



**MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova**

BAKALÁŘSKÁ PRÁCE

Jméno Příjmení

Název práce

Název katedry nebo ústavu

Vedoucí bakalářské práce: Vedoucí práce

Studijní program: studijní program

Studijní obor: studijní obor

Praha ROK

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V dne

Podpis autora

Poděkování.

Název práce: Název práce

Autor: Jméno Příjmení

Katedra: Název katedry nebo ústavu

Vedoucí bakalářské práce: Vedoucí práce, katedra

Abstrakt: Abstrakt.

Klíčová slova: klíčová slova

Title: Name of thesis

Author: Jméno Příjmení

Department: Name of the department

Supervisor: Vedoucí práce, department

Abstract: Abstract.

Keywords: key words

Obsah

1 Úvod	2
1.1 RTS hry blíže	3
1.1.1 Mapy	5
1.1.2 Jednotky	6
1.1.3 Budovy	9
1.1.4 Suroviny	12
1.1.5 Vývoj technologií	13
1.2 Uživatelé	15
1.2.1 Tvůrci her	15
1.2.2 Hráči her	17
1.3 Ukázková hra	17
1.4 Cíle práce	18
2 Analýza	20
2.1 Herní engine	20
2.2 Rozdíly systémů	20
2.2.1 Zobrazení a ovládání	20
2.2.2 JIT vs. AOT	21
2.2.3 Souborové systémy	22
2.2.4 Shrnutí	22
2.3 Formát a načítání dat	23
2.3.1 Struktura balíčku	23
2.3.2 Data entit	24
2.3.3 Formáty assetů	24
2.3.4 Formát uložených úrovní	27
2.4 Reprezentace stavu hry	28
2.4.1 Mapa	28
2.5 Poskytované služby	28
2.5.1 Pathfinding	28
2.5.2 Projektily	28
3 Ukázková hra	29
4 Programátorská dokumentace	30
5 Uživatelská dokumentace	31
Závěr	32
Seznam použité literatury	33
A Přílohy	35
A.1 První příloha	35

1. Úvod

Strategické hry jsou žánrem, ve kterém hráči využívají svých mentálních schopností, především taktického a strategického myšlení, pro porážku jednoho či více nepřátel. Ve většině případů se strategické hry zabývají tématem války. Takto popisuje strategické Ernest Adams ve své knize *Fundamentals of Game Design*[3, str. 419]

Žánr strategických her obsahuje mnoho poddruhů s velice rozdílnými nároky jak na hráče, tak na vývojové prostředí a na vývojáře samotného. Prvním kritériem pro rozdělení strategických her je, zda se hra odehrává jako posloupnost diskrétních tahů, nebo zda se hra odehrává v přímé závislosti na ubíhajícím reálném čase. Druhým kritériem je relativní četnost a důležitost strategických rozhodnutí vůči taktickým rozhodnutím. Tato dvě kritéria použil Mark H. Walker[16] pro jejich rozdělení na tyto poddruhy:

- *Real-time strategy* (RTS) - reálný čas, strategická rozhodnutí
- *Real-time tactics* (RTT) - reálný čas, taktická rozhodnutí
- *Turn-based strategy* (TBS) - diskrétní tahy, strategická rozhodnutí
- *Turn-based tactics* (TBT) - diskrétní tahy, taktická rozhodnutí

Cílem této práce je vytvořit platformu umožňující tvorbu Real-time strategy (RTS)¹ her. V následujících částech popíšeme rozdíly mezi poddruhy strategických her a vymezíme podmnožinu, jejíž vývoj bude naše platforma podporovat.

Real-time strategy

RTS, v překladu strategické hry probíhající v reálném čase, jsou poddruhem strategických her ve kterém se změny stavu odehrávají v přímé závislosti na změně času v reálném světě. Reakce v reálném čase jsou náročnější jak pro hráče, který je často nucen použít suboptimální strategii, tak pro hru samotnou, která musí provádět výpočet dalšího stavu v omezeném čase. Stejně tak umělá inteligence, jakožto součást hry, musí reagovat na aktivity zbylých hráčů s omezeným časem, což limituje množství dat a složitost výpočtu, které může umělá inteligence použít. Z tohoto důvodu je vývoj RTS her složitým procesem, spojujícím mnoho oborů, který se pokusíme zjednodušit vytvořením naší platformy.

Zbylé poddruhy strategických her neplánujeme v naší platformě explicitně podporovat, ale nijak nevylučujeme, že bude možné do jisté míry využít naší platformu i pro tvorbu těchto poddruhů strategických her. Zároveň je pro tvorbu RTS her výhodné znát příbuzné žánry, z kterých je možno se inspirovat a přebírat některé z jejich mechanik. Z tohoto důvodu ve zkratce popíšeme i zbylé poddruhy.

¹Název "real-time strategy", poprvé použitý při propagaci hry Dune II[2], je připisován prezidentu a spoluzakladateli Westwood Studios[?], Brettu Sperrymu. Toto studio následně využilo zkušenosti získané při tvorbě Dune II pro vývoj jedné z nejznámějších sérií RTS her, Command & Conquer[?].

Real-time tactics

Prvním příbuzným žánrem jsou *Real-time tactics* (RTT) hry, někdy také nazývány fixed-unit real-time hry [15], neboli hry s pevným počtem jednotek probíhající v reálném čase. Hlavním rozdílem, odlišujícím RTT od RTS, je omezení strategických rozhodnutí a větším důrazem na taktická rozhodnutí a micromanagement jednotlivých jednotek. RTT hry nedovolují hráči tvorbu nových jednotek, stavbu budov či produkci surovin, hráč je tedy nucen s jednotkami, které má na začátku souboje, vyhrát celý souboj. Jedním z příkladů čistě RTT her je série Blitzkrieg[?]. Hráč začíná každou misi s jednotkami, které si vybral před misí. Tyto jednotky jsou jediné, které bude moci v průběhu mise využít, což nutí hráče použít svých taktických schopností a maximalizovat účinnost těchto jednotek.

Turn-based strategy

Turn-based strategy jsou strategické hry, ve kterých změny stavu probíhají v diskrétních tazích. Doba mezi tahy často není nijak omezena, což hráči umožňuje vymyslet optimální strategii. Oproti RTS tyto hry často omezují taktickou část problémů a umožňují hráči soustředit se výhradně na strategickou část hry, tedy plánování budov, produkci surovin, vývoj technologií a celkovou strategii pro jeho ekonomiku a armádu. Příkladem tohoto žánru je série her Civilisation[5]. Naše platforma nebude explicitně podporovat tahy, myslíme si však, že bude možné toto rozdelení do tahů vytvořit použitím platformou poskytovaných prostředků.

Turn-based tactics

Turn-based tactics umožňují hráči přímé ovládání několika málo jednotek, které v každém tahu mohou provést jeden či více úkonů. Těmito úkony mohou být střelba na nepřátelskou jednotku, vyléčení přátelské jednotky, pohyb po mapě či například zničení terénu. Ukázkovým příkladem je série her X-COM[?]. Ve hře X-COM 2 hráč vlastní až malé desítky jednotek, z kterých vybírá malou skupinu a vysílá ji na jednotlivé mise. Při misi má každá jednotka v každém tahu možnost vykonat dvě akce. Akce může být přesun o omezenou vzdálenost, použití schopnosti či útok na nepřítele. Pro normální jednotky útok ukončuje tah dané jednotky a hráč může provést tah následující.

Shrnutí

Jak můžeme vidět, žánr strategických her zahrnuje hry s velmi rozmanitými vlastnostmi a požadavky. Z tohoto důvodu se naše práce zaměří na podporu vývoje her jednoho konkrétního poddruhu, a to RTS. Přestože naše platforma bude cílena na tvorbu tohoto poddruhu, nevylučujeme, že bude možné využít ji i pro tvorbu her spadajících do jednoho ze zbylých poddruhů strategických her.

1.1 RTS hry blíže

Jak bylo řečeno výše, naše platforma bude navržena pro podporu vývoje RTS her. RTS hry jsou ovšem stále příliš velká množina s příliš rozmanitými mechanika-

kami na to, aby jedna platforma dokázala podporovat všechny možné RTS hry. Proto zde dále omezíme námi podporovanou podmnožinu RTS her.

Již od svého vzniku na konci osmdesátých let a začátku devadesátých let minulého století obsahovaly RTS hry několik konceptů, které lze nalézt v drtivé většině her tohoto žánru i dnes.

Těmito koncepty jsou:

- Výroba a ovládání jednotek s cílem ovládnutí části mapy a zničení nepřátelských jednotek a budov
- Stavba budov pro umožnění stavby nových druhů budov, jednotek či získání surovin
- Získávání surovin pro stavbu jednotek a budov
- Výzkum nových technologií

Tyto koncepty, jako základní kámen RTS her, se bude naše platforma snažit podporovat a zjednodušit tvůrcům her jejich implementaci.

Hlavní inspirací pro tvorbu platformy, a tím i pro typ her, které bude platforma nejjednodušejí podporovat, byla hra Stronghold Crusader. Na této a dalších hrách ukážeme v následujících částech blíže základní principy RTS her a námi podporované implementace těchto principů.

Strategie vs. Taktika

Jedním z hlavních problémů RTS her je pečlivé vyvážení kombinace strategie/macromanagementu a taktiky/micromanagementu. Tyto pojmy bývají často špatně chápány a někdy zaměňovány, pokusíme se je proto konkrétně definovat. Strategií míníme rozhodnutí týkající se globálního průběhu hry. Taktika naopak zahrnuje konkrétní pozice konkrétních jednotek, jejich pohyb po mapě a spolupráci v jedné bitvě.

