0,
      "hitpoints" : 50,
      "speed" : 1.7
    },
    {
      "id" : 1,
      "hitpoints" : 50,
      "speed" : 1.7
    },...]
}
```

CurrentAttack Toto pole ukládá informaci o aktuálním výběru protihráčů, který se hráči zobrazil. Slouží ke kontrole, že se hráč pokouší útočit na hráče, kterého reálně vidí. Má následující formát:

```
{
  "index" : 0,
  "opponents" : [{
     "token" : "tkn",
     "username" : "Username",
     "gold" : 100,
     "layout" : {...},
    },...
  ]
}
```

Pole `index` určuje, kterého nepřítele z pole `opponents` hráč v tuto chvíli vidí. Každý nepřítel má token pro identifikaci, uživatelské jméno, množství zlata a rozložení jeho pevnosti, které je stejné jako pole tabulky `BaseLayout`.

TowerUpgrade Zde se ukládá informace o probíhajícím vylepšování hráčových věží. Má následující formát:

```
{
  "towers" : [{..},{..},{..}],
  "timestamp" : 123456789,
  "result" : 25,
  "endTimestamp" : 123457890
}
```

Pole `towers` obsahuje tři objekty věží, z kterých se vylepšení vytváří, mají stejný formát jako objekty věží v `BaseLayout`. Dále JSON obsahuje informaci o výsledné kombinaci (daná konstantou) a začátek a konec vylepšení, obě jako unix timestamp.

Replays Tato tabulka ukládá informace o proběhlých útocích mezi hráči. V záznamu je token hráče, na kterého byl útok cílený, jméno útočícího hráče, množství ukradených zlaťáků a JSON reprezentace průběhu útoku. Její struktura je rozebrána v následující sekci.

2.1.3 Herní engine

Herní engine je komponenta, která bude provádět útoky a vypočítávat cesty mezi věžemi. Na obrázku 2.3 lze vidět návrh tříd enginu.

Pomocné třídy pro simulaci Engine obsahuje 4 třídy, které reprezentují jednotlivé entity, které se mohou při simulaci objevit.

Třída `Tower` reprezentuje věž. Její metoda `update` se stará o to, kdy má produkovat střely, které se získají metodou `get_missile`. Obě metody mohou být měněny potomky, čímž jde vytvořit libovolné chování věže.

Třída `Missile` reprezentuje střelu. Přes metodu `update` se řeší pohyb střely a `do_effect` poté aplikuje efekt na danou jednotku. Obě metody lze v potomcích měnit pro získání nového chování střely.

Třída `Creep` se stará o reprezentaci útočníka. Její `update` metoda se stará o pohyb.

Třída `Debuff` reprezentuje efekt, který působí na útočníka. Jeho metoda `update` postupně zraňuje útočníka, ke kterému je efekt přiřazen. `Start_effect` a `end_effect` se dají v potomcích použít pro akce, které se provedou při aplikaci efektu a při jeho odstranění.

Hlavní třída `Simulation` je hlavní třída enginu. Udržuje v sobě reprezentaci herního pole a využívá ostatní třídy pro zpracování algoritmů.

Metody `Set_target_location`, `set_starting_location` jsou použity k nastavení startu a cíle pro jednotky. Metody `add_tower`, `add_attacker` přidají do simulace věže a jednotky. Metody `find_path` a `calculate_simulation` poté provádějí hlavní algoritmy.

Metoda `find_path` slouží k hledání cesty mezi věžemi. Začíná v daném počátku a hledá nejkratší cestu do zadaného koncového bodu. Věže jsou nepřekonatelná překážka, kterou musí obejít. Je zde použit algoritmus A-star, který je podrobněji popsaný u implementace třídy `AStar`. Vrací seznam dvojic souřadnic x a y, každá z nich odpovídá jednomu úseku cesty.

Metoda `calculate_simulation` provede útok nastavený pomocnými metodami. Pracuje s třídami `Tower`, `Missile`, `Creep` a `Debuff`. Simulace probíhá po jednotlivých iteracích, kde jedna iterace odpovídá cca 33.3 ms reálného času. Iterace probíhají, dokud nejsou všechny jednotky buď mrtvé, nebo úspěšně došly do cíle. Průběh každé iterace se dá popsat následujícím pseudokódem:

```
do_iteration()
  if time_to_spawn_next_creep()
    spawn_creep()

  foreach tower in towers
    tower.update()
    if tower.new_missile
      append_result(tower.new_missile)

  foreach creep in attackers
    creep.update()
    if creep.state_changed
     append_result(creep.state)

  foreach missile in missiles
   missile.update()
   if missile.finished
   append_result(missile.end_info)
```

Na začátku iterace se vždy zkontroluje, jestli není čas vypustit dalšího útočníka a poté se provedou `update` funkce věží, útočníků a střel. Pokud věž vyprodukuje novou střelu, přidá se do seznamu střel a přidá se záznam do výsledku simulace. Stejně tak se přidávají do výsledku změny stavu útočníků a informace o zmizení střely.

JSON reprezentace záznamu Funkce `calculate_simulation` vrací reprezentaci záznamu ve formátu JSON. Vzhledem k tomu, že tento záznam bude posílán do mobilního zařízení, které ho bude zobrazovat, jsou na něj kladeny dvě hlavní podmínky: Průběh simulace musí jít ze záznamu bez chyby zreplikovat a záznam by měl obsahovat co nejmenší možné množství dat. Záznam má následující formát:

```
{
  "creeps" : [{
    "created" : 1500,
    "checkpoints" : [
      {"timestep" : 1550, "x" : 1, "y" : 2},
      {"timestep" : 1675, "x" : 10, "y" : 2},
      ...],
    "damaged" : [{"timestep" : 1765, "value" : 10},...],
    "state" : [{"timestep" : 1765, "value" : [1,2,3]},...],
    "deleted" : 1750
  },...],
  "missiles" : [{
    "created" : {"timestep" : 1750, "x" : 5, "y" : 4},
    "type" : 0,
    "deleted" : {"timestep" : 1765, "x" : 7, "y" : 9}
  },...],
  "lastTimestep' : 2450
}
```

V záznamu jsou přenášeny informace o pohybu útočníků a střel, předpoklad je, že zařízení ve chvíli, kdy přehrává záznam už má uložené rozložení věží, cesty mezi nimi a informace o jednotkách, které útočí. Ještě také obsahuje počet iterací, který simulace trvá(`lastTimestep`).

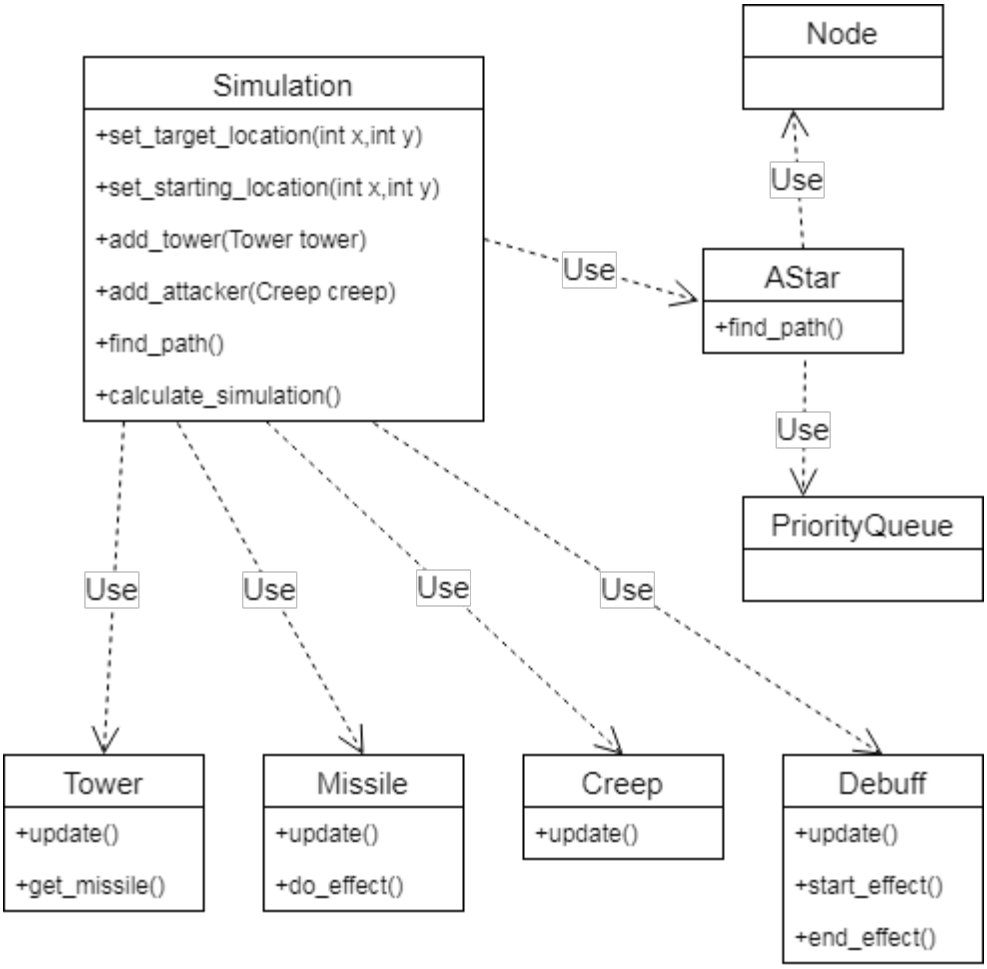
Útočníci Pole `creeps` obsahuje informace o útočnících. Každý má číslo iterace, ve které vznikl (`created`) a ve které zmizí z herní plochy (`deleted`).

