FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Počítačová hra - realtime strategie inspirovaná hrou Starcraft
Student:	Jakub Hrabálek
Vedoucí:	Ing. Miroslav Balík, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Navrhněte a implementujte pc hru - realtimovou strategii inspirovanou hrou Starcraft.

1. Navrhněte mechaniky hry, princip vybalancování a popište je.
2. Hru implementujte.
3. Diskutujte problém navigace a hledání cest skupin objektů a řešení implementujte.
4. Implementujte hru více hráčů v architektuře client-server s podporou více než dvou hráčů.
5. Navrhněte a implementujte skriptovanou umělou inteligenci s možností libovolného kombinování {hráč, AI} * <2, n>.
6. Navrhněte přívětivé uživatelské rozhranní a spustitelné balíčky pro Linux a MS Windows.
7. Aplikaci podrobte uživatelským testům a diskutujte případné změny podle zpětné vazby.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 15. února 2022 v Praze.

Bakalářská práce

POČÍTAČOVÁ HRA - REALTIME STRATEGIE INSPIROVANÁ HROU STARCRAFT

Jakub Hrabálek

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Miroslav Balík, Ph.D.
10. května 2022

Obsah

Seznam obrázků

Seznam výpisů kódu

Prohlášení

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

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

V Praze dne 10. května 2022 .....

Abstrakt

Cílem bakalářské práce je zejména implementace počítačové hry žárnu strategie v reálném čase. Jsou popsány základní mechaniky a principy hry. Práce se zabývá analýzou a návrhem hry a popisem řešení problémů a algoritmů, které bylo nezbytné během vývoje prozkoumat. Těmito tématy jsou detaily grafického prostředí, efekty, herní mapa a její generátor, herní objekty a jejich navigace a hledání cest v rámci prostředí mapy. Důležitým prvkem je také hra po síti a tedy problém synchronizace stavu hry a popis konkrétního řešení. Vedle síťové hry se práce také zabývá skriptovanou umělou inteligencí a možností více než dvou hráčů a kombinováním lidských hráčů společně s umělou inteligencí. Důležitým výstupem je rozsáhlý funkční projekt hry vytvořený v jazyce Python optimalizovaný pomocí nástroje Cython používající 2D grafickou knihovnou Pyglet společně s dalšími strukturami v OpenGL. Výsledná hra je dostupná ve formě spustitelných balíčků pro platformy Windows a Linux.

Klíčová slova hra, multiplayer, strategie v reálném čase, Python, Cython, Pyglet, OpenGL, AI, client-server

Abstract

The aim of the bachelor thesis is mainly the implementation of a real time strategy computer game. The basic mechanics and principles of the game are described in the document. The work deals with the analysis and design of the game and a description of problem solving and algorithms that were necessary to explore during development. These topics are details of the graphical environment, effects, game map and map generator, game objects and their navigation and finding paths within the map environment. An important element is also the the network game and thus the problem of synchronizing the state of the game and a description of a specific solution. In addition to network game, the work also deals with scripted artificial intelligence and the possibility of more than two players and combining human players together with artificial intelligence. An important output is a large functional game created in Python optimized by Cython using the Pyglet 2D graphics library together with other structures in OpenGL. The resulting game is available in the form of executable packages for Windows and Linux platforms.

Keywords hra, multiplayer, real-time strategy, Python, Cython, Pyglet, OpenGL, AI, client-server

Shrnutí

Cíl práce

Cílem práce je vytvoření funkčního prototypu hry strategie v reálném čase společně s prozkoumáním problémů, které s vývojem hry souvisí a jejich následná analýza a popis. Součástí výstupu jsou také spustitelné balíčky, stručný návod a demonstrace v podobě videí.

Vývojové prostředí a grafika

Hra je vytvořena v jazyce Python s optimalizováním pomocí nástroje Cython (konverze kódu do C/C++). V práci jsou nastíněny možnosti a výhody společně s nevýhodami tohoto přístupu.

Grafické prostředí používá knihovnu Pyglet a OpenGL struktury a je reprezentováno 2D mapou s generátorem. Implementovány jsou systémy částicových efektů pro střely, světla apod. Grafika hry je vlastní výroby vytvořená metodou pixel-art.

Herní objekty a hledání cest

V práci jsou popsány herní objekty a jejich interakce v prostoru hry – zejména typy jednotek a střel. Současně s tím je rozveden způsob pohybu a navigace objektů a jakým způsobem jsou řešeny různé velikosti objektů.