Mezi strategická rozhodnutí v RTS hrách patří kupříkladu které budovy hráč postaví, v jakém pořadí dané budovy postaví, které suroviny bude produkovat, které suroviny vyneschá, které jednotky bude rekrutovat a které vyneschá. Tato 3 rozhodnutí jsou úzce propojena, protože výběr surovin určuje budovy, které bude hráč schopen postavit a typy jednotek, které bude moci zrekrutovat. Stejně tak výběr budov určuje suroviny, které hráč může produkovat a typy jednotek, které může zrekrutovat.

Taktika/micromanagement je v RTS hrách reprezentován ovládáním jednotek, jejich přesné pozice, pohybu, směru útoku, používání schopností atd. Micromanagement lze ale vidět i v ekonomické stránce hry, kdy hráč

Nejlepším příkladem pro rozlišení pojmu strategie a taktiky je série her Total War[?], které kombinuje mód tahové strategie s módem real-time tactics (RTT). V jedné části hry hráč přebírá kontrolu nad celým svým národem, rozhoduje, které budovy budou ve kterých městech postaveny, které jednotky budou rekrutovány a kde budou které armády umístěny. V druhé části, při souboji nepřátelských armád, hráč přebírá kontrolu nad konkrétní armádou a ovládá jednotlivé bataliony, jejich umístění, pohyb a útoky v reálném čase.

Naše platforma nemá za cíl nijak omezovat možnosti vytvářených her v rámci jejich zaměření na taktiku či strategii. Toto rozhodnutí chceme nechat čistě na uživateli naší platformy.

1.1.1 Mapy

Reprezentace herní mapy je jedním z hlavních rozhodnutí při tvorbě RTS her. Hlavní funkcí herní mapy je reprezentovat terén pro pohyb jednotek a stavbu budov. Tato funkce může zahrnovat neprostupné části mapy, několik různých druhů terénu prostupné různým druhům jednotek či části mapy ovlivňující rychlosť pohybu jednotek. Při stavbě budov je pak často omezen typ terénu, na kterém je hráč schopen danou budovu postavit. Vzhledem k uzavřenosti většiny RTS her je složité zjistit, jak je v každé z nich herní svět reprezentován, z pozorovatelného vnějšího chování lze ale odvodit několik základních druhů reprezentací.

Nejstarší a nejjednodušší reprezentací je rozdělení mapy na stejně velké dlaždice. Tyto dlaždice mohou být různých tvarů, nejčastěji jsou však čtvercové či hexagonální. Příkladem takového reprezentace je právě hra Stronghold Crusader[6], podle které chceme naši platformu modelovat. Jak je vidět na obrázku 1.1, herní mapa je viditelně rozdělena na stejně velké čtvercové dlaždice. Dále můžeme vidět, že je kamera orientována diagonálně vůči dlaždicím. Účelem této orientace je skrytí toho, že je mapa složena z dlaždic. Každá dlaždice je nějakého typu, který určuje její vzhled, což můžeme vidět v části zvýrazněné červenou barvou. Vidíme zde řady pěti dlaždic stejného typu, oddělené vždy jednou dlaždicí pouštěního typu. Můžeme si všimnout, že i dlaždice stejného typu mohou mít několik různých vzhledů. Typ dlaždice je dále využíván jako omezení při stavbě budov, kde například kamenolom lze postavit pouze na dostatečném počtu dlaždic typu kámen. Toto můžeme vidět v části označené modrou barvou, kde se hráč pokouší umístit budovu, která ovšem nemůže být postavena na dlaždicích typu mokřadu. Dlaždice tohoto typu jsou proto zvýrazněny červenou barvou v rámci půdorysu budovy. Dále můžeme v horní části ukázky vidět dlaždice s rozdílnou výškou od okolního terénu, tvořící nedostupnou oblast mapy. Dále můžeme vidět pro jednotky neprostupný typ dlaždic obsahujících vodu. Uprostřed obrázku poté vidíme velkou oblast kamení a železa, kde oblast kamení ukazuje texturu kvalitně skrývající složení mapy z dlaždic, naopak na oblasti železa jasně vidíme hranice jednotlivých dlaždic, především ve směru z levého dolního rohu do pravého horního rohu.



Obrázek 1.1: Ukázka dlaždic ve hře Stronghold Crusader

Na každé dlaždici může být postavena až na výjimky nejvýše jedna budova a stát nejvýše jedna jednotka. Při pohybu jednotek není tato vlastnost dodržována, lze tedy jednotky přesouvat přes dlaždice na kterých již jiná jednotka stojí.

Naše platforma bude podporovat rozšířenější verzi tohoto druhu mapy, ve které nebudeme vynucovat limity na počty jednotek a budov na jedné dlaždici. Tato omezení budou přenechána pro implementaci tvůrcem her využívajících naši platformu a bude vytvořeno rozhraní pro co nejjednodušší implementaci těchto limitů.

Naše platforma bude podporovat mapu s těmito vlastnostmi:

Další
druhy
(5)

M1: Terén rozdelený na čtvercové dlaždice.

M2: Dlaždice s různou výškou.

M2: Dlaždice různých typů.

M3: Možnost dotazovat se na jednotky nacházející se na dlaždici.

M4: Možnost dotazovat se na budovy postavené na dlaždici.

1.1.2 Jednotky

Jednotky jsou základním nástrojem hráče pro boj s nepřítelem. Pohybem po mapě, poškozováním ostatních jednotek a ničením budov jednotky umožňují hráči vést souboj s protivníkem, získat strategickou výhodu a následně vyhrát hru.

Pohyb

Hlavním odlišujícím prvkem jednotek od budov je jejich schopnost pohybu. Pohyb jednotky je určen jejím typem, kde každý typ jednotek může procházet jinými typy terénu, pohybovat se nad terénem, na vodě či pod vodou. Naše platforma bude podporovat pohyb jednotek kdekoli nad terénem, navíc bude tvůrcům poskytnuta komponenta umožňující chůzi po terénu, neboť je to nejčastější způsob pohybu.

Pohyb jednotek je nejčastěji řízen hráčem, ať už na úrovni příkazů jednotlivým jednotkám, tak na úrovni slučování jednotek do skupin a ovládání těchto skupin. Naše platforma bude umožňovat jak ovládání jednotlivých jednotek, tak celých skupin. Dále umožníme vývojáři přidat složitější ovládání, například slučování do permanentních formací a následné ovládání těchto formací.

V některých hrách existují také jednotky, které se mohou své schopnosti pohybu vzdát a stát se budovu, buď dočasně, nebo trvale. Příkladem takového jednotky/budovy mohou být budovy Nočních elfů ze hry Warcraft 3 [4]. Tyto entity jsou stavěny jako budovy, tedy jsou umístěny do světa a postaveny jednotkou, stejně jako u všech ostatních ras. Následně je ale hráči umožněno, pomocí speciální ability těchto budov, změnit je dočasně na jednotky a pohybovat s nimi či je dokonce použít pro boj. V tomto módu ale ztrácí všechny funkce budov, tedy není možné je použít pro sběr surovin či produkci jednotek. Následně je možné znovu je zakořenit, čímž získávají zpět své funkce budovy. Naše platforma by měla takové jednotky/budovy také podporovat.

Umělá inteligence

Ve velké části RTS her jsou jednotky schopny do určité míry autonomního rozhodování bez zásahu hráče, od střelby na cíl, který se ocitne v jejich dostřelu, po vyhledání krytu, pokud jsou pod palbou.

Jako příklad jednoduché umělé inteligence jednotek můžeme vzít hry Starcraft[?] a Warcraft[4] od společnosti Blizzard[?]. Zde se jednotky chovají velice předvídatelně, splňují přesně hráčovi rozkazy a nedělají nic navíc, což James Lantz[8] označuje jako jeden z faktorů, které umožnili hře Starcraft II vytvořit jednu z prvních masivních e-sport scén na světě.

Dobrým příkladem jednotek s vysokou autonomií je série Company of Heroes[13], kde jednotky automaticky vyhledávají krytí, rozutečou se, pokud jsou pod palbou dělostřelectva, a v případě příliš velkých ztrát utečou z boje. Příklad tohoto chování můžeme vidět na obrázku 1.2. V levé části vidíme skupinu jednotek, útočících na nepřátelskou jednotku mimo obraz ve směru červené šipky. Na prostřední části můžeme vidět, že po zničení nepřátele se jednotka začala přesouvat ve směru modré šipky, nejspíše pro získání lepšího krytí a lepší palebné pozice vůči zbývajícím nepřátelským jednotkám. Tento přesun je vykonán bez zásahu hráče, jak můžeme vidět díky modré ikoně nad jednotkou, která značí, že jednotka není zvolena hráčem a není jím tedy ovládána. Oproti tomu v pravé části můžeme vidět, že jednotka hráčem označena je díky bílé barvě ikony nad jednotkou. Vidíme, že hráč vydal rozkaz pro přesun zpět, zvýrazněný zeleně. Tato reakce byla vynucena umělou inteligencí jednotky, která se sama od sebe rozeběhla ve směru nepřátelských linií a donutila hráče reagovat. Jak vidíme, autonomie má svou cenu, a to v nepředvídatelnosti chování jednotek. Při jednoduché umělé inteligenci jednotek je hráč schopen předvídat jejich chování a využít ho pro svůj prospěch. Naopak při složité umělé inteligenci, jako právě v případě Company of Heroes, je hráč často nucen provést více pokusů při vydávání rozkazu, či vydávat rozkazy pro negaci automatického chování jednotky, protože není schopen toto chování jednoduše odhadnout. Tato skutečnost činí hry často realističejší, protože simuluje chování reálných vojáků, kteří rozkaz interpretují a implementují podle svého, není ale vhodná pro souboje více hráčů, a už vůbec né více hráčů na profesionální úrovni.