Informace o pohybu jsou uložené v poli `checkpoints`, je to seznam trojic číslo iterace, x a y. Každá trojice reprezentuje okamžik v čase daný číslem iterace a polohu danou souřadnicemi x a y. Mezi jednotlivými záznamy se útočník pohyboval stejnou rychlostí, lze z nich tedy spočítat jeho polohu v každé iteraci. Například pro dvojici `{"timestep" : 1750, "x" : 5, "y" :5}` a `{"timestep" : 1760, "x" : 5, "y" :10}` spočítáme, že v iteraci 1752 byl na pozici `[5,6]`.

Pole `damaged` reprezentuje životy útočníka v průběhu simulace. Jednotlivé hodnoty udávají, že v iteraci s daným číslem ztratil tento útočník daný počet životů. Zařízení, které simulaci přehrává má informaci o začínající hodnotě životů jednotky a stačí mu tedy vždy ve správný okamžik odečíst.

O stavu útočníka vypovídá pole `state`. Obsahuje vždy číslo iterace a hodnotu stavu, která v dané iteraci začala. Hodnota se skládá z identifikátorů jednotlivých efektů, které na sobě útočník má.

Střely Střely jsou reprezentovány polem `missiles`. Každá obsahuje čas a místo vzniku (`created`) a zániku (`deleted`). Dále pak ještě typ střely, podle



Obrázek 2.3: Návrh tříd herního enginu

kterého se určuje její barva. Dráha letu se reprodukuje velice jednoduše, střela se rovnoměrně pohybuje mezi místem vzniku a místem zániku, tak aby dorazila na místo v té iteraci, která je napsaná u zániku.

2.2 Klient

Klient je vyvíjen v herním Enginu Unity a je rozdělen na tři hlavní části: View, Model a AWS SDK.

2.2.1 AWS SDK

AWS SDK je knihovna od Amazonu připravená pro použití v Unity, zajišťuje v projektu autorizaci přístupu k serveru a komunikaci s ním. Dává možnost

přímo volat funkce ze služby Lambda, není tedy nutné je vystavovat jako veřejné REST API.

2.2.2 Model

Tato část klienta zodpovídá za ukládání herního stavu a za ovládání komunikace se serverem. Na obrázku 2.4 lze vidět navrženou strukturu tříd.

Komunikace s View Veškerá komunikace s View klienta probíhá přes třídu `GameLogicFacade`, která používá návrhový vzor fasáda. Tato třída tedy neobsahuje skoro žádnou herní logiku, a pouze přeposílá požadavky na zodpovědné implementační třídy.

Třídy zodpovědné za komunikaci se serverem Pro každou Lambda funkci, která je na serveru, existuje v modelu třída, která ji reprezentuje. Všechny tyto třídy dědí od `ServerRequest`, což je třída, která zaobaluje použití AWS SDK. Zodpovídá za zavolání dané lambda funkce a předání všech parametrů, zpracování výstupu pak předá potomkovi. Jednotlivé třídy, které z ní dědí, tedy řeší pouze vstupní a výstupní data konkrétní lambda funkce.

Třídy zodpovědné za herní logiku Model obsahuje pět tříd, které zodpovídají za herní logiku různých částí hry.

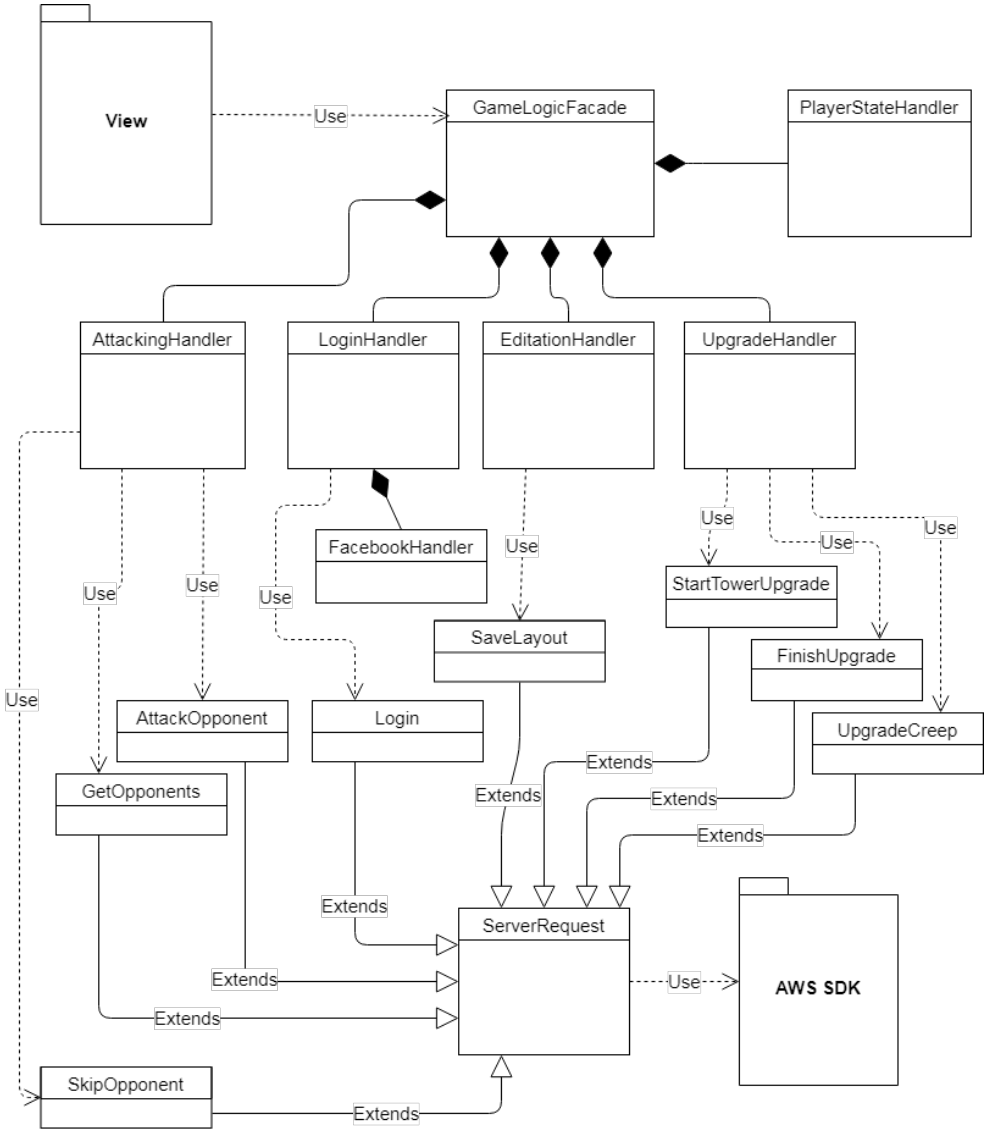
1. **PlayerStateHandler** Třída, která je zodpovědná za držení stavu hráče a jeho pevnosti. Ukládá a mění hráčovo zlato, zkušenosti, úroveň a reprezentaci pevnosti. Sama o sobě nevolá žádnou lambda funkci, pouze se do ní přesměrují výsledky funkcí z jiných tříd, které ji zajímají. Informace z přihlášení nebo z výsledků útoků.

2. **AttackingHandler** Tato třídá zodpovídá za ukládání a updatování stavu útoku. Používá tři lambda funkce : `AttackOpponent`, `GetOpponents` a
`SkipOpponent`. Tyto funkce odpovídají získání informací o pevnostech hráčů, na které lze útočit, přeskočení jednoho hráče a zaútočení na hráče.

3. **LoginHandler** Přes tuto třídu probíhá přihlášení uživatele do systému. Pomocí třídy `FacebookHandler`, která obaluje knihovnu pro komunikaci s Facebookem, získá token pro přihlášení přes lambda funkci `Login`.

4. **EditationHandler** Třída zodpovědná za upravování hráčovy pevnosti. Udržuje změny, které hráč v upravovacím módu udělal, a až hráč editaci ukončí, tak je pomocí lambda funkce `SaveLayout` uloží na server.

5. **UpgradeHandler** Poslední třída řešící logiku zodpovídá za vylepšování věží a jednotek. Používá lambda funkce `UpgradeCreep` na odeslání změn



Obrázek 2.4: Návrh tříd komponenty Model

jednotek a `StartTowerUpgrade` a `FinishUpgrade` na puštění či ukončení vylepšování věže.

2.2.3 View

Komponenta View zodpovídá za zobrazování herního stavu hráči. Z obrázku 2.4 je vidět, že využívá třídu `GameLogicFacade` pro komunikaci s modelem.

Zároveň odchytává vstup z obrazovky zařízení a propaguje ho zpátky do modelu.

Komponentová Architektura Tato část klienta je vytvořena za pomoci komponentové architektury použitého herního enginu. Komponentová architektura se používá, když jeden prvek je potřeba na více místech. Kód prvku je přemístěn do třídy komponenty a prvek se pak stává jenom nádobou pro skupinu komponent. [10]

Struktura Jednotlivé prvky vytvořené pomocí komponentové architektury jsou ovládané třídou `UIController`. Tato třída zodpovídá za komunikaci s Modelem a za zobrazení prvků uživatelského rozhraní, které odpovídají stavu modelu a hráčovým vstupům.

Kapitola 3

Implementace

V této kapitole je podrobněji rozepsáno, jakým způsobem byly jednotlivé části hry implementovány.

3.1 Server

První část implementace se zabývá herním serverem, který beží v prostředí AWS Lambda.

3.1.1 Lambda funkce

Hlavní část implementace serveru jsou Lambda funkce, které zpracovávají požadavky od herního klienta.

Použité technologie Všechny Lambda funkce jsou psané v Pythonu verze 3.6. Pro komunikaci s databází využívají knihovnu PyMySQL.

Komunikaci s databází Všechny lambda funkce v projektu potřebují komunikovat s databází, tato komunikace je implementována ve třídě `Database`. Třída zaobaluje vytváření SQL dotazů a jejich posílání přes knihovnu PyMySQL.