Způsob hledání cest probírá nevýhody algoritmu A* a rozvádí řešení pomocí vektorového pole. Je popsán konkrétní algoritmus pohybu objektů a detekce překážek společně se řešením kolizí s ostatními objekty. Řešení je podrobeno dílčím optimalizacím a jsou popsány limitace implementovaného řešení a případné další nápady k vylepšení.

Hra více hráčů a umělá inteligence

Sekce s problematikou hry více hráčů rozebírá možnosti synchronizací stavu hry a popisuje implementaci sdílené deterministické simulace. Popisuje problém zachování determinismu a riziko de-synchronizace, které je zajištěno mimo jiné používáním výhradně celočíselných algoritmů a principů.

Implementace umělé inteligence je skriptovaná a v základu na bázi stavového automatu. Součástí logiky je také analýza okolního prostředí a vyhodnocování hrozeb nebo rozmístění jednotek a budov.

Projekt podporuje hru více než dvou hráčů a libovolné kombinování lidských hráčů a umělé inteligence a způsoby jsou v práci popsány a demonstrovány.

Testování

Hra je řádně otestována uživatelskými testy a jejich výstupy společně s řešením nalezených problémů jsou v rámci práce popsány.

Práce se také obecně zabývá testováním počítačové hry a diskutuje problém testování rozsáhlého stavového prostoru a demonstruje určité postupy, které byly použity a prakticky provedeny.

Závěr

Všechny náročné cíle se podařilo splnit a byl vytvořen funkční hratelný prototyp, který byl podroben uživatelským testům s pozitivní zpětnou vazbou.

Seznam zkratek

GIL	Global Interpreter Lock
API	Application Programming Interface
GUI	Graphic User Interface
BFS	Breadth-first search
A*	A star
GPU	Graphics Processing Unit
RTS	Real-time strategy
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
FPS	Frames Per Second
AI	Artificial Intelligence
OOP	Object-oriented programming
HSV	Hue Saturation Value
GRASP	General Responsibility Assignment Software Patterns
2D	Two-dimensional
OS	Operating system

Kapitola 1
Cíl práce

Hlavním cílem a výstupem práce je vytvoření funkčního prototypu hry strategie v reálném čase, který bude hratelný a podroben uživatelským testům.

Neoddělitelným cílem je také prozkoumání všech problémů, které s vývojem hry souvisí, a to na nižší úrovni detailů – řešení tedy nepředpokládá použití vysoko-úrovňových frameworků jako jsou Unity, Unreal Engine a další, ve kterých jde spíše o kombinování již vytvořených částí, nýbrž o detailnější analýzu a implementaci jednotlivých problémů. Tento předpoklad nicméně není na úkor výsledku, naopak poskytuje volnost v možnostech implementace, optimalizace a funkčnosti celého projektu. Nezbytnou součástí je podrobení prototypu uživatelskému testování a provedení úprav na základě zpětné vazby.

Poznatky získané během vývoje jsou popsány v této textové části práce – jedná se zejména o grafické prostředí, herní mapu a generování, navigace objektů a hledání cest a hru více hráčů po síti nebo proti umělé inteligenci. Popis je zaměřen na všechny detaily, které jsou pro hru klíčové a bylo nezbytné jim během vývoje věnovat více času.

Dalším výstupem je stručný uživatelský manuál a prezentační či vysvětlující videa.

Kapitola 2

Popis mechanik hry a inspirace

Tato kapitola představuje žánr hry real-time strategie a popisuje základní principy. Nedílnou součástí jsou úvahy o vybalancování hry a strategických mechanikách, které jsou ve výsledné hře implementovány.

2.1 Inspirace

Projekt je inspirován řadou real-time strategií (RTS) aneb strategií v reálném čase – jmenovitě například Starcraft [24], Command & Conquer [25], Age of Empires 2 [26] – a to spíše starším verzím těchto her nebo jejich remasterovaných[1] verzí. Vzhledem k cílům práce a ambicióznosti projektu se uvedeným hrám nesnaží přímo konkurovat, ale přiblížit v základních aspektech – práce tedy neobsahuje detailní konkurenční srovnání. V této kapitole jsou popsány nejdůležitější principy, na kterých hra stojí.

2.2 Základní herní princip

Žánr hry real-time strategie [23] spočívá ve stavbě základny, zajištění přísunu surovin a výroby obranných nebo útočných jednotek. Hra snese vzdálené srovnání s deskovými strategickými hrami nebo tahovými počítačovými strategickými hrami se zásadním rozdílem, že strategické prvky jsou zjednodušeny a hráč musí jednotlivá rozhodnutí provádět v reálném čase. Výsledek hry tedy nezávisí pouze na strategických rozhodnutí, ale také na postřehu a reflexech hráče. Hra v reálném čase je zábavnější než tahová a také její hraní je dostupnější pro širší okruh hráčů.