Obrázek 1.2: Ukázka autonomního chování jednotek ve hře Company of Heroes[13]

Platforma bude umožňovat vykonat libovolný kód v rámci každého výpočtu stavu každé jednotky, bude tedy pouze na vývojáři, zda se budou jednotky chovat jednoduše a předvídatelně, nebo zda budou vykonávat složité, avšak nepředvídatelné úkony bez hráčova vědomí. Pro ulehčení vývoje bude platforma poskytovat předpřipravené komponenty, umožňující základní úkony jako střelbu na cíl, pohyb po mapě a útok na blízko.

Produkce

Jednou z vlastností definujících RTS hry je možnost produkce nových jednotek. Existuje několik systémů produkce jednotek, úzce svázaných se systémem surovin v dané hře. (viz. 1.1.4) Od kontinuální produkce, kde hráč zvolí produkovány jednotky a suroviny jsou spotřebovány v průběhu produkce, po diskrétní produkci, kde hráč musí vlastnit všechny suroviny potřebné pro výrobu dané jednotky při začátku produkce a všechny suroviny jsou odečteny v jeden okamžik. Kontinuální systém umožňuje hráči naplánovat produkci armády v předstihu, i když v daném okamžiku nevlastní dostatečné suroviny. Naopak při diskrétní produkci je hráč nucen čekat do chvíle, kdy má všechny suroviny, a až poté může začít s produkcí. Naše platforma se pokusí podporovat oba systémy. Bude záležet pouze na tvůrci hry, jak se k surovinám a produkci jednotek zachová a který z těchto systémů bude implementovat.

Počet jednotek je často limitován, jak pro účely vyvážení hry, tak pro omezení zátěže hardwaru. Z hlediska vyvážení síly jednotek umožňuje limit na počet jednotek předejít tzv. "Zergu", kdy hráč vytvoří obrovské množství levných jednotek, které následně převálcují jakýkoli odpór. Z hlediska hardwarové náročnosti je účel limitu vcelku zřejmý, protože každá jednotka zabírá určité množství paměti a výpočetního výkonu. Stronghold Crusader omezuje počet jednotek na 1000 pro každého hráče. Toto omezení se jeví především jako limit na hardwarovou náročnost hry. Naše platforma žádné explicitní limity nestanovuje, avšak v uživatelské dokumentaci pro vývojáře budeme silně doporučovat stanovení limitů na počet budov, jednotek a projektů. Za tímto účelem umožníme vývojáři při vytvoření každé jednotky, budovy či projektu učinit rozhodnutí, zda je vytvoření možné a případně toto vytváření zrušit.

Hráč často začíná s malým počtem jednotek, jejichž účelem je zamezit tzv. "Rush" strategii, ve které je cílem vytvořit co nejrychleji co možná nejvíce levných jednotek a zničit nepřitele ještě před tím, než je schopen začít produkovat své jednotky. Ve hře Stronghold Crusader hráč začíná každou hru s několika lučišníky a kopiníky, jejichž počet je určen v nastavení před začátkem hry. Toto bude v naší platformě umožněno přidáváním jednotek v rámci editace mapy, případně bude tvůrce hry schopen umožnit hráči určit počty jednotek před začátkem hry pomocí grafických elementů v uživatelském rozhraní a následně při začátku hry vytvořit požadované množství jednotek. Tyto jednotky bude poté hráč vlastnit již na počátku hry.

Boj

V drtivé většině RTS her mají jednotky tzv. "hit pointy", zkráceně *HP*, které určují počet zásahů, které může jednotka obdržet než bude zabita. S každým zásahem jsou poté tyto *HP* odečítány a v okamžiku, kdy je jednotka poškozena

na 0 *HP* je zabita. Naše platforma bude tento systém samozřejmě podporovat, ale nebude ho nijak explicitně vyžadovat, bude tedy tvůrci umožněno použít jakýkoli jím implementovaný systém.

Jednotky mohou obdržet poškození z mnoha zdrojů, nejčastěji však útokem z blízka (tzv. *melee*) či z dálky (tzv. *ranged*). Útok na blízko je omezen dosahem, rychlostí útoků a velikostí uděleného poškození. Naše platforma bude podporovat komponentu poskytující útoky na blízko právě s těmito parametry. Útok na dálku lze rozdělit do dvou typů, tzv. *hit-scan* a *projektily*. První typ je reprezentován například laserovými zbraněmi, které v okamžiku výstřelu urazí celou vzdálenost, dokud nenarazí na terén či nějakou entitu (budovu či jednotku). Druhý typ v okamžiku výstřelu vytvoří projektil, který se v průběhu času pohybuje herním světem, dokud také nenarazí na terén či nějakou entitu. Naše platforma bude podle předlohy Strongholdu Crusader podporovat především projektilové útoky. Za tímto účelem bude vytvořena komponenta umožňující střelbu projektilů, dále projektily samotné, simulace jejich letu a především možnost výpočtu pro střelbu na pohyblivý cíl. Hit-scan útoky nebudou přímo podporovány, mělo by však být umožněno tvůrci hry tento typ útoků implementovat manuálně.

Shrnutí požadavků

Naše platforma bude podporovat jednotky s těmito vlastnostmi:

- J1: Pohyb jednotek volně kdekoliv nad terénem.
- J2: Podpora pohybu po terénu.
- J3: Ovládání jednotek a skupin jednotek.
- J4: Rozšířitelnost o složitější ovládání.
- J5: Podpora jednoduché i složité umělé inteligence v podobě vykonání libovolného kódu.
- J6: Podpora diskrétní i kontinuální produkce jednotek.
- J7: Přidávání jednotek při editaci mapy.
- J8: Přidávání jednotek při startu hry.
- J9: Podpora systému hit-pointů.
- J10: Útoky na blízko i na dálku.
- J11: Projektily.

1.1.3 Budovy

Stavba budov představuje jednu z hlavních prezentací hráčovi strategie. Podle postavených budov lze často vcelku přesně odhadnout, jakou strategii hráč zvolil, čímž je umožněno nepřátelům reagovat a adaptovat svou strategii odpovídajícím způsobem. Při volbě strategie lze ale narazit na problém, kdy je hráč nucen zvolit svou strategii před tím, než nalezne protivníky a tedy před tím, než může vidět

jejich strategii. Tento problém je velmi výrazný při tzv. "rock-paper-scissors" strategiích, popisovaných například v knize Fundamentals of Game Design[3, str. 329]. Hlavní myšlenkou tohoto návrhu her je situace, kdy strategie 1 porází strategii 2, strategie 2 porází strategii 3 a strategie 3 porází strategii 1. Ve chvíli kdy naše hra umožňuje hráči zvolit pouze jednu z těchto strategií bez viditelnosti protivníkovy strategie, degeneruje hra v loterie, zda hráč náhodně vybere správnou strategii porážející tu vybranou nepřítelem. Jedním z řešení tohoto problému, navrhovaných v článku od studia Oxeye[12], je co nejmenší rozdíl v síle prvních úrovní technologie, což umožní hráčům reagovat a změnit svoji strategii před tím, než je rozdíl mezi jejich silami neúnosně velký. Dalším možným řešením, použitým ve Stronghold Crusader[6], je neskrývat před hráčem nepřítelovu strategii. Toto řešení lze použít v různé míře, od odhalení nepřátelských budov po úplné odkrytí celé mapy, tedy pozice všech jednotek i budov, ať už přátelských, nepřátelských či neutrálních. Naše platforma použije toto poslední řešení, kdy budeme hráč vidět všechny jednotky a budovy všech hráčů.

Budovy mají ve hrách mnoho funkcí, mezi které patří například:

- Produkce jednotek
- Vylepšování jednotek
- Produkce surovin
- Uskladnění surovin
- Obrana
- Stavba budov
- Zkoumání technologií

Naše platforma se pokusí co nejvíce zjednodušit implementaci těchto funkcí poskytnutím programátorského rozhraní pro umístění budov a jednotek do herního světa, přidání a odebrání surovin hráči či střelbu projektilů. Dále umožníme tvůrci her přístup ke grafickému rozhraní, do kterého bude možné umístit okna, tlačítka a další elementy, které následně použitím programátorské rozhraní budou schopné implementovat všechny zmíněné funkce budov.

Jak již bylo řečeno v sekci o jednotkách 1.1.2, naše platforma bude nechávat volbu mezi kontinuální a diskrétní produkci jednotek na tvůrci hry. Budeme se tedy snažit do co největší míry podporovat oba tyto způsoby.

Obrana

Obrana bude podporována v podobě komponent, které umožní budovám jak útok na blízko, jako v případě pastí ve hře Stronghold Crusader[6], tak na dálku, jako například obranné věže ve hře Warcraft 3[4].

Dále v rámci funkce obrany umožňují v některých hrách budovy jednotkám pohybovat se po nich. Jak můžeme vidět na obrázku 1.3 ze hry Stronghold Crusader[6], jednotky mohou být umístěny na hradbách, věžích, bránách či na tvrzi. Na obrázku můžeme vidět různé typy jednotek, umístěné po obvodu hradu pro jeho obranu. Jak bylo řečeno v části o jednotkách, naše platforma bude

umožňovat neomezený pohyb jednotkami, tedy i ve vzduchu, na budovách či skrz budovy. Protože se pohyb po budovách vyskytuje ve velkém množství RTS her, poskytne platforma možnost rozšíření terénu o plochu budov a komponentu umožňující chůzi po těchto plochách.

Umístění na budovách poskytuje jednotkám ochranu před nepřátelskými jednotkami útočícími na blízko, které se musí nejdříve dostat do blízkosti cíle, než zaútočí.