Nasazení do cloudu Pro nasazení jednotlivých funkcí je potřeba v administraci AWS Lambda vytvořit funkci, přiřadit jí název a nahrát kód. Nahrávání kódu probíhá buď přes ZIP soubor, nebo pro soubory větší na 10MB přes službu AWS S3. Poté už stačí pouze zadat název souboru a funkce, která se v něm má spouštět. Musí brát dva argumenty: event a context. Event obsahuje vstupní data a context detaily o volání funkce.

3.1.1.1 Jednotlivé funkce

Všechny funkce, kromě jedné vyjímky, přijímají na vstupu řetězec nazývaný token, podle kterého identifikují hráče, který funkci volá.

GetOpponents Tato funkce nejdříve získá z databáze informace o hráči, který funkci volal. Podle jeho úrovně a hodnocení spočítá minimální a maximální úroveň a hodnocení, kterou mohou mít vyhledávaní soupeři. Poté dokud nemá alespoň dva protivníky, tak vždy zkusí vyhledat nové s aktuálním rozpětím úrovně a hodnocení. Nakonec hráčovi uloží v databázi že dostal nalezené protivníky a vrátí je jako výsledek.

SkipOpponent Funkce upraví hráčův záznam v databázi. Zvýší index vybraného protivníka v uložených protivnících z funkce `GetOpponents`.

AttackOpponent Nejdříve kontroluje, že je volána s validními argumenty. Hráč který funkci volá musí mít v uložených protivnících z `GetOpponents` na indexu, který mohl být upraven podle `SkipOpponent` protivníka s tokenem, který přišel jako parametr funkce. Pokud tato podmínka neplatí funkce vrátí vyjímku.

Dále vytvoří instanci třídy `Simulation` z herního enginu, naplní jí věžemi a jednotkami a spustí simulaci. Podle výsledku simulace spočítá odměnu pro hráče a změní jeho záznam v databázi. Nakonec uloží reprezentaci simulace do replays, aby k ní dostal přístup hráč, který byl napadán, až se příště přihlásí. Nakonec vrátí jako výstup průběh simulace a odměnu, co se má hráči zobrazit.

UpgradeCreep Funkce zkontroluje, že má hráč dostatek zlaťáků pro dané vylepšení a následně upraví záznam v databázi. Hráči ubere zlaťáky a změní hodnoty u jednotek, které jsou dané vstupními hodnotami.

GetRankings Funkce získá z databáze seznam hráčů, seřazený podle hodnocení. V něm najde hráče, který funkci volal a do odpovědi přidá na jakém indexu se hráč nachází. Zjednodušuje tak zobrazení pro klienta, který informaci nemusí sám hledat.

StartTowerUpgrade Nejdříve zkontroluje vstupy. Vrací vyjímku v několika případech: když hráč už vlastní probíhající vylepšení, když některá z věží ve vstupech nebyla nalezena nebo když hráč nemá dostatek zlata na zaplacení upgradu.

Následně vytvoří informaci o vylepšení, která obsahuje seznam vylepšujících se věží, čas začátku, čas konce a výslednou věž, která bude z vylepšení vytvořena. Tuto informaci uloží do databáze a odečte hráči příslušnou hodnotu zlata.

FinishUpgrade Zkontroluje, pokud hráč má uložené probíhající vylepšení a jestli už nastal čas, kdy je možné vylepšení ukončit. Pokud ano, tak vylepšení z databáze vymaže a změní věže, kterých se týkalo. Nakonec vrátí výsledek vylepšení.

UpdateCreepPositions Ze vstupu vezme id jednotek, kterým se mění umístění a jejich novou pozici. Poté podle těchto hodnot zpřehází uložení jednotek v databázi.

AuthenticateUser Nejdřív zkontroluje vstup, pokud nepřišel token, nebo je prázdný, končí vyjímkou. Poté se pokusí získat záznam o hráči z databáze. Pokud ho nenajde, hráče vytvoří s nastavenými počátečními hodnotami.

Před návratem hodnot uživateli provede několik akcí. Pokud má hráč nějaké nové zlato, v databázi ho přesune do aktivního zlata. Dále pokud má probíhající vylepšení, pokusí se ho zavoláním `FinishUpgrade` dokončit, a jako poslední zkntroluje, jestli má hráč v databázi nějaké informace o útocích, které na něj někdo provedl.

Všechny získané informace pošle jako odpověd.

GiveGold Jednoduchá funkce, která upraví všem hráčům záznam v databázi a tím jim přidá zlaťáky. Do kolonky `NewGold` přičte hodnotu 20.

Jako jediná není volána nikdy z klienta a nemá žádná vstupní data. Periodicky každou půl hodinu je spouštěna na serveru, pomocí služby AWS CloudWatch Events.

SetUsername Uloží do záznamu hráče v databázi uživatelské jméno, které přišlo na vstupu.

SaveLayoutChanges Nejdříve uloží do databáze nově vzniklé věže, poté aplikuje změny míst existujících věží. Pokud je ve změnách věž, která v záznamu na serveru není, končí vyjímkou. Poté se pomocí volání `find_path` na vytvořeném objektu `Simulation` pokusí najít a uložit novou cestu mezi věžemi, pokud cesta není nalezena, končí vyjímkou. Nakonec vrátí novou hodnotu cesty, protože všechny ostatní informace už klient má. Pokud funkce neproběhne do konce a skončila by vyjímkou, všechny změny věží se vrátí zpět a databáze je ve stavu, v jakém byla před voláním funkce.

3.1.2 Herní engine

Tato komponenta je taktéž psaná v Pythonu 3.6 a některé její části využívají i Cython.

3.1.2.1 Hledání cesty mezi věžemi

Jednou z funkcí herního enginu je vyhledání cesty mezi věžemi. K tomu slouží implementace algoritmu A* ve třídě `AStar`.

Algoritmus A* Algoritmus funguje na principu dvou seznamů: otevřený a uzavřený. Otevřený seznam obsahuje všechny pozice, které jsou kandidáty na prozkoumání. Uzavřený seznam obsahuje pozice, které už prozkoumány byly. Otevřený seznam začíná s počáteční pozicí a uzavřený seznam je prázdný.

V každém cyklu algoritmu se pak vezme pozice z otevřeného seznamu, která má nejlevnější ohodnocení, ohodnotí se její sousedé a přidají se do otevřeného seznamu. Pokud některý ze sousedů již je v uzavřeném seznamu, je ignorován. Zpracovávaná pozice se sousedům přiřadí jeko jejich předchůdce a přidá se do uzavřeného seznamu. Algoritmus končí ve chvíli, kdy zpracovávaná pozice je cílová pozice, v tu chvíli jsme našli cestu a její rekonstrukci provedeme pomocí řetězu předchůdců jednotlivých pozic.

Ohodnocení pozic se skládá ze dvou složek: vzdálenosti z výchozího bodu a heuristického odhadu vzdálenosti do cíle. Jako heuristika je v algoritmu použita takzvaná manhattanská vzdálenost, což je absolutní hodnota součtu rozdílů souřadnic na ose x a na ose y mezi daným bodem a cílem.[11]

3.1.2.2 Simulace útoku

Jak již bylo zmíněno v návrhu, simulaci útoku obstarává třída `Simulation`. V její metodě `calculate_simulation` se postupně volají funkce update přes všechny herní objekty, což jsou instance tříd `Missile`, `Creep` a `Tower`. Dále ještě jednotky mohou obsahovat instance tříd `debuff`.

Tower Tato třída se v základní podobě vytváří s následujícími parametry:

- `x`, `y` - pozice věže,
- `firerate` - frekvence střílení,
- `radius` - dostřel věže,
- `damage` - zranění, které věž způsobuje,
- `target_limit` - počet střel, které může vystřelit najednou.

Její metoda `update`, volaná v každé iteraci, vždy nedříve kontroluje, jestli uplynul čas od poslední střely větší než `firerate`. Pokud ano, tak dále kontroluje pro všechny jednotky jejich vzdálenost od věže a jakmile najde nějaké, které jsou blíže než `radius` tak vytvoří pomocí metody `get_missile` základní střelu a přidá ji do seznamu aktivních střel. V jednom takovém střílení může vytvořit pouze tolik střel kolik udává hodnota `target_limit`.

Missile Třída reprezentující základní střelu, bere následující parametry:

- `x`, `y` - pozice kde střela je vytvořena,
- `target` - jednotka, za kterou letí,
- `damage` - zranění, které způsobí jednotce až do ní narazí,
- `speed` - rychlost, jakou střela letí.

V metodě `update` nejdříve zkontroluje, jestli už cílové jednotka neumřela a poté vždy upraví svůj směr letu podle aktuální podoby jednotky a v tomto směru se pohne o hodnotu `speed`. Nakonec zkontroluje, jestli jednotku nedohnala a v tu chvíli zavolá metodu `do_effect`, která v základu zraní jednotku o hodnotu `damage`.