Obrázek 1.3: Jednotky umístěné na budovách ve hře Stronghold Crusader[6]

V některých hrách, kde bohužel Stronghold Crusader[6] není jednou z nich, poskytuje vyvýšení nad terén jednotkám větší dostrel. Tento efekt nemusí být omezen pouze na vyvýšení pomocí budov, ale lze ho dosáhnout už při rozdílných výškách terénu, na kterém je umístěn střelec a jeho cíl, obecněji na rozdílu výšky pozice střelce a cíle, pokud se střelec nemusí pohybovat přímo po terénu. Naše platforma bude podporovat realistické chování projektileů splňující tuto vlastnost.

Budovy, podobně jako jednotky, mají ve většině her určitý počet tzv. "hit pointů", které určují počet zásahů, které může budova obdržet před tím, než bude zničena. Navíc oproti jednotkám mohou budovy často obdržet poškození pouze od omezené podmnožiny jednotek, nejčastěji pouze obléhacích strojů. Stejně jako u jednotek přenecháme systém poškození na tvůrce hry. Naše platforma pouze umožní budově reagovat na zásah projektilom či zbraní, ať už snížením svých *HP*, nebo ignorováním daného útoku v případě že přišel od jednotky či projektilu, který danou budovu nemůže poškodit.

Stavba budov

Budovy mají často restrikce, které je hráč nucen splnit před stavbou budovy. Tyto restrikce mohou sahat od reliéfu a typu terénu, přes existenci jiných budov ve stejném místě, po vlastnictví určitého množství surovin či typu jednotek. Naše platforma umožní tvůrci před stavbou budovy zjistit stav všech těchto typů restrikcí a případně vetovat stavbu budovy.

Jednou z restrikcí je výzkum určité technologie či stavba určité předcházející budovy. Tyto možné závislosti jsou blíže popsány v sekci Vývoj technolo-

gií1.1.5. Tímto druhem restrikcí jsou budovy uspořádány do postupně se zlepšujících úrovní, které hráč v průběhu času odemyká. Každá z úrovní obsahuje řadu rozdílných budov, umožňujících zvolit různé strategie. Platforma bude umožňovat tvůrce postupné zpřístupňování budov a jednotek, čímž bude tvůrce schopen implementovat postupné zkoumání nových technologií.

Existující hry využívají několik možností, jak hráči poskytnout zpětnou vazbu o splnění restrikcí při stavbě budovy. Jednoduší možností, použitou ve hře Stronghold Crusader[6], je zobrazení půdorysu v různých barvách podle splnění restrikcí. Další, složitější možností je zobrazení průhledného či jinak upraveného modelu budovy na místech, kde ji nelze postavit. Naše platforma bude podporovat jednoduší způsob, tedy zobrazení půdorysu budovy v různých barvách podle požadavků tvůrce hry.

Shrnutí požadavků

Naše platforma bude podporovat tyto vlastnosti:

- B1: Stavbu budov v herním světě.
- B2: Komponenty pro útok budov na blízko i na dálku.
- B3: Rozšiřitelnost dostupného terénu v herní mapě o prostor na budovách.
- B4: Podpora zvýšení dostřelu při umístění jednotky na vyvýšený terén, například budovu.
- B5: Stavba budov při editaci mapy.
- B6: Stavba budov při startu hry.
- B7: Podpora systému hit-pointů.
- B8: Tvůrcem definované reakce na obdržení zásahu budovou.
- B9: Zobrazení půdorysu v různých barvách pro zpětnou vazbu splnění restrikcí.
- B10: Kontrolu požadavků při stavbě budovy.
- B11: Přístup ke grafickému rozhraní, možnost zobrazení elementů hráči.

1.1.4 Suroviny

“Resource management”, tedy management surovin, je přítomný ve všech hrách žánru RTS již od jeho vzniku. Od koření v Dune II, přes zlato a dřevo ve Warcraft 3[4], po všechny typy surovin ve hře Stronghold[6], získávání surovin je jednou z hlavních motivací konfliktu v RTS hrách.

Systémy surovin lze rozdělit podle způsobu získávání a počtu typů surovin. Podle způsobu získávání můžeme systém surovin rozdělit na

1. Aktivní získávání surovin
2. Pasivní získávání surovin

Při aktivním získávání surovin existuje ovladatelná herní entita, která svým pohybem mezi pozicemi na mapě přináší suroviny. Tento pohyb může být ovládán hráčem, ale nejčastěji dokáže pracovat jednotka samostatně. Příkladem může být Warcraft 3 [4], kde speciální jednotky získávají dřevo a zlato přenášením z lesů/dolů do hráčovy hlavní budovy. Při pasivním získávání přibývají suroviny bez akcí entit, pouze díky vlastnictví určité části mapy nebo druhu budovy. Zdroj bývá nekonečný nebo skoro nekonečný, poskytující suroviny do obsazení nebo zničení zdroje. Každý z těchto stylů podporuje jinou strategii kontroly mapy.

Naše platforma bude podporovat jak pasivní, tak aktivní získávání surovin. Bude pouze na tvůrci, v jakém okamžiku budou suroviny přičteny, ať už v závislosti na čase nebo na pohybu určitých jednotek.

Shrnutí požadavků

Naše platforma bude podporovat tyto vlastnosti:

S1: Podporovat aktivní i pasivní získávání surovin.

S2: Poskytnout rozhraní pro přidání a odebrání surovin hráči.

1.1.5 Vývoj technologií

Volba vyzkoumaných technologií důležitou součástí strategické části RTS her.

Technologie jsou často uspořádány ve stromové struktuře či orientovaném acyklickém grafu (DAG), kde vyzkoumání technologie v rodičovském uzlu odemyká technologie následujících uzlů. Příklad takového uspořádání můžeme vidět v ukázce ze hry *Civilisation V*[5]1.4, kde vidíme počátek stromu technologií. V této hře jsou technologie uspořádány do DAGu, začínajícího v jednom kořeni. Můžeme vidět žluté vrcholy, značící vyzkoumané technologie, dále zelené vrcholy, značící technologie, které mají splněné všechny předky a mohou být vyzkoumány, černé technologie, které bude možné začít zkoumat po odemčení všech předků, a nakonec červené technologie, značící technologie v dalším věku. Dále můžeme vidět hrany spojující závislé technologie.

Další možné uspořádání je několik disjunktních stromů technologií, kdy je hráč nuten zvolit jeden z těchto stromů. Toto uspořádání můžeme vidět ve hře Company of Heroes [13], kde jsou hráči dostupné tři vzájemně výlučné cesty, každá zaměřená na jinou oblast boje. Každá z těchto cest je dále rozdělena na dvě větve postupně se zlepšujících technologií.

Každé větvení v grafu technologií představuje možné rozhodnutí hráče, kterou z větví se hráč vydá a které technologie odemknese. Toto rozhodnutí jsou jedním z hlavních projevů hráčovi strategie.

Vyzkoumání technologie může mít mnoho různých efektů. Nejčastějším efektem bývá odemknutí nového typu jednotek nebo budov. Další možností je změna vlastností již vlastněných jednotek nebo budov. V neposlední řadě pak může vyzkoumání technologie odemknout nové schopnosti nebo kouzla, které hráč může následně použít při taktických soubojích.

Explicitní strom technologií, viditelný na 1.4, se v RTS hrách vyskytuje spíše výjimečně. Nejčastěji je odemykání nových typů jednotek a budov umožněno stavbou určitého typu budovy nebo dosažením určitého stupně vylepšení již existující budovy. Jako příklad můžeme vzít Warcraft 3 [4], kde závislosti budov na



Obrázek 1.4: Výřez stromu technologií ze hry Civilisation V[5]

stupních vylepšení a existenci jiných budov tvoří strom technologií. Graf tvořený těmito závislostmi můžeme vidět na obrázku 1.5. Zde vidíme stupně vylepšení budov, reprezentované šedými šipkami, a závislosti budov na ostatních budovách a stupních jejich vylepšení. Aby byl hráč schopen postavit budovu, musí vlastnit všechny budovy na kterých je tato budova závislá v požadovaném nebo lepším stupni vylepšení. Ve hře je tento graf reprezentován požadavky zobrazenými při najetí myší na ikonu zamčené budovy.



Obrázek 1.5: Budovy a závislosti mezi nimi tvořící obdobu stromu technologií ve hře Warcraft 3

Jak bylo řečeno v sekci o budovách 1.1.3, naše platforma bude umožňovat tvůrci postupné odemykání budov a jednotek, což umožní tvůrci her implementovat výzkum nových technologií. Pro technologie měnící vlastnosti typů jednotek bude pouze na tvůrci, aby změnil logiku hry a chování umělé inteligence v závislosti na vyzkoumaných technologiích.

Shrnutí požadavků

Naše platforma bude podporovat tyto vlastnosti:

T1: Umožnit postupné odemykání budov a jednotek.

T2: Změny chování a vlastností jednotek v průběhu hry.

1.2 Uživatelé

V předešlé sekci jsme popsali druh her, který bude naše platforma podporovat. V této sekci popíšeme požadavky na chování platformy z pohledu všech druhů našich uživatelů.

1.2.1 Tvůrci her

Prvním druhem uživatele budou tvůrci her, používající naši platformu pro zjednodušení své práce a vyřešení problémů opakujících se ve většině RTS her. Tvůrce samozřejmě nemusí být pouze jeden člověk, vzhledem ke složitosti RTS her existuje při jejich tvorbě mnoho rolí, které požadují velmi různorodé schopnosti a mohou být plněny více lidmi. Příkladem rolí může být 3D umělec tvořící modely a animace, producent hudby, tvůrce umělé inteligence, programátor herní logiky a nakonec člověk, který toto vše integruje dohromady.

Programátor

Naším hlavním cílem je umožnit tvůrcům umělé inteligence a programátorům herní logiky použít všechny jazyky .NET Frameworku, především C#, pro jejich práci. Z programátorského pohledu bude naše platforma sloužit jako knihovna poskytující tyto funkce:

1. Udržovat aktuální stav hry a umožňovat dotazy na tento stav.
2. Vytvářet nové jednotky, budovy a projektily.
3. Registrovat si metody, které budou zavolány při určitých událostech v průběhu hry.
4. Upravovat terén.
5. Vykreslovat terén, jednotky, budovy, projektily a grafické rozhraní.
6. Ukládat a načítat stav hry.
7. Implementace pro základní problémy jako pohyb po terénu, let projektilů a výpočet úhlu střelby projektilů.
8. Ovládání kamery.

Protože implementace vykreslování a grafického rozhraní by byla nad rámec jedné bakalářské práce, využije naše platforma pro tyto účely existující herní engine UrhoSharp. Pro programátory následně poskytneme přístup k relevantním částem UrhoSharp enginu.

Platforma bude poskytovat ovládání kamery v několika módech. Tato funkcionality bude potřebná již pro implementaci editoru, poskytneme ji tedy i tvůrcům her. Prvním módem bude klasická RTS top-down kamera, jakou můžete vidět na ukázce ze hry Company of Heroes 1.6. Kamera je umístěna v dostatečné vzdálenosti pro zobrazení celých skupin jednotek, skloněna zhruba 45 stupňů od horizontální polohy. Ve hře Company of Heroes je ovšem oproti jiným starším RTS

hrám jako Warcraft 3 nebo Age of Empires 3 s kamerou možné rotovat a přiblížovat, což nám přijde jako velice atraktivní. Příklad tohoto pohybu kamery můžete vidět na obrázku 1.7, kde můžeme vidět pohled z Německé strany na vylodění na pláži Omaha. Toto bychom rádi podporovali i v naší platformě. Druhým módem kamery bude tzv. "free-float", kdy kamera volně "létá" nad terénem. Třetím módem bude sledování jednotky, kdy se kamera bude chovat jako v prvním módu, její pohyb bude ale řízen pohybem jednotky.



Obrázek 1.6: Klasický RTS pohled kamery ve hře Company of Heroes



Obrázek 1.7: Co lze vidět s pouhým přiblížováním a otáčením kamery hře Company of Heroes

Moddeři

Pro integraci vytvořených jednotek, budov, dlaždic a jejich modelů, textur a logiky umožní naše platforma vytvořit *balíček*, obsahující vše vytvořené programátory a umělci. Tento balíček poté platforma umožní přidat a načíst v libovolné instalaci platformy.

Platforma bude také sloužit jako editor úrovní, které budou využívat logiku, jednotky, budovy a typy terénu dodané pomocí balíčků. Vytvořené úrovně bude následně možné uložit zpět do balíčku a distribuovat spolu s balíčkem do dalších instancí naší platformy.

Pro editaci mapy poskytne platforma několik základních nástrojů a umožní tvůrcům modifikovat nástroje či přidávat své vlastní. Základní nástroje by měli umožnit editaci terénu mapy (změnu typu terénu na všechny tvůrci definované typy a editaci výšky dlaždic) a přidávání všech druhů jednotek a budov do mapy.

Formát ukládání úrovní bude definován otevřeně pomocí prostředků nezávislých na programovacím jazyce, címž chceme umožnit tvorbu separátních editorů map produkujících úrovně v námi používaném formátu.

1.2.2 Hráči her

Z pohledu hráče se bude platforma chovat jako instalovatelná aplikace, která hráči umožní za běhu přidávat balíčky a následně využít jejich obsah.

Při běhu platforma umožní plný přístup k nastavení UrhoSharp enginu, tedy k nastavení rozlišení, Vsync, triple buffer a dalších. Dále umožní správu balíčků, tedy přidávání, odebírání a spouštění. Při spuštění balíčku umožní platforma hráči výběr z existujících map, jak pro hraní, tak pro editaci. Dále platforma umožní vytvoření a editaci úplně nové mapy.

Při tvorbě nové mapy či editaci existující mapy mít bude hráč přístup ke všem jednotkám, budovám a typům terénu, ke kterým mu dají editační nástroje specifikované tvůrcem balíčku přístup. Následně bude hráči umožněno mapu uložit, a to přepsáním zdrojové mapy, kterou hráč načetl k editaci, nebo vytvořením nové mapy pod novým jménem.

1.3 Ukázková hra

Pro ukázkou bude vytvořen balíček s jednoduchou hrou demonstrující možnosti naší platformy. Ukázková hra bude zároveň sloužit jako referenční příklad použití naší platformy.

Ukázková hra bude obsahovat několik jednotek, demonstrujících tyto vlastnosti:

1. Jednoduchou a složitou umělou inteligenci
2. Plně automatické jednotky, neovladatelné hráčem
3. Jednotky útočící na dálku
4. Jednotky útočící na blízko
5. Aktivní získávání surovin

6. Pohyb po terénu
7. Pohyb nad terénem (létání)
8. Rozdílnou rychlosť pohybu rôznych jednotiek
9. Rozdílnou prístupnosť častí mapy pre rôzne jednotky
10. Pohyb po budovách

Demonstrovanými vlastnosťmi budov budou:

1. Restrikce na místo stavby
2. Neprostupnosť budov pro některé jednotky
3. Produkce surovin budovami
4. Přidání plochy nebo části plochy budovy jako přístupný terén pro určité typy jednotek

Jako obecné vlastnosti bude ukázková hra demonstrovat:

1. RTS mód kamery
2. Volný pohyb kamery
3. Sledování jednotky kamerou
4. Tvůrcem definované prvky v uživatelském rozhraní
5. Minimapu

Balíček obsahující ukázkovou hru bude demonstrovat tyto vlastnosti:

1. Tvorbu vlastních úrovní za použití jednotek, budov a typů terénu obsažených v balíčku.
2. Ukládání a načítání hry.

1.4 Cíle práce

Cílem této práce je vytvořit platformu pro vývoj 3D RTS her pro jednoho hráče za použití herního enginu UrhoSharp, umožňující vývojářům vytvářet hry jako separátně distribuované balíčky, které bude poté koncový uživatel schopen přidat do naší platformy nainstalované na uživatelském počítači a použít je pro hraní dodaných úrovní či tvorbu svých vlastních.

Při tvorbě hry bude umožněno tvůrci použít jazyky frameworku .NET pro vytvoření Umělé inteligence jednotek, budov a nepřátelských hráčů, pro vytvoření další logiky hry a pro přidání nástrojů do editoru map.

Požadované vlastnosti platformy:

1. Podporované vlastnosti jednotek:
 - (a) Pohyb jednotek (J1, J2)

- (b) Ovládání jednotek (J3, J4)
- (c) Umělá inteligence, definice chování (J5)
- (d) Produkce jednotek (J6)
- (e) Přidávání jednotek jako součást mapy (J7, J8)
- (f) Podpora systému hit-pointů (J9)
- (g) Útoky na blízko i na dálku (J10, J11)

2. Podporované vlastnosti budov:

- (a) Stavba budov v herním světě (B1, B9, B10)
- (b) Podpora obraných budov (B2, B4)
- (c) Rozšiřitelnost dostupného terénu o plochu budov (B3)
- (d) Přidávání budov jako součásti mapy (B5, B6)
- (e) Zničitelnost budov (B7, B8)
- (f) Produkce jednotek, surovin, stavba budov pomocí budovy (B11)

3. Podpora surovin:

- (a) Přidávání a odebírání libovolného počtu surovin (nejen celočíselných) (S1, S2)

4. Podpora výzkumu technologií:

- (a) Umožnit postupné odemykání dostupných jednotek a budov, umožnit změny chování jednotek za běhu (T1, T2)

5. Vlastnosti pro tvůrce balíčků:

- (a) Platforma musí umožňovat přidávání balíčků za běhu, obsahujících nové typy jednotek, budov, dlaždic, projektilů a hráčů spolu s jejich modely, texturami a AI.
- (b) Platforma musí umožňovat použití přidaných balíčků pro tvorbu map a uložení vytvořených map do balíčku použitého pro jejich tvorbu.
- (c) Editor map musí být rozšiřitelný o nástroje z balíčku.
- (d) Herní grafické rozhraní musí umožňovat tvůrci přidávat vlastní okna, tlačítka a další prvky.

6. Vlastnosti pro koncového hráče:

- (a) User interface pro stolní počítače, umožňující vybírání balíčků, map a oponentů, dále načítání a ukládání her, a nastavování zobrazení hry.
- (b) Herní user interface musí obsahovat minimapu, poskytující hráči přehled o větší části mapy než kterou vidí vlastní kamerou.
- (c) Ovládání kamery umožňující klasický top-down pohled, volné poleto-vání kamery po mapě a následování jednotky.
- (d) Ukládání a načítání hry.

2. Analýza

V první kapitole jsme specifikovali cíl naší práce, tedy implementaci platformy založené na herním enginu UrhoSharp umožňující tvorbu RTS her a jejich distribuci. V této kapitole popíšeme problémy při implementaci této platformy, námi zvolená řešení a jejich alternativy.

2.1 Herní engine

Jak jsme napsali v sekci 1.4 Cíle práce, naším cílem je vytvořit platformu za použití herního enginu UrhoSharp. „*UrhoSharp je multiplatformní 3D a 2D engine který může být použit pro tvorbu animovaných 3D a 2D scén za použití modelů, materiálů, světel a kamer*“[18], jak říká úvodní stránka dokumentace enginu. Jak už název napovídá, UrhoSharp je .NET binding pro Urho3D engine[14], což je opensource herní engine implementovaný v C++.