Creep Třída pro reprezentaci jednotek. Jako parametry bere:

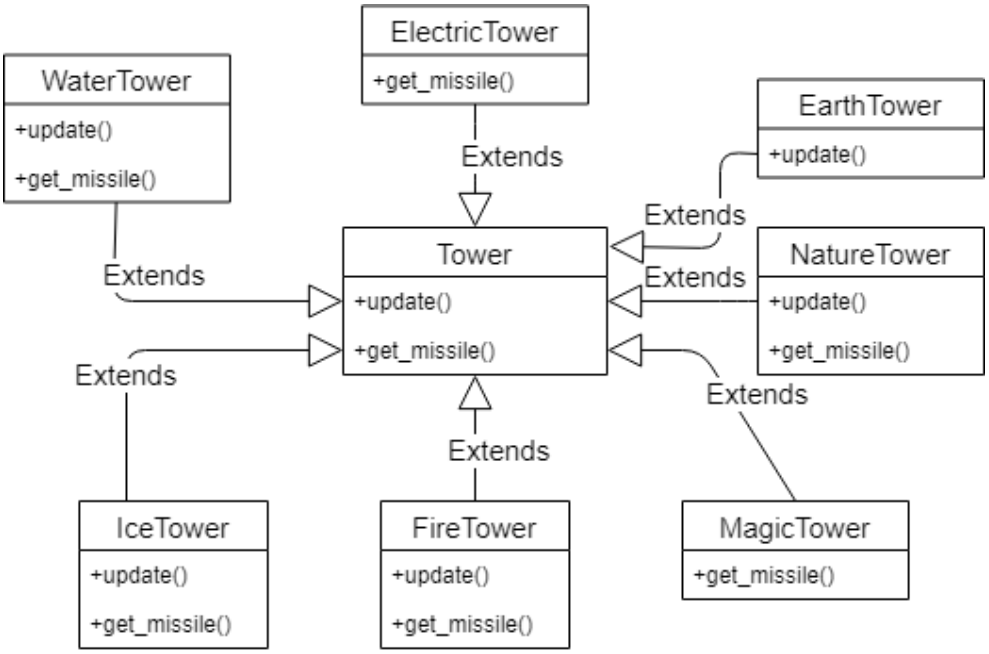
- `x`, `y` - startovní pozice jednotky,
- `hitpoints` - počet životů,
- `path` - cesta, kterou má jednotka sledovat,
- `speed` - rychlost.

Metoda `update` provádí dvě věci: Pohybuje jednotkou po cestě a volá `update` na efektech, která jsou na ní aktivní. Pohyb po cestě probíhá následovně: Máme zvolený bod cesty, který je aktuální cíl, k němu se každý cyklus jednotka pohne o hodnotu `speed`. Pokud se ocitne na místě aktuálního cíle, vezme si z `path` následující, pokud už další není, dorazil na konec cesty a jednotka se úspěšně zničí.

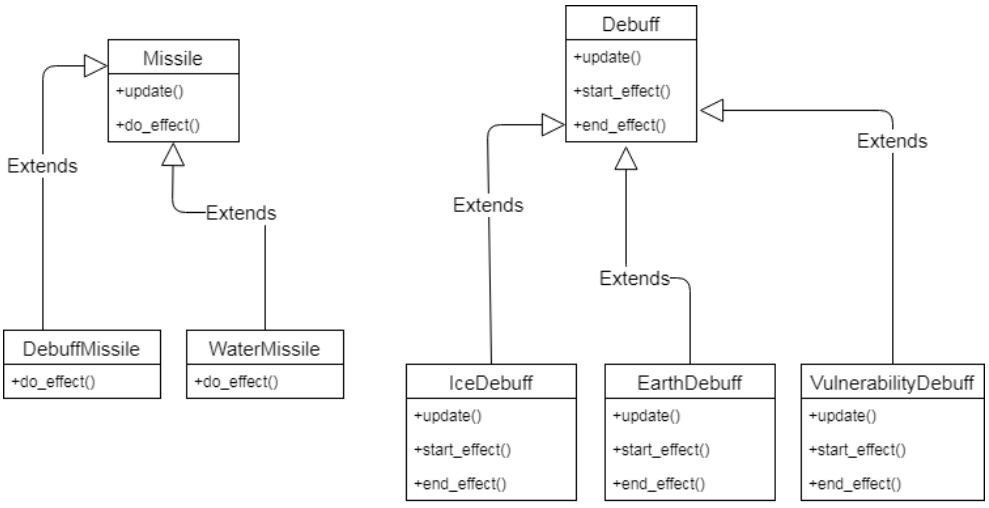
3.1.2.3 Speciální věže

Hra obsahuje větší množství věží a pouze některé z nich jsou implementovány pomocí základní třídy `Tower`. Pro ostatní věže s pokročilou logikou byly implementovány nové třídy z ní dědící, jejich přehled je možné vidět na obrázku 3.1. Pomocné třídy, které dědí z tříd `Missile` nebo `Debuff` jsou vidět na obrázku 3.2.

První typ speciálních věží jsou ty, které aplikují na jednotku nějaký efekt. Věže tohoto typu modifikují pouze metodu `get_missile` kde vytvoří třídu `DebuffMissile` dědící z `Missile`. Tato třída drží instanci třídy `Debuff` a kromě zranění jednotky, jí tuto instanci předá. Na tomto principu fungují 4 věže: `IceTower`, `FireTower`, `MagicTower` a `ElectricTower`. `MagicTower` a



Obrázek 3.1: Diagram tříd speciálních věží



Obrázek 3.2: Diagram tříd speciálních střel a efektů

`IceTower` využívají pro efekt třídu `IceDebuff`, která na začátku svého působení sníží rychlost jednotky a na konci ji vrátí zpátky. `FireTower` používá základní efekt a `ElectricTower` má svůj vlastní: `VulnerabilityDebuff`. Tento efekt na začátku zvýší koeficient zranění jednotky a na konci svého působení ho zase vrátí zpět.

Věž `NatureTower` má vlastní implementaci metody `update`, ve které v periodě určené parametrem z konstruktoru vystřelí na všechny jednotky kolem namísto jedné. `EarthTower` také mění metodu `update`, vyměňuje v ní střílení věže za aplikování efektu `EarthDebuff`, který snižuje rychlost a zvyšuje koeficient zranění. Poslední speciální věží je `WaterTower`, ta vytváří svojí speciální střelu `WaterMissile`, která neskončí u prvního nepřítele, kterého zasáhne, ale pokračuje k několika dalším.

3.1.2.4 Optimalizace

Jak hledání cesty, tak simulace útoku, jsou náročnější výpočty a jejich běh trvá jednotky až desítky sekund, záleží na složitosti rozložení dané pevnosti. Proto jsou nejčastější a nejnáročnější matematické výpočty přesunuty do vlastního souboru a místo Pythonu napsané v Cythonu.

Cython je programovací jazyk podobný Pythonu, který lze převézt a zkompilovat jako jazyk C. Čímž se dá získat oproti Pythonu velké zrychlení. Při práci s nativními objekty Pythonu zrychlení nebude příliš velké, ale pro číselné výpočty, které se na strukturu Pythonu nevážou, může být zrychlení obrovské.[12]

Pro zpracování v Cythonu tedy byly vybrány dva nejčastější matematické výpočty. Vzdálenost mezi dvěma body, která je použitá v každém cyklu simulace pro každou střelu a pro každou věž, a výpočet směru letu střely z její aktuální polohy a z polohy jejího cíle.

3.2 Klient

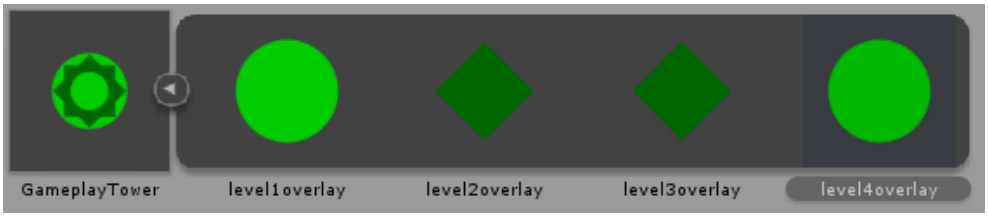
Druhá část implementace je klientská aplikace vyvinutá v herním enginu Unity pro platformu Android.

3.2.1 View

V této části implementace klienta je diskutováno, jak bylo docíleno podoby zobrazení hry uživateli.

3.2.1.1 Zobrazení herní plochy a simulace útoku

Herní pole Herní pole je mřížka o rozměrech 20x20. Pro každé pole z mřížky je vytvořen herní objekt s čtvercovou texturou, bílý pro pole kam lze stavět věže a červený pro pole, kde to nelze. Zlatý důl je zobrazen jako žlutý kruh.



Obrázek 3.3: Skládání vrstev zobrazení věže

Cesta, po které chodí útočníci, je zadaná jako soubor souřadnic polí mřížky přes které vede, zobrazena je pomocí hnědých obdélníků, které jsou mezi každými dvěmi následujícími souřadnicemi.

Věže jsou zobrazeny pomocí 4 prvků, které jsou vrstveny přes sebe a tím odlišují jednotlivé úrovně věže. Na obrázku 3.3 jsou vidět jednotlivé vrstvy a výsledná věž čtvrté úrovně. První úroveň má pouze barevný kruh, zhruba velikosti jednoho pole mřížky. Druhá úroveň přidává čtverec o trochu menší než původní kruh, který má lehce tmavší barvu. Třetí úroveň přidá ten samý kruh, ale rotovaný o 90 stupňů, a čtvrtá úroveň přidává ještě jeden malý kruh doprostřed na čtverce. Jednotlivé typy věží se pak odlišují barvou.

Simulace Z informací, které příjdou ze serveru je nutné zreplikovat průběh útoku, k tomu slouží třídy: `ReplayController`, `MissileReplayController`, `CreepReplayController` a `CreepReplay`.

Třída `ReplayController` dostane JSON ze serveru a zaznamená čas začátku simulace. Podle něj poté v každém cyklu herního enginu předá aktuální čas simulace třídám, které se starají o střely a útočníky.

`CreepReplayController` na začátku simulace vytvoří pro každého útočníka herní objekt s komponentou `CreepReplay`, které předá informace o jednom útočníku. Při každém cyklu pak pouze distribuuje získaný aktuální čas simulace těmto objektům.

`CreepReplay` se stará o simulaci jednoho útočníka, vzhledem k aktuálnímu času simulace, který pravidelně dostává. Pokud je čas mimo rozsah, kdy je daný útočník naživu, přemístí herní objekt mimo obrazovku. Dále třída řeší stav útočníka, má seznam změn stavů, které jdou chronologicky za sebou. Pokaždé, když čas překročí jednu změnu, překreslí se barva útočníka, aby odpovídala stavu. Obdobným způsobem jako stav, se mění i životy jednotky.