To do
(10)

2.2 Rozdíly systémů

Naším hlavním cílem byla implementace pro systém Windows především kvůli nejrozsáhlější podpoře frameworku .NET, dále kvůli zřejmým výhodám uspořádání ovládání v podobě klávesnice a myši, a v neposlední řadě kvůli naší zkušenosti s tímto systémem. Návrh platformy by měl zároveň umožňovat co nejjednodušší rozšíření na další systémy. Herní engine a platforma .NET sice mnohé rozdíly systémů abstrahuje a umožňuje multiplatformní řešení, existují ovšem oblasti, které i při využití těchto abstrakcí vyžadují pro každý systém specifické řešení. Při implementaci koster řešení pro jiné systémy než systém Windows jsme narazili na rozdíly, specifika a omezení, které v této sekci přiblížíme.

Při implementaci pro mobilní systémy existuje oproti PC systémům několik problémů, vycházejících především z rozdílných operačních systémů, velikostí obrazovek a způsobu přijímání vstupu od uživatele. Rozdíly mezi PC systémy (Windows, různé distribuce Linuxu, macOS) nejsou tak rozsáhlé, přesto se mohou vyskytnout problémy především kvůli různým implementacím platformy .NET používaných mimo systém Windows.

2.2.1 Zobrazení a ovládání

Na mobilních systémech je mnohem bližší vztah mezi GUI, tedy grafických uživatelským rozhraním, a ovládáním. Oproti PC systémům je zde jediným možným vstupem dotyková obrazovka. GUI musí tedy sloužit jak pro zobrazení informací hráči, tak pro získání drtivé většiny vstupu od hráče.

Dalším rozdílem je velikost obrazovky, která je obecně mnohem menší než u jiných systémů. I přes to, že herní engine umožňuje tvorbu grafického rozhraní použitelného na všech systémech, tyto dva rozdíly nutí nás i potencionální tvůrce her na naší platformě k výrazně odlišnému návrhu rozhraní pro mobilní systémy. Engine Urho3D poskytuje separátní vývojové prostředí, které je možné použít pro definici uživatelského rozhraní a export této definice do XML souboru, který

je poté možné načíst za běhu hry pro zobrazení specifikovaného uživatelského rozhraní. Tento systém lze použít k definici uživatelských rozhraní specifického pro cílový systém dané verze aplikace. Stejně tak tvůrci balíčků budou nutenci vytvořit dvě různé definice uživatelského rozhraní, a následně distribuovat dvě separátní verze balíčku, nebo za běhu rozhodovat, kterou z verzí načíst a zobrazit.

Příklad těchto problémů a jedno z možných řešení můžeme vidět na obrázcích ze hry Hearthstone[?], kde ?? ukazuje mobilní verzi hry a ?? ukazuje PC verzi hry. Jak je na první pohled zřejmé, vlastní herní plocha sdílí v obou verzích stejný design. Jedním z rozdílů je pozice kamery, která je v mobilní verzi umístěna mnohem blíže herní ploše a ukazuje její menší výřez. Toto je jedno z možných řešení menší velikosti obrazovek mobilních zařízení, díky kterému budou herní prvky na těchto obrazovkách větší. Další součástí řešení problému velikosti obrazovek je velikost fontu, která je v mobilní verzi mnohem větší než u PC verze. K této změně musí být následně provedena odpovídající změna designu karet a oblastí, kde se písmo a číslice vyskytují. Pro řešení problému vstupu pomocí dotykové obrazovky můžeme vidět na ?? dva různé stavy hry, kdy v prvním jsou hráčovy karty zmenšené a schované v pravém dolním rohu obrazovky, odkud jsou následně po "kliknutí" hráče přemístěny do centrální pozice a zvětšeny, čímž zakrývají velkou část herní plochy. Tento systém vytahovaní, přemístování a zvětšování ovládacích prvků aplikace je obecným trendem v mobilních zařízeních, umožňujícím větší velikost ovládacích prvků za cenu většího počtu interakcí uživatele se zařízením pro dosažení stejného cíle než by bylo potřeba v PC verzích aplikací.

Jak můžeme vidět na příkladu ze hry Hearthstone[?], vedou nás problémy s velikostí obrazovek a dotykovým ovládáním k separátnímu designu a implementaci uživatelského rozhraní a některých částí her. Pro tuto separátní implementaci jsme v naší práci připravili základní kostru, upustili jsme ovšem od konečné implementace z důvodu nedostatku času.

2.2.2 JIT vs. AOT

Dalším rozdílem, tentokrát s rozdílným chováním i mezi jednotlivými mobilními systémy, je jejich chování k executable souborům aplikací. Jak píší Joseph a Ben Albahari[?], C# a další jazyky cílené na platformu .NET se překládají do "Common Intermediate Language" (CIL), z kterého jsou obvykle až za běhu komplikovány do instrukční sady stroje, na kterém právě běží. Tento způsob se označuje jako "Just-In-Time" (JIT) komplikace, a je standardním způsobem spouštění .NET aplikací. V některých případech je ale použit jiný způsob, a to tzv. "Ahead-of-time" (AOT) komplikace, kdy je CIL kód ještě před distribucí zákazníkovi zkompilován do instrukční sady cílového stroje a následně je distribuována tato již zkompilovaná verze. Tento způsob je používán pro zrychlení odezvy při větších velikostech assembly, kde se předchází zpoždění v důsledku komplikace CIL kódu. Další využití, zde již ne pouze optimalizační, ale vynucené systémem samotným, je při distribuci na systém iOS[11].

Tato skutečnost znemožňuje naší platformě jednoduché nahrání assembly pomocí reflexe a nutila by nás k složitějšímu řešení, které by se podle aktuálního systému muselo rozhodovat, kterou verzi assembly nahrát. Navíc by tento způsob nutil tvůrce balíčků přeložit svůj kód pro všechny možné architektury. Toto je důvod, proč jsme upustili od podpory systému iOS.

To do
(11)

2.2.3 Souborové systémy

Každá aplikace má několik odlišných druhů souborů. Tyto druhy můžeme odvodit z pro ně navržených adresářů ve Windows API[?], či z tohoto API odvozené implementace v platformě .NET [?]. Těmito druhy souborů jsou:

- Roaming user data
- Local user data
- Private app data
- Public app data
- Static app data

Některé z těchto druhů jsou na různých systémech sloučeny. Systém Android například nepodporuje multi user systémy, navíc tento systém neumožňuje sdílení souborů pomocí filesystému, které nahrazuje “FileProvider” API. Nemá zde tedy smysl mluvit o user data nebo public app data. Všechna data aplikace jsou privátní a mohou být uložena na jednom ze dvou míst, internal nebo external storage. Pro specifická data jako nastavení poskytuje pak systém Android tzv. Shared preferences a dále přístup k databázy.

Přístup k souborům, jak distribuovaným spolu s aplikací, tak vytvářených či přidávaných po instalaci aplikace, se mezi systémy odlišuje v několika ohledech.

Přístup k souborům distribuovaným spolu s aplikací, v našem případě s naší platformou, se mezi systémy odlišuje několika způsoby. Prvním je jejich otevření, či vůbec nalezení. Na systému Android je každá aplikace distribuována jako .apk soubor. Formát Apk je zip archiv, obsahující všechny soubory naší aplikace, od kódu, přes assety, po preference. Tento archiv je v duchu Linuxového VFS přímo namapován do stromu souborového systému viditelného z naší aplikace. Bohužel .NET filesystem API nedokáže s tímto mapováním pracovat, tedy není možné ho využít pro čtení těchto souborů. Řešením je využití Xamarin.Android zabalujícího Android Java API, které s tímto archivem pracovat dokáže.

Toto nás nutí pro vyčlenění komponenty pro práci se soubory do zvláštní implementace pro každý systém.

Dalším problémem je, že oproti PC systémům nelze do těchto souborů zapisovat, ani za použití jiného API. Toto není sice dobrý design aplikace ani na PC systémech, kde by data specifická pro daného uživatele měla být umístěna do adresáře jako `MyDocuments/$app` na Windows nebo `/home/$USER/$app` na systému Linux, protože adresář, v kterém je aplikace nainstalovaná, může být pro většinu uživatelů namapován pouze pro čtení, ale při tvorbě jednodušších aplikací je vcelku obvyklé vytvářet soubory v aktuálním adresáři aplikace a zapisovat do nich.

2.2.4 Shrnutí

Velká část problému popsaných v této části je řešitelná a tato řešení jsme u každého problému nastínili. Naše platforma bude obsahovat kostru těchto řešení, nebude je ovšem implementovat úplně, především kvůli nedostatku času.

Některé problémy, jako například zákaz JIT komplikace na systému iOS, jsou ovšem fatální pro myšlenku naší práce a byli jsme nuceni vyřadit tento systém i z možnosti budoucí podpory.

2.3 Formát a načítání dat

Důležitou součástí implementace naší platformy je systém balíčků pro distribuci vytvořených her. Tyto balíčky obsahují všechny součásti hry, od modelů a textur, přes logiku a umělou inteligenci, po mapy a úrovně vytvořené tvůrcem hry. Všechny tyto součásti musí naše platforma být schopna načíst za běhu a použít jak pro tvorbu nových map, tak pro hraní již existujících.

2.3.1 Struktura balíčku

Pro implementaci načítání balíčků musíme definovat strukturu, kterou budou balíčky splňovat, a podle které bude platforma určovat typy souborů a jejich závislosti.

První možností je založit strukturu balíčku na pevné adresářové struktuře, kde každý balíček bude tvořen jedním adresářem obsahujícím další pevně specifikovanou podadresářovou strukturu. Jednotlivé typy zdrojů, tedy 3D modely, textury, popis jednotek nebo skripty, by následně byly rozděleny a identifikovány touto adresářovou strukturou. Pro popis typů jednotek, budov, projektilů, dlaždic a nepřátele jsme se inspirovali v existujících hrách, ať už Civilization V nebo Kerbal Space Program, a využili jsme XML soubor pro definici závislostí. Dalšími možnostmi bylo využít formát JSON nebo dokonce definovat vlastní formát. Možnou výhodou formátu JSON je jeho expresivita, umožňující minimalizovat velikost souborů. Vzhledem k velikosti grafických dat jsme ovšem usoudili, že tato výhoda není dostačující pro volbu tohoto formátu. Pro formát XML jsme se rozhodli především kvůli vestavěné podpoře v platformě .NET a možnosti validace vůči schématu, která nám ulehčí od implementace vlastní validace.