Poslední zodpovědnost této třídy je pohyb útočníka. Pro replikace pohybu příjde ze serveru seznam míst, kde se má jednotka v jakém čase nacházet. Mezi jednotlivými po sobě jdoucími dvojicemi se aktuální poloha určí podle poměru uplynulého času od prvního z nich a celkového času mezi body. V polovině času musí být jednotka na půl cesty mezi pozicemi.

`MissileReplayController` je třída zodpovědná za zobrazování střel, které vycházejí z věží a dopadají na útočníky. Při každém cyklu s aktuálním časem

zkontroluje informace o všech střelách, a když narazí na střelu, která je v tu dobu aktivní a není vytvořená, tak ji vytvoří. Dále všem aktivním střelám mění pozici stejným způsobem jako ji mění útočníci, s jediným rozdílem, že střela má pouze jednu počáteční pozici a jednu koncovou pozici, ne seznam pozic. Když střela přestane být aktivní, znovu ji smaže. Pro vytváření a mazání střel je použit návrhový vzor Object Pool, protože vytvoření herního objektu je v Unity složitější operace a vytvářet pokaždé nový pro každou střelu je extrémě neefektivní.

Object Pool Návrhový vzor Object Pool se používá v případech, kdy je drahé vytvářet objekty, a kdy víme, že objektů bude potřeba za život aplikace vytvořit velké množství ale zároveň v jednu chvíli jich bude potřeba pouze malé množství.

Třída Object Pool drží instance daných objektů, které nejsou momentálně využité a pokud někde je potřeba tento objekt dá k dispozici jeden z těchto nepoužitých. Poté, až se skončí s daným objektem práce, místo jeho zničení se vrátí zpátky do Object Poolu. Pokud je vyžádán objekt ve chvíli, kdy Object Pool nemá žádné nepoužité, tak se vytvoří nový. [13]

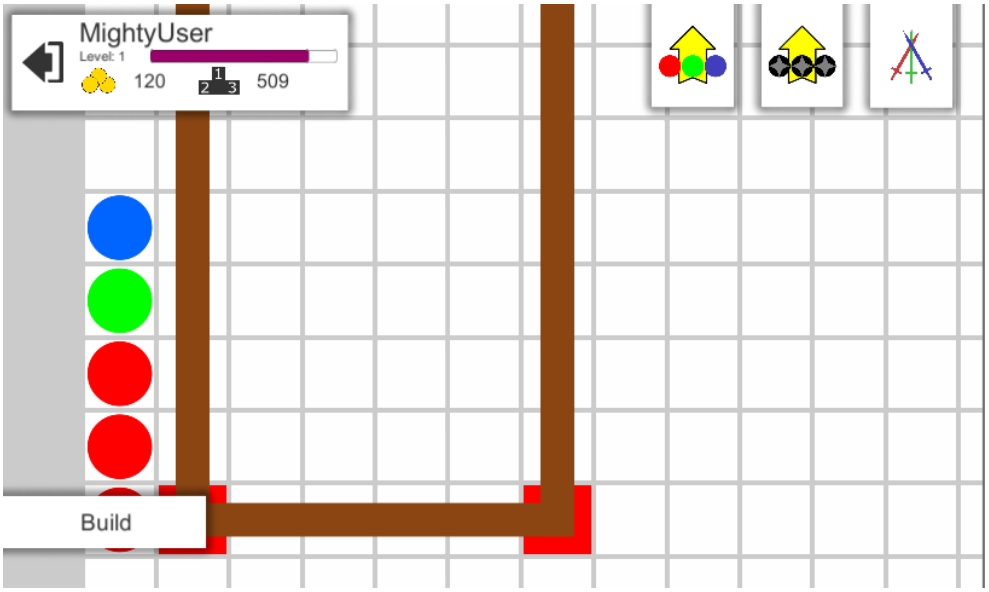
3.2.1.2 Uživatelské rozhraní

Použité komponenty Unity obsahuje velké množství připravených komponetů, z kterých lze rovnou vytvářet herní objekty. Hlavní komponenty použité pro vytvoření uživatelského rozhraní:

- Image - přidá objektu obrázek,
- Text - stylizovatelný text,
- Button - tlačítko na které lze navázat funkce,
- Animator - komponenta obstarávající animace,
- HorizontalLayoutGroup, VerticalLayoutGroup, GridLayoutGroup, LayoutElement - komponenty používané pro automatické pozicování dalších objektů, které jsou jejich potomci.

Většina prvků uživatelského rozhraní je postavena na herním objektu, který má komponentu Image. Tomuto základnímu prvku se v Unity říká Panel.

Třída UIController Tato třída má v sobě odkazy na všechny prvky uživatelského rozhraní a přepíná mezi nimi. Přepínání funguje aktivací příslušné animace, která prvek buď ukáže na obrazovku nebo z ní odstraní. V Unity není možné volat metody prvků uživatelského rozhraní z jiných vláken než z jejího hlavního vlákna, takže pro změny, vyvolané z modelu, je potřeba uložit informaci o změně a provést ji až v příštím průchodu hlavního vlákna.



Obrázek 3.4: Hlavní obrazovka hry

Přihlášení Přihlašovací obrazovka je tvořena jedním panelem s komponenty Text a Button. Tlačítko zahájí přihlášení přes Facebook, respektive pošle požadavek na přihlášení do modelu, který ho zpracuje a o průběhu dává vědět třídě `UIController`.

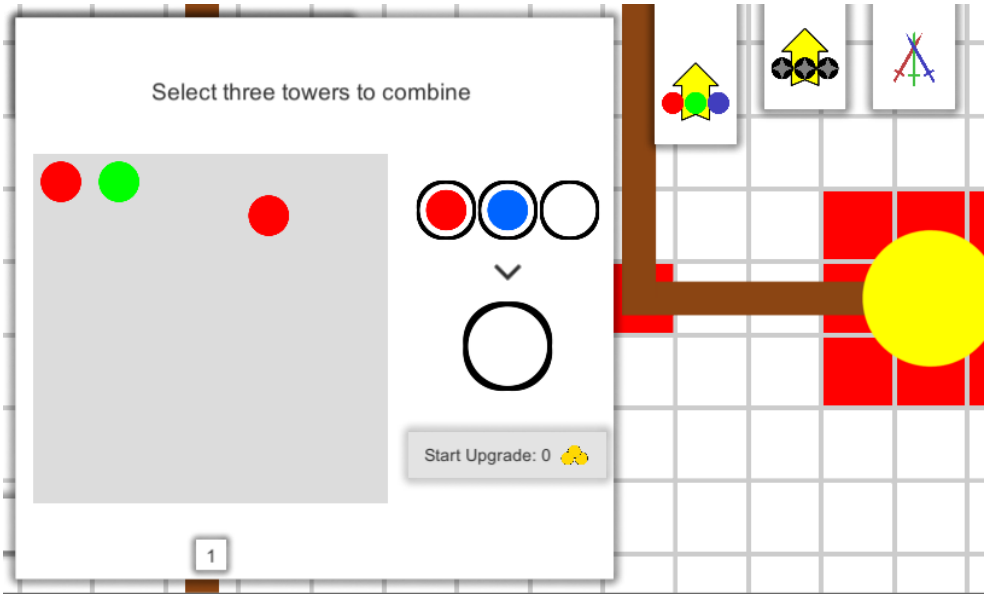
Hlavní obrazovka Z obrázku 3.4 je možné vidět, že hlavní pohled na hru má tři části uživatelského rozhraní.

První částí je informační panel s údaji o hráči, jeho úrovni, zlatě a hodnocení. Je vytvořený pomocí kombinace komponent Text, Image a Button. Komponenta Button je použita pro tlačítka na odhlášení. Panel zkušeností vedle textu o úrovni je vytvořen pomocí dvou obrázků, bílý je statický a fialový mění dynamicky velikost podle aktuálních zkušeností hráče.

Druhou část tvoří tlačítka v pravém horním rohu. První dvě otevírají vylepšování věží a útočníků a třetí zahajuje útok na ostatní hráče. Všechny tři jsou pouze panel s jedním tlačítkem. V jednu chvíli může být aktivní pouze jediné, při kliknutí na tlačítko se zavře obsah všech ostatních a až poté se zobrazí obsah aktivního tlačítka.

Poslední částí je přepínání mezi prohlížením a stavěním v levém dolním rohu. Je tvořené jedním panelem a tlačítkem, které má pouze Text.

Vylepšování věží Vzhled obrazovky pro vylepšování věží lze vidět na obrázku 3.5. Obrazovka se skládá z nápisu s instrukcí, tlačítka, které spouští vylepšení, míst pro vylepšované věže a ze seznamu věží. Pro přetahování věží



Obrázek 3.5: Obrazovka vylepšování věží

ze seznamu na místa pro vylepšení je implementována „drag and drop“ komponenta. Unity dává k dispizici interface `IDropHandler` a `IDragHandler`, které když komponenta implementuje, tak začne dostávat události o tažením myši (nebo prstu po dotekové obrazovce).

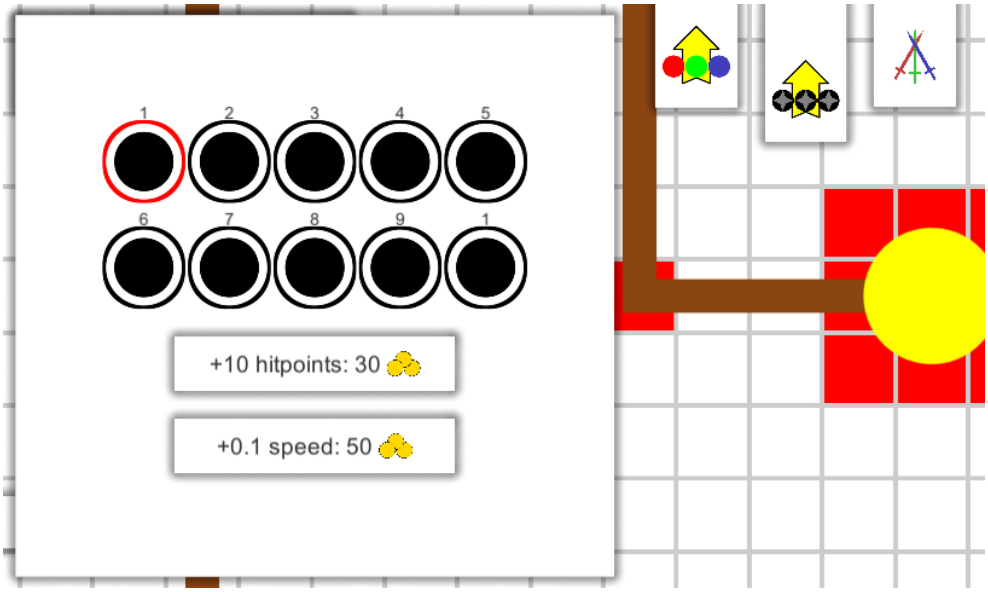
Vylepšování jednotek Obrazovka pro vylepšování jednotek, kterou je možné vidět na obrázku 3.6, využívá stejný mechanismus přetahování objektů jako obrazovka vylepšování věží, dále má dvě jednoduchá tlačítka a vybrání věže změní barvu jejího okraje.

Útočení Obrazovka pro útočení na ostatní hráče má stejnou strukturu jako hlavní obrazovka. Jediná vyjímka je výměna tlačítka na stavění za tlačítko pro přeskakování nepřátel.

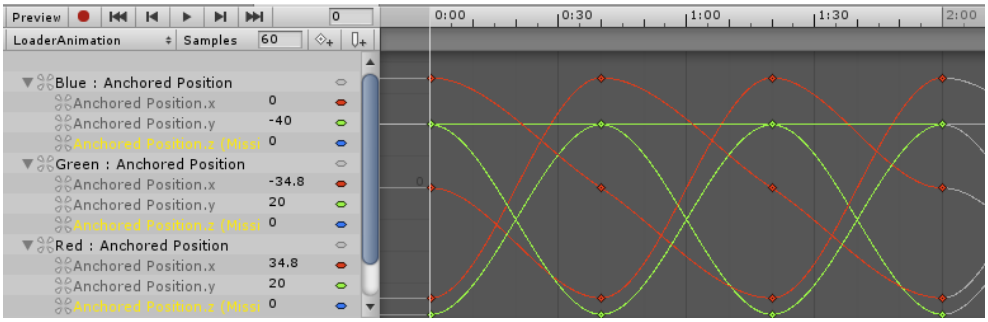
3.2.1.3 Animace

V uživatelském rozhraní je použito velké množství animací. Animované jsou všechny přechody mezi jednotlivými prvky rozhraní, ikona načítání a průběh vylepšování věží. S vyjímkou vylepšování věží jsou všechny animace vytvořené pomocí animačního systému herního enginu Unity.

Animační systém enginu Unity Vytváření animací v herním enginu Unity má dvě části: vytvoření jednotlivých animací a jejich spojení pomocí přechodů. Pro vytváření animací má engine jednoduchý editor. V něm je časová osa a do



Obrázek 3.6: Obrazovka vylepšování útočníků

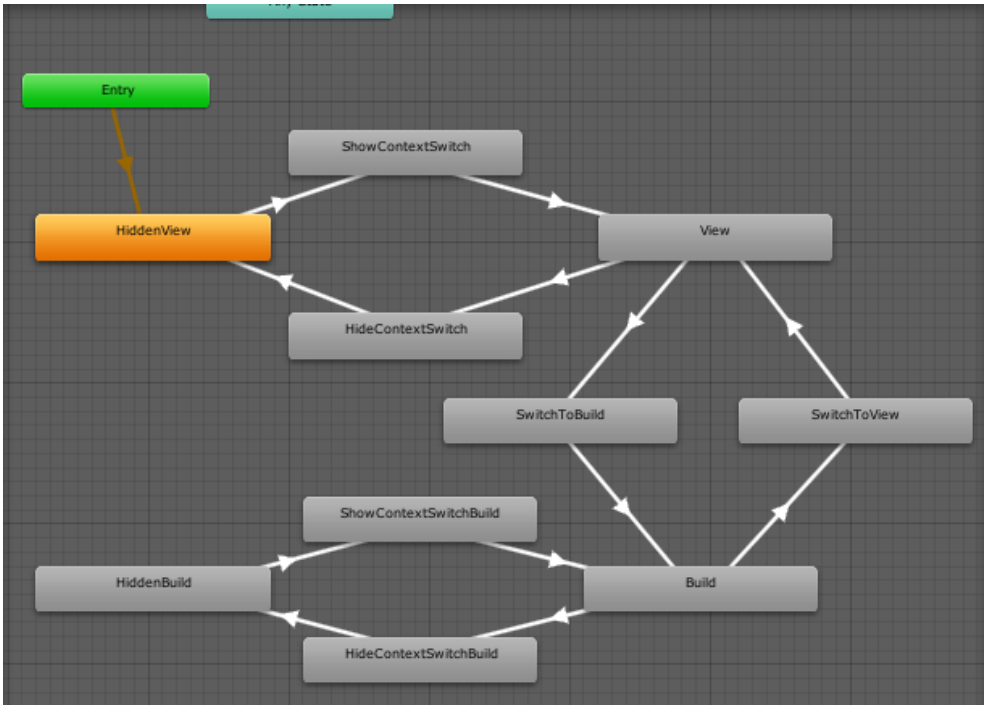


Obrázek 3.7: Editace animace načítání

ní lze přidávat klíčová místa, kde se určí hodnoty proměnných prvku který je animován. Mezi klíčovými místy pak editor hodnoty plynule mění. Na obrázku 3.7 je vidět příklad editování animace načítání. Na červených liniích je dobře vidět průběh, kde se postupně vyměňují místa tří barevných prvků, čímž se vytvoří rotující efekt.

Spojování jednotlivých animaci v celek probíhá v komponentě `Animator`. Na obrázku 3.8 lze vidět nastavení konkrétní komponenty pro animace tlačítka, které přepíná mezi stavěním a normálním pohledem na pevnost.

`Animator` je vlastně stavový diagram, kde jednotlivé stavy jsou animace, které se buď opakují dokud nenastane podmínka, nebo po jednom průběhu animace se spustí přechod do dalšího stavu. V panelu nalevo jsou vidět proměnné, které lze nastavit z kódu na hodnotu a tím podmínit přechod mezi



Obrázek 3.8: Komponenta pro provázání animací

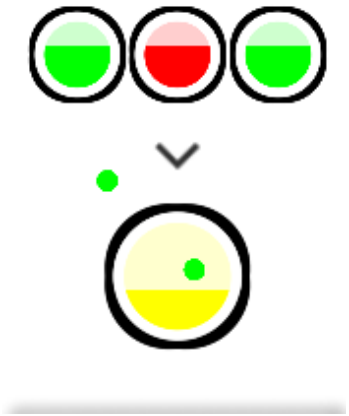
stavy.

Animace vylepšování věží Tato animace není vytvořena pomocí zmíněného animačního systému, protože její průběh není vždy stejný. Délka animace závisí na úrovni vylepšovaných věží a jednotlivé prvky se v animaci pohybují náhodně.

Animace se skládá ze tří částí:

- vylepšující se věže, které pomalu mizí,
- létající kuličky symbolizující proudění materiálu mezi věžemi,
- výsledná věž, která se pomalu objevuje.

Podobu animace lze vidět na obrázku 3.9. Mizící a objevující se věže jsou vytvořeny pomocí průhledné bíle vrstvy, která má stejný tvar a velikost jako věž, kterou maskuje. Její velikost se odvíjí od času vylepšení, který už uběhl, v poměru k celkovému času vylepšení. Létající kuličky jsou samostatné objekty, které mají zadaný počátek v jedné z věží a konec ve vylepšující se věži. V každém herním cyklu se pak pohybují tímto směrem a vytvářejí náhodné odchylky, aby pohyb nebyl úplně přímočarý, když dorazí do cíle, na náhodnou dobu se nastaví jako neviditelné a začínají od začátku.



Obrázek 3.9: Animace vylepšení věží

3.2.2 Model

Hlavní část implementace modelu je komunikace se serverem, která probíhá při každé akci, kterou hráč vykoná. Data jsou uložena v paměti a při každém zapnutí hry jsou znovu stažena ze serveru, aby hráč měl vždy aktuální informace.

Komunikace se serverem Základ pro posílání požadavků na server je třída `ServerRequest`. V ní je implementace volání jednotlivých lambda funkcí pomocí AWS SDK, která vypadá následovně:

```
RegionEndpoint endpoint =
                    Amazon.RegionEndpoint.EUCentral1;

if(credentials == null){
  try{
    credentials = new CognitoAWSCredentials (
                      "token",
                      endpoint);
  } catch(Exception e){
    GameLogicFacade.getInstance().onDisconnected();
  }
}

AmazonLambdaClient client = new AmazonLambdaClient(
                                credentials,
                                endpoint);
InvokeRequest request = new InvokeRequest();
request.FunctionName = functionName;
request.Payload = payload;
request.InvocationType = InvocationType.RequestResponse;
client.InvokeAsync(request,lambdaCallback);
```

Nejdříve se vytvoří přístupové údaje vytvořením nové instance třídy *CognitoAWSCredentials*. Ta už se stará o komunikaci se službou Amazon Cognito, která přidělí aplikaci práva pro přístup k Lambda funkcím. Za pomoci těchto údajů se vytvoří instance *AmazonLambdaClient* a na ní se spustí *InvokeRequest*, v kterém se určuje jaká Lambda funkce je volaná a nastaví se zde serializovaná data, která funkce bere jako vstup.