Toto rozdělení ovšem vedlo ke dvěma problémům. Prvním bylo velké množství malých XML souborů, jejichž správa by mohla vést k častým omylům a následným chybně pojmenovaným souborům, špatným odkazům a chybějícím souborům. Druhým problémem bylo přidávání balíčku do běžící hry. Hráč by mohl sice specifikovat adresář reprezentující balíček, následné ověření, zda je tento balíček korektní a lze ho nahrát nás nutí k procházení adresářové struktury, načítání mnoha souborů a k pokusům o jejich načtení a validaci.

Řešením bylo vytvořit centrální XML soubor, definující celý balíček. Všechny typy jednotek, budov, projektilů, nepřátele, logik úrovní, všechny úrovně obsažené v balíčku a další jsou popsány v tomto souboru. Následně všechny assety, tedy modely, textury či assembly mohou být specifikovány relativní cestou vůči adresáři obsahujícímu tento soubor. Tímto způsobem lze replikovat předchozí uspořádání, kde je každý typ assetů rozdělen do vlastního adresáře, ale navíc tento způsob umožňuje tvůrci balíčku specifikovat vlastní rozdělení a umístění assetů. Zároveň toto uspořádání ulehčuje přidání balíčku a ověření jeho korektnosti, kde stačí, aby uživatel zadal cestu k tomuto souboru, a pouhou validací XML souboru podle schématu lze ověřit jeho správnost.

Tedy v konečném řešení je každý balíček reprezentován jedním XML souborem, který dále obsahuje relativní cesty, odkazující na zbylý obsah balíčku. Tento soubor má formát daný pevným schématem a tento formát je kontrolován při každém načítání.

2.3.2 Data entit

Nyní již víme, jak data rozdělit a odkazovat se na ně. Následně musíme určit, jaká data budeme u jednotlivých typů entit požadovat a jaká data umožníme tvůrcům her si zde uložit. Naším cílem je umožnit tvůrcům specifikovat typy entit a k nim náležející data, definující vlastnosti těchto typů entit. Naše platforma bude vynucovat specifikaci nezbytných dat, jako jsou 3D model, textury, reference na assembly obsahující logiku, barva na minimapě a ikona pro zobrazení na liště jednotek. Protože jsou tyto vlastnosti společně většině entit v námi podporovaných typech her, nahrává naše platforma automaticky zde specifikované vlastnosti při vytvoření entity v herním světě a následně jsou tato data používána součástmi naší platformy.

Dále umožníme přidat libovolná další data do XML elementu reprezentujícího typ entity pro použití tvůrcem hry v jeho implementaci logiky. Tato data nebude naše platforma nijak validovat, pouze je při vytvoření entity v herním světě předá při načítání její logiky. Bude pouze na tvůrce této logiky, aby získaná data validoval a následně z nich inicializoval logiku či použil systém výjimek k oznamenání chyby v datech. Tento způsob implementace umožní tvůrcům vytvářet universálnější logiku, kterou budou schopni odlišit právě načítanými daty ze souboru. Alternativou k tomuto řešení by bylo všechna data, která zde tvůrci uloží do souboru, zahrnout do kódu logiky.

2.3.3 Formáty assetů

V předešlé části jsme popsali, jak náš systém balíčků umožňuje zaznamenat umístění dat potřebných pro spuštění úrovně. V této části dále upřesníme, jakých formátů můžou tato data být a jaká jsou omezení při použití určitých formátů dat. Slovíčkem *“asset”* označujeme jakákoli data, které může tvůrce hry poskytnout naší platformě a použít je při tvorbě hry. Tato data lze rozdělit do několika kategorií:

- Grafická data
- Logika a pluginy
- Další

V následujících částech popíšeme jednotlivé kategorie dat, jejich formáty a využití.

Grafická data

Mezi grafická data patří především 3D modely a k nim náležící textury a animace. Podporované formáty těchto dat jsou omezeny námi používaným herním enginem Urho3D. Tento engine interně používá knihovnu Open Asset Import

Quote
proč
je to
špatně
(12)

Library (Assimp)[1], která umožňuje načítání mnoha formátů 3D modelů do uniformní reprezentace, která je následně používána vlastním enginem.

Mezi další grafická data používaná naší platformou jsou textury dlaždic. Jak blíže popíšeme v sekci o implementaci mapy2.4.1, je právě tato implementace jednou z hlavních služeb poskytovaných naší platformou tvůrcům her. Tato implementace také zahrnuje zobrazení mapy a dlaždic, ze kterých je složena. Každá dlaždice má svůj typ a z tohoto typu odvozen svůj vzhled. Každý z těchto typů dlaždic je, jak jsme popsali v sekci o struktuře balíčků ??, definován XML elementem, obsahujícím právě cestu k textuře dlaždice a část této textury odpovídající tomuto typu dlaždice. Tuto část textury následně naše platforma zkopíruje do nové textury, obsahující textury všech typů dlaždic v aktivním balíčku. Proč takovou akci děláme je blíže popsáno v sekci o implementaci mapy2.4.1. Pro tuto akci musí být textura manipulovatelná z našeho C# kódu implementace platformy. Bohužel UrhoSharp wrapper v aktuální verzi neposkytuje přístup k Urho3D dekomprezii textur, není tedy možné pro tento typ grafických dat použít komprimované formáty textur.

V neposlední řadě požaduje naše platforma pro každý druh entit texturu obsahující jejich ikony a pro každý typ entity specifikaci části textury odpovídající ikoně tohoto typu. Tyto data jsou požadovány pro základní implementace nástrojů editoru úrovní, které umožňují vytváření jednotek a budov v herním světě. Tyto nástroje vytvářejí tlačítka, kterými uživatel volí aktuálně umístovanou jednotku či budovu. Vzhled těchto tlačítek je určen právě touto texturou.

Tvůrci je umožněno využít celou sílu enginu UrhoSharp, může tedy do svého balíčku přibalit jakákoli grafická data podporovaná tímto enginem a následně je využít.

Logika a pluginy

Jedním z hlavních cílů naší práce bylo umožnit využití jazyka C# jako skriptovacího jazyka a tím i jeho použití při tvorbě logiky hry a umělé inteligence nepřátele. Pro splnění tohoto požadavku musíme v našem programu zahrnout Common Language Runtime (CLR), tedy virtuální stroj platformy .NET, který přeloží CIL kód obsažený v assembly a umožní nám ho spustit. Nejjednodušším a nejlogičtějším řešením je využít C# a .NET pro tvorbu vlastní platformy. V takovém případě budeme mít v našem procesu zaručeně obsaženou CLR a volání pluginů bude možné uskutečnit pomocí mechanismu Reflection. Tato myšlenka nás následně vedla k použití UrhoSharp jako herního enginu.

Alternativním řešením by bylo využití jiného jazyka pro tvorbu platformy, například C++, a následné využití C++/CLI pro vytvoření "mostu" mezi unmanaged kódem tvořícím platformu a managed kódem, umožňujícím nahrávání pluginů za běhu a jejich využití z unmanaged kódu. Tuto alternativu jsme zavrhlí především kvůli složitosti implementace. Při použití herního enginu připraveného pro .NET, jako je právě UrhoSharp, můžeme tvůrcům pluginů poskytnout plnou sílu tohoto enginu a umožnit jim využít všechny jeho schopnosti bez našeho zásahu. V případě využití enginu v jiném jazyce by musel náš "most" explicitně implementovat všechny schopnosti enginu, které by jsme chtěli poskytnout tvůrcům pluginů a přeposílat každý požadavek na jeho implementaci v enginu.

Obě předešlá řešení využívají systém "Reflection" pro načítání tvůrcem dodaných assembly, nalezení tříd odpovídajících jednotlivým jednotkám, vytvoření

instancí těchto tříd a jejich následné použití. Pojmem "Reflection" označujeme sadu tříd umožňující introspekci .NET assembly, poskytující informace o třídách obsažených v těchto assemblies, jako například implementovaná rozhraní, poskytované metody či obsažené atributy. Dále "Reflection" umožňuje načítání existujících assembly za běhu programu, či dokonce generování nových assembly.

Assembly jsou identifikovány jménem, verzí, *culture* a veřejným klíčem. Při načítání je assembly nahrána do tzv. kontextu. Existují čtyři kontexty, do kterých jsou assembly nahrávány. Těmito kontexty jsou[17]:

- Default Load Context
- Load-From Context
- Reflection-only Context
- No Context

Reflection-only context slouží pro zkoumání assembly pomocí reflection a znemožňuje vykonání kódu nahraného do tohoto kontextu, proto je pro nás nezajímavý a dále ho nebudeeme rozebírat.

Default Load Context je kontext, ve kterém je nahrána assembly naší aplikace a všechny její závislosti. Do tohoto kontextu lze manuálně nahrávat další assembly, pokud se tyto assembly nachází v *GAC*, *applicationBase* nebo *PrivateBinPath*. Pokud je identická assembly již nahrána, nenahrává se znova ale vrací se reference na již nahranou assembly. Závislosti nahrávaných assembly jsou automaticky vyhledávány na těch samých místech.

Load-From Context je kontext, do kterého nahrává assembly metoda *Assembly.LoadFrom*. Do tohoto kontextu lze nahrát assembly specifikováním cesty navíc ke čtyřem výše zmíněným vlastnostem, a tato cesta je přidána jako pátá identifikační vlastnost assembly. Tímto způsobem lze nahrávat assembly ležící mimo GAC, *applicationBase* a *PrivateBinPath*. Závislosti jsou hledány v Default Contextu, případně v adresáři, ze kterého byla assembly nahrána a nakonec na zmíněných cestách pro nahrávání do Default Contextu.