Pro serializaci dat do formátu JSON má Unity velice jednoduchou funkcionalitu. Stačí vytvořit datovou třídu označenou jako *Serializable* a poté zavolat funkce *FromJson* nebo *ToJson*. Následující kód ukazuje definici takové třídy a její serializaci a deserializaci.

```
[Serializable]
public class Exception
{
  public string errorMessage;
  public string errorType;
}

Exception ex = JsonUtility.FromJson<Exception>(
                                  sourceString);
String serialization = JsonUtility.ToJson(ex);
```

Kapitola 4

Testování

Důležitější části serverové implementace jsou pokryty testy, ale hlavní zaměření při testování hry bylo na uživatelské testy. Probíhalo ve dvou částech: uzavřená alfa verze a otevřená beta verze.

4.1 Alfa verze

Uzavřený alfa test byl prováděn se skupinou uživatelů z blízkého okolí, celkem hru v alfa verzi hrálo 6 uživatelů. Hra byla zpřístupněna pomocí distribuční platformy Google Play, která nabízí možnost jak otevřených, tak uzavřených testů. Hráči neměli konkrétní zadání, co mají ve hře provést, aby ze zpětné vazby šlo zjistit, jak bude hra působit na nové uživatele.

Zpětná vazba uživatelů Hlavní část zpětné vazby sestávala z chyb ve hře, které hráčům mohly znepříjemnit život, nebo přímo znemožnit hru hrát. Několik kritických chyb, které byly opraveny ještě v průběhu alfa verze:

- Pokud hráč na někoho útočí zrovna ve chvíli, kdy se dokončí vylepšování veží, změní se rozložení věží zpět na hráčovu pevnost a záznam útoku v tu chvíli přestává dávat smysl.
- Rychlé mačkání tlačítek pro otevření panelů vylepšování může způsobit, že se panely před bojem nezavřou a není vidět na herní plochu a není možné je v tu chvíli zavřít.
- Po dosažení maximální úrovně hráči nemohli útočit, protože je server v tu chvíli odpojil.

Další důležitá zpětná vazba, která vyšla z průběhu testování byl nedostatek informací o tom, jak se hra hraje. Všichni uživatelé měli potřebu se zeptat, jak hra funguje a co že všechno můžou dělat, a jak se to dělá.

V poslední řadě z průběhu alfa verze vzešlo, že je potřeba doplnit alespoň nějaká analytická data, aby šlo vyhodnotit, jak hráči hru používají, na čem se zaseknou, nebo jak často používají některé funkce.

4.2 Beta verze

Před vydáním beta verze, která má být otevřená pro jakékoliv uživatele bylo potřeba doplnit nedostatky objevené v alfa verzi.

Tutoriál První potřebný krok, který vzešel z otázek uživatelů alfa verze, je tutoriál, který hráče provede základními úkony ve hře. Tutoriál postupně uživateli ukazuje zvýrazněné akce, které má udělat společně s popisným textem. V jeho rámci uživatel projde následující scénáře:

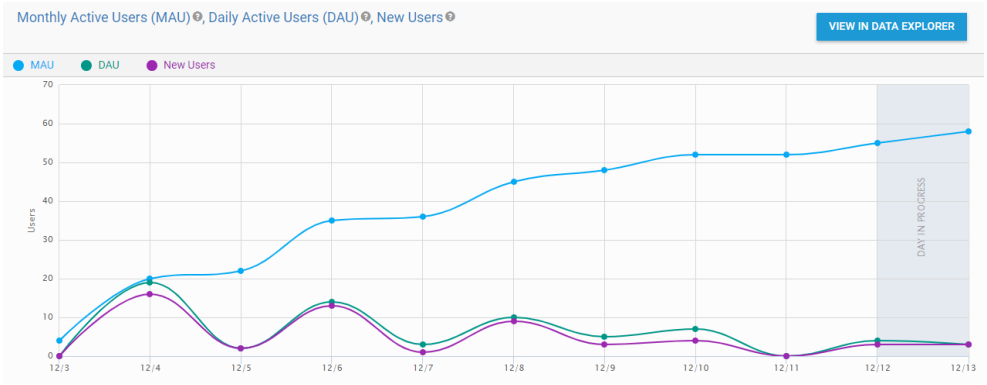
- útok na jiného hráče,
- vylepšení věže,
- vylepšení jednotky,
- změna rozestavení věží v pevnosti,
- otevření tabulky skóre všech hráčů.

Přidání odesílání analytických dat Pro kontrolu, jak hráči ve hře fungují bez konkrétní zpětné vazby, je do hry přidáno odesílání analytických dat. Pro tuto funkcionalitu je použita služba Unity Analytics, která je přímo dostupná z herního enginu Unity.

Služba sama automaticky posílá několik základních událostí, z kterých nejdůležitější jsou informace o času a frekvenci hraní jednotlivých uživatelů. Dále dává možnost posílat vlastní události, jejichž název a účel si může vývojář sám nadefinovat. Poslední důležitá funkce, která je pro analytiku v beta verzi použita, jsou takzvané *Funnels*, což jsou sekvence událostí speciálních událostí. Je na nich možné sledovat, jaké procento uživatelů sekvenci dokončilo a jakou dobu jim to trvalo. [14]

Speciální události byly vytvořeny pro následující akce:

- otevření vylepšování jednotek,
- výměna dvou různých jednotek,
- vylepšení životů a rychlosti jednotky,
- začátek vyhledávání nepřátel,
- vynechání nabídnutého nepřítele,



Obrázek 4.1: Počet hráčů podle Unity Analytics

- přeskočení probíhající animace útoku,
- získání veže se zkombinovanou barvou,
- otevření žebříčku skóre hráčů.

Pro funkci *Funnels* byly použity následující sekvence:

- Jednotlivé úkoly tutoriálu
- Získání vylepšení věže druhé, třetí a čtvrté úrovně

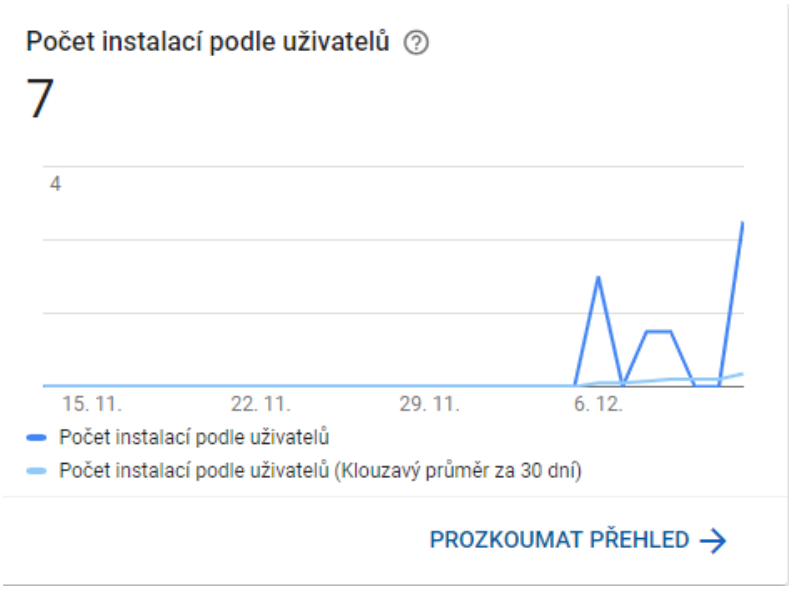
4.3 Vyhodnocení výsledků testování

Otevřená beta verze byla dostupná pro hráče 10 dní, během kterých byla sbírána data o jejich chování. Hra neoslovila příliš velký počet lidí, ale i přesto lze v přehledu Unity Analytics vidět jisté výsledky.

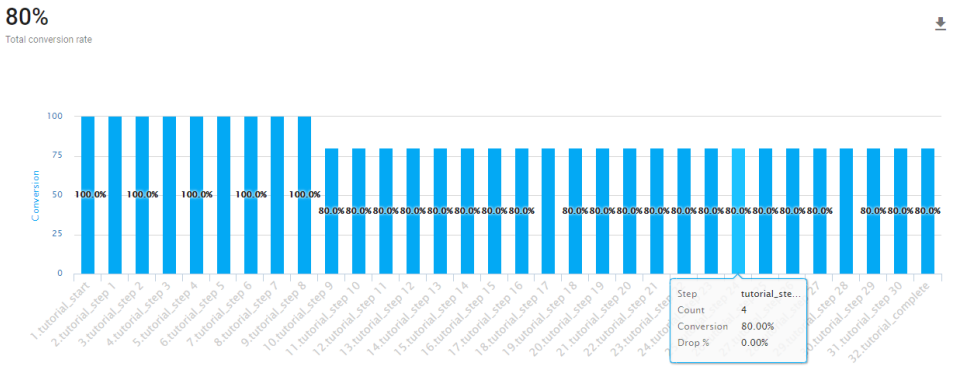
Počet hráčů Kontrolování počtu hráčů bohužel nefungovalo v Unity Analytics příliš dobře. Na obrázku 4.1 je na modré křivce vidět, že počet uživatelů za daný měsíc se blíží 60. To bohužel vůbec nesouhlasí s počtem instalací ze statistik Google Play na obrázku 4.2. Celkem tedy hrálo beta verzi 7 hráčů. Toto číslo i souhlasí s počtem hráčů, kteří byli zobrazeni na žebříčku ve hře.