No Context je využíván při načítání assemblies vygenerovaných pomocí reflection emit a *Assembly.LoadFile*. Navíc je to jediný způsob, jak načíst dvě verze té samé assembly. Pod pokličkou je vytvořen každé nahrané assembly zvláštní privátní kontext. Problémem tohoto kontextu je, že nejsou automaticky nahrávány závislosti. Tedy nezbývá nic jiného než závislosti nahrát manuálně před načtením assembly nebo odchytit *AssemblyResolve* event.

V naší platformě používáme *Assembly.LoadFrom*. Tento způsob nám umožňuje nahrávat assembly z libovolných podadresářů uvnitř tvůrce tvořených balíčků, bez omezení na jejich adresářovou strukturu. Díky načítání závislostí ze zdrojového adresáře assembly mohou tvůrci her přibalit jimi používané knihovny do balíčku a ty budou následně při použití automaticky načteny.

Druhou možností by bylo vynutit tvůrce balíčků specifikovat cesty, ve kterých se mohou vyskytovat assembly, a tyto cesty následně přidat do *PrivateBinPath*. Tento přístup by ovšem omezil místa, kde může naše platforma mít umístěné balíčky, na podstrom adresáře, ve kterém je umístěna naše platforma. Jak jsme psali v části o platformách a souborových systémech 2.2.3, není možné na všech platformách umístit balíčky do adresářového podstromu aplikace. Na platformách,

na kterých to možné je, to poté nemí dobrý design umísťovat zapisovatelná a proměnlivá data do podstromu adresáře aplikace, protože uživatel nemusí mít práva do tohoto adresáře a jeho podstromu zapisovat, jak říká tento, i když již trochu zastaralý, návod pro umisťování souborů v systému Windows[9].

Naše aktuální implementace nijak neřeší zahazování assemblies ve chvíli, kdy hra končí a hráč načítá jiný balíček. Vzhledem k velikosti assembly v paměti by ale toto neměl být při běžných počtech používaných balíčků problém. V budoucnu by bylo možné využít systému AppDomain, čímž by jsme oddělili balíčky od sebe a umožnili jejich zahazování nepoužívaných assemblies. Toto řešení by ale mohlo narazit na problém při rozšiřování na další platformy a implementace platformy .NET, které nemusí systém AppDomain podporovat.

To do
(13)

2.3.4 Formát uložených úrovní

Ukládáním úrovně rozumíme serializaci aktuálního stavu hry a uložení takto serializovaných dat do souboru.

Pro serializaci jsme měli několik požadavků.

1. Otevřenost schématu serializovaných dat
2. Minimalizace velikosti serializovaných dat
3. Rychlosť serializace a deserializace

Účelem prvního požadavku je umožnit tvorbu nezávislých editorů úrovní, importujících a exportujících náš formát dat. Tento požadavek splňuje serializace do XML, specifikovaného pomocí XSD schématu, a binární serializace popsaná pomocí interface description language. Příkladem takové binární serializace je formát Protocol buffers[?] od společnosti Google. Z binárních serializačních frameworků je pro jazyk C# nejlépe podporován právě formát Protocol buffers, ať už použit sám o sobě či za podpory knihovny protobuf-net, umožňující automatickou generaci IDL souborů z anotací ve zdrojovém kódu.

Druhý a třetí požadavek nás vedl k binární serializaci, která minimalizuje velikost dat a má nejrychlejší zpracování. Tuto skutečnost sděluje jak dokumentace Protocol buffers[7], tak ji můžeme vidět experimentálně podloženou v benchmarku Maxima Novaka[10]. Tímto jsme vybrali formát Protocol buffers, který lze v jazyce C# použít buď separátní specifikací message a následnou manuální serializací, nebo využitím knihovny Protobuf-net, která z anotací ve zdrojovém kódu generuje specifikaci message a metody pro serializaci a deserializaci dat.

Bohužel náš graf scény byl natolik složitý, že serializace pomocí Protobuf-net začínala být příliš složitá. Vzhledem k tomu, že samy Protocol buffers nepodporují reference, nemá ani Protobuf-net velkou podporu referencí. Přestože by nejspíš naše data šli serializovat pomocí Protobuf-net, rozhodli jsme se použít manuální specifikaci a serializaci, která nám dává větší kontrolu nad postupem serializace a konečným formátem dat. To nám navíc umožnilo rozdělit specifikace do separátních souborů podle logických závislostí, okomentovat tyto specifikace a případně distribuovat separátně.

2.4 Reprezentace stavu hry

V předchozí části jsme popsali, jak je stav hry serializován a následně uložen. V této části popíšeme, jak je stav hry reprezentován za běhu.

2.4.1 Mapa

Jednou z hlavních funkcionalit poskytovaných naší platformou tvůrcům logiky her je reprezentace mapy a možnost dotazovat se na její stav.

Logické rozdělení

Existuje několik způsobů, jakými lze rozdělit mapu.

Grafická reprezentace

2.5 Poskytované služby

Naší platformou podporovaný druh her často požaduje služby. Platforma je proto implementuje a poskytuje tvůrcům her.

2.5.1 Pathfinding

2.5.2 Projektily

Simulace projektilů je častým problémem v RTS hrách, tedy v typu her, který naše platforma chce podporovat. Rozhodli jsme se proto implementovat tuto službu, umožňující výpočet počátečního směru projektilu při střelbě na pohyblivý cíl, simulaci letu se stálou gravitací a detekci zásahů.

Typy projektilů

Existuje několik typů simulací projektilů. [?]

3. Ukázková hra

4. Programátorská dokumentace

5. Uživatelská dokumentace

Závěr

Seznam použité literatury

- [1] Open-asset-importer-lib. <http://www.assimp.org/index.php>.
- [2] https://en.wikipedia.org/wiki/Dune_II.
- [3] ADAMS, E. (2009). *Fundamentals of Game Design*. New Riders, 2 edition. ISBN 0-321-64337-2.
- [4] BLIZZARD ENTERTAINMENT, INC. Warcraft 3. <https://playwarcraft3.com/en-us/>. Accessed:2019-03-18.
- [5] FIRAXIS GAMES, INC. (2010). Civilization V. <https://civilization.com/civilization-5/>.
- [6] FIREFLY STUDIOS (2002). Stronghold Crusader. <https://fireflyworlds.com/games/strongholdcrusader/>.
- [7] GOOGLE (2018). Developer Guide. <https://developers.google.com/protocol-buffers/docs/overview>.
- [8] LANTZ, J. (2008). Opinion: The Evolution of the Modern RTS. http://www.gamasutra.com/php-bin/news_index.php?story=18326.
- [9] MICROSOFT (2006). Chapter 4: Data and Settings Management. [https://docs.microsoft.com/en-us/previous-versions/ms995853\(v=msdn.10\)](https://docs.microsoft.com/en-us/previous-versions/ms995853(v=msdn.10)).
- [10] NOVAK, M. (2014). Serialization Performance comparison (C#/.NET) – Formats & Frameworks (XML–DataContractSerializer & XmlSerializer, BinaryFormatter, JSON–Newtonsoft & ServiceStack.Text, Protobuf, MsgPack). <https://maxondev.com/serialization-performance-comparison-c-net-formats-frameworks-xmldatacontr>
- [11] O'BRIEN, L., SCHONNING, N., DUNN, C., UMBAUGH, B. a PAKALA, Y. (2017). iOS App Architecture. <https://docs.microsoft.com/en-us/xamarin/ios/internals/architecture>. Accessed:2019-04-08.
- [12] OXEYE GAME STUDION (2008). RTS Game-play Part 3: Build Options. <http://www.oxeyegames.com/rts-game-play-part-3-build-options/>.
- [13] RELIC ENTERTAINMENT (2006). Company of Heroes. <http://www.companyofheroes.com/>.
- [14] URHO3D. Urho3d. <https://urho3d.github.io/>.
- [15] WALKER, M. H. (2004). Strategy Gaming: Part II. <https://web.archive.org/web/20070129065355/http://archive.gamespy.com/articles/february02/strategy02/>. [Online; seen 2019-04-01].
- [16] WALKER, M. H. (2004). Strategy Gaming: Part V – Real-Time vs. Turn-Based. <https://web.archive.org/web/20081201113359/http://archive.gamespy.com/articles/february02/strategygames05/index.shtml>. [Online; seen 2019-04-01].

- [17] WENZEL, M., YISHENGJIN, LATHAN, L., PRATT, T., PETRUSHA, R., HOFFMAN, M., JONES, M. a DEV, A. (2017). Best Practices for Assembly Loading. <https://docs.microsoft.com/en-us/dotnet/framework/deployment/best-practices-for-assembly-loading>.
- [18] XAMARIN (2015). UrhoSharp. <https://docs.microsoft.com/en-us/xamarin/graphics-games/urhosharp/>.

A. Přílohy

A.1 První příloha

To do . . .

- 1 (p. 4): Možná vynechat Možná vynechat
- 2 (p. 4): citace
- 3 (p. 4): citace
- 4 (p. 4): možna graf nasledujiciho, jak se ovlivnujou
- 5 (p. 6): Další druhy popsat další druhy implementací
- 6 (p. 9): Přeuspořádat Lépe uspořádat sekci o budovách
- 7 (p. 11): zničitelnost
- 8 (p. 11): různé druhy poškození
- 9 (p. 11): restrikce na umístění
- 10 (p. 20): porovnání s dalšími enginy
- 11 (p. 21): Popsat iOS restrikce na JIT
- 12 (p. 24): Quote proč je to špatně Quote proč je to špatně
- 13 (p. 27): odnačítání assemblies