Nepovedlo se dohledat proč přesně Unity ukazuje čísla, která nesouhlasí s reálnými uživately. Moje podezření je, že zaregistrovala jako uživatele i spuštění hry v Editoru, ve chvíli kdy bylo potřeba něco opravit.

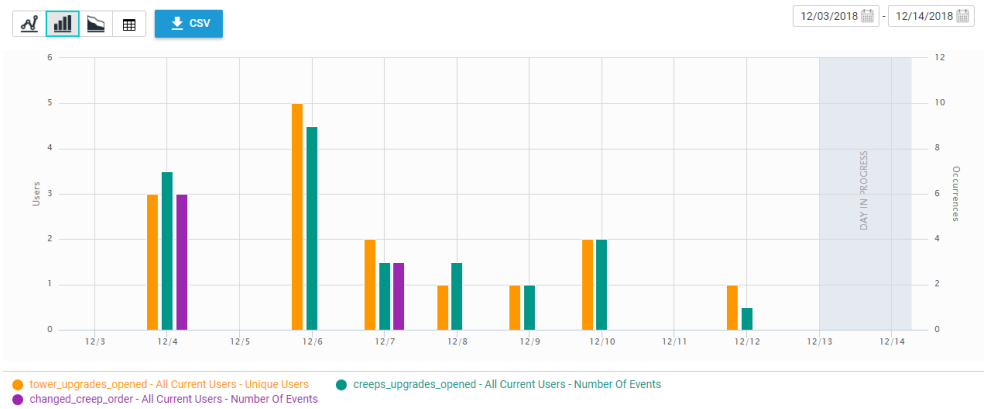
Průchod tutoriálem Na obrázku 4.3 lze vidět rozložení jaké procento hráčů, kteří tutoriál začali, ho i dokončilo. Celkem tutoriál dokončili čtyři uživatelé, jeden uživatel hru vzdal v první třetině a dva další uživatelé hru pouze nainstalovali a zadali si přezdívku.



Obrázek 4.2: Počet hráčů podle Google Play]



Obrázek 4.3: Průchod hráčů Tutoriálem



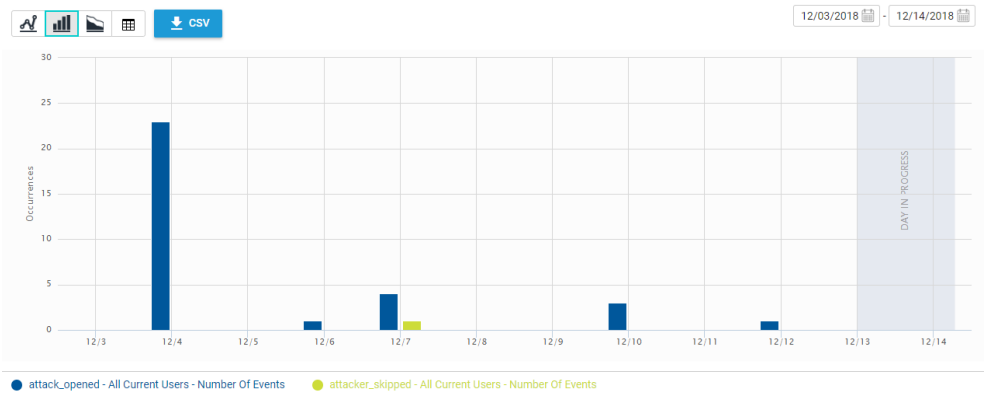
Obrázek 4.4: Statistiky vylepšování jednotek

Předpokládám, že důvod proč uživatelé odešli po zadání přezdívky je grafická stránka hry. V tu chvíli je totiž poprvé možné vidět celou podobu hry. Uživatele, který skončil na osmém kroku tutoriálu, pravděpodobně neoslovila tématika stavění vlastní pevnosti, protože tento krok tutoriálu popisuje změnu pozice věží.

Vyhodnocení událostí Počet hráčů byl sice nízký, ale přesto se povedlo nasbírat zajímavá uživatelská data a potvrdit nebo vyvrátit některé předpoklady o průchodu hrou.

První nejasnost byla, jak moc se hráči budou snažit vylepšovat si vlastní jednotky a měnit jejich pořadí. Na obrázku 4.4 lze vidět porovnání otevírání obrazovek pro vylepšení věží a jednotek. Vylepšení věží je zobrazeno oranžovou barvou, vylepšení jednotek zelenou. Počet otevření je skoro na stejné úrovni, ale při vylepšení jednotek hráči málokdy něco změnili. Fialovou barvou jsou zobrazeny změny, které hráči udělali a je vidět, že jsou přítomné pouze ve dvou dnech testování. Bylo by tedy potřeba více hráčům dát najevo, že je důležité vylepšovat svoje jednotky.

Druhá důležitá informace, kterou události poskytly, je využití tlačítka pro přeskočení nabízeného nepřítele při útoku. Na obrázku 4.5 je vidět, že přeskočení nepřítele bylo použito ze všech útoků pouze jednou. Možné řešení bude snížení ceny za přeskočení, nebo ujasnění použití tlačítka v tutoriálu.



Obrázek 4.5: Využití tlačítka přeskočení nepřítele

Závěr

Cílem práce bylo vyvinout hru pro více hráčů a dostatečně ji otestovat, aby byla připravená pro vydání. Cíl byl splněn, z technického hlediska hra funguje dobře a nebyl by problém ji vydat. K tomu, aby hra byla i úspěšná, ovšem ještě chybí velká část práce.

Otevřené testování nebylo v zadání práce, ale rozhodl jsem se ho uskutečnit i přes očekávání, že grafické zpracování nebude mít příliš velký úspěch. To se nakonec ukázalo jako pravdivý předpoklad a otevřené testování mělo kvůli tomu malou účast. Přesto však tato fáze přinesla užitečná data a po sehnání lepších grafických podkladů by hra mohla mít úspěch.

Použití cloudového prostředí pro vývoj serveru dopadlo dobře, za celou dobu vývoje nebyl se serverem jediný problém a využití serverless prostředí Lambda vývoj velice urychlilo.

Literatura

[1] Choosing the Right Name for Your Game. *Evatotuts+ [online]*, leden 2013, [cit. 2018-2-16]. Dostupné z: https://gamedevelopment.tutsplus.com/articles/choosing-the-right-name-for-your-game--gamedev-3895

[2] Cloud Services. *Techopedia [online]*, [cit. 2018-2-17]. Dostupné z: https://www.techopedia.com/definition/29017/cloud-services

[3] IaaS, PaaS, SaaS - modely cloudových služeb. *IBM [online]*, [cit. 2018-2-17]. Dostupné z: https://www.ibm.com/cloud-computing/cz-cs/learn-more/iaas-paas-saas/

[4] Nasr, S.: Mobile Gaming and Scalability in the Cloud. *ComputeNext [online]*, srpen 2014, [cit. 2018-2-17]. Dostupné z: https://www.computenext.com/blog/mobile-gaming-and-scalability-in-the-cloud/

[5] Marzouk, Z.: Why Sega's latest games required a move to the cloud. *ITPRO [online]*, listopad 2017, [cit. 2018-2-17]. Dostupné z: http://www.itpro.co.uk/databases/29909/why-segas-latest-games-required-a-move-to-the-cloud

[6] Windels, J.: Pokemon Go Figure: A data analysis of the most popular game of all time. *Wandera [online]*, únor 2017, [cit. 2018-2-17]. Dostupné z: https://www.wandera.com/blog/pokemon-go-data-analysis-popular-game/

[7] Rojas, A.: A Brief History of AWS. *Media temple [online]*, srpen 2017, [cit. 2018-2-23]. Dostupné z: http://mediatemple.net/blog/news/brief-history-aws/

[8] Meier, R.: An Annotated History of Google Cloud Platform. *Medium [online]*, Únor 2017, [cit. 2018-2-23]. Dostupné z:

https://medium.com/@retomeier/an-annotated-history-of-googles-cloud-platform-90b90f948920

[9] Callaway, R.: A brief history of cloud-based integration in Microsoft Azure. *QuickLearn [online]*, Listopad 2015, [cit. 2018-3-3]. Dostupné z: http://www.quicklearn.com/blog/2015/11/02/historyofthecloud/

[10] Nystrom, B.: *Game Programming Patterns*. Genever Benning, první vydání, ISBN 978-0-9905829-0-8.

[11] Wenderlich, R.: Introduction to A* Pathfinding. *raywenderlich.com [online]*, Září 2011, [cit. 2018-6-8]. Dostupné z: https://www.raywenderlich.com/4946/introduction-to-a-pathfinding

[12] Yegulalp, S.: What is Cython? Python at the speed of C. *InfoWorld [online]*, Únor 2018, [cit. 2018-6-9]. Dostupné z: https://www.infoworld.com/article/3250299/python/what-is-cython-python-at-the-speed-of-c.html

[13] Shvets, A.: Object Pool Design Pattern. *Source making [online]*, [cit. 2018-6-18]. Dostupné z: https://sourcemaking.com/design_patterns/object_pool

[14] Technologies, U.: Analytics events. *Unity documentation [online]*, [cit. 2018-11-24]. Dostupné z: https://docs.unity3d.com/Manual/UnityAnalyticsEvents.html

Příloha A

Seznam použitých zkratek

SaaS Software as a service

PaaS Platform as a service

IaaS Infrastructure as a service

AWS Amazon web services

Příloha **B**

Obsah přiloženého DVD

- `readme.txt` ..... stručný popis obsahu DVD
- `thesis` ..... text práce a jeho zdrojové soubory
  - `sources` ..... zdrojové soubory práce
  - `assignment.pdf` ..... zadání práce ve formátu PDF
  - `thesis.pdf` ..... text práce ve formátu PDF
- `implementation` ..... zdrojové kódy implementace
  - `server` ..... zdrojové kódy serverové části
  - `client` ..... zdrojové kódy clientské části