Představení obsahu hry vychází z vlastního pozorování a testování her daného žánru. Jsou vybrány základní klíčové prvky, které jsou v rámci těchto her sjednocující a podobné. Obsah může být pestřejší a rozsáhlejší, to nicméně závisí na důkladném herním designu, který tato práce předkládá v základní podobě. Důležitým aspektem je také časová zvládnutelnost – projekt je jinak navržen tak, že ho lze dále rozvíjet o další herní obsah a principy. Těmito základními herními prvky jsou:

- herní prostředí – mapa (tráva, voda, lesy),
- surovinové pole k těžení zdrojů,

[1] Pojem, který ve slangu hráčů počítačových her znamená novou verzi původní hry, která má novou grafiku s vysokým rozlišením, ale samotná hra v principu zůstává stejná.

- základna hráče skládající se z budov různých účelů (rafinerie, továrna apod.),
- vozidlo pro těžbu surovin,
- pěší jednotky a mechanizované jednotky,
- obranné věže.

Cíle hry mohou být různé – např. zabrání větší části mapy, ubránění území, obsazení a udržení strategických bodů nebo přímočará kompletní porážka protivníka – tedy zlikvidování jeho základny a jednotek. Potenciál hry otevírá různé možnosti, nicméně tato verze projektu předpokládá jeden cíl hry ve formě porážky všech protivníků.

2.3 Vlastnosti herních jednotek

Hráč výměnou za natěžené zdroje staví jednotky z továren nebo dalších budov. Každá jednotka určitého typu má následující vlastnosti:

- cena za výrobu a čas výroby,
- pěší jednotka (rozměr 1*1, rychlost chůze),
- mechanizovaná jednotka (rozměr 2*2, rychlost a akcelerace, rychlost otáčení podstavce, rychlost otáčení palebné věže, možnost střelby za pohybu: ano/ne),
- typ střely a počet střel v zásobníku a prodleva mezi střelami (dostřel, síla poškození, instantní zásah nebo projektil, bodové nebo plošné poškození),
- obrana (míra odolnosti, obrana proti typům střel – kladná nebo záporná).

2.4 Princip vybalancování

Rychlá úvaha by mohla vést k závěru, že ve hře zvítězí ten, kdo co nejrychleji natěží nejvíce surovin a postaví co nejvíce nejsilnějších jednotek – v takovém případě by se ale jednalo spíše o akční hru (ačkoliv akční prvek je také důležitou součástí). Aby hra dostala strategický rozměr, je potřeba jednotlivé prvky hry nějakým způsobem vybalancovat, aby nebylo možné se držet jediné výherní strategie.

Konkrétní princip vybalancování podobných her není veřejně dokumentován, nicméně zákonitosti lze odhadnout vlastním pozorováním – práce tedy vychází ze zkušeností s řadou her tohoto typu.

Pro návrh funkčního principu vybalancování je nutné herní stavový prostor zjednodušit do matematického modelu [27]. Návrh modelu v této práci stojí na základním předpokladu, že hráči disponují zdrojem surovin, jehož výše je přímo úměrná času stráveným ve hře. Zdroje surovin jsou směnitelné za budovy, nebo navýšení těžby, nicméně pro model je klíčová směna za jednotky disponující palebnou silou, které slouží pro konfrontaci s protihráčem. Navržený model pracuje s parametrem palebné síly jednotky *strength*, odolnosti *endurance* a ceny jednotky *cost*:

$$strength = \frac{50 * (shot\_count * damage)}{shot\_reload\_delay * shot\_count + reload\_delay} \tag{2.1}$$

$$endurance = \frac{health * ramor}{strength} \tag{2.2}$$

$$cost = strength * health * k \tag{2.3}$$

Palebná síla udává míru způsobeného poškození během jedné sekundy (konstanta *50* představuje počet herních logických sekvencí během jedné sekundy a další proměnné určují počty střel a prodlevy). Míra odolnosti poté představuje počet sekund do úplného zničení. Cena jednotky je úměrná síle a odolnosti s nějakou další korekcí (konstanta k) vycházející z testů reálného chování celé hry.

Podstatná je proměnná obrany *armor*, která je rozdílná v závislosti na typu střely a jednotky. Jednotky hráče tedy nejsou rozlišeny pouze větší silou a větší výdrží, ale zachovávají mezi sebou princip kámen – nůžky – papír. Příklad:

Obrněné vozidlo O: $strength_O = 100, health_O = 500, armor_{O,T} = 0.5, armor_{O,P} = 2$

$$endurance_{O,T} = (500 * 0.5)/300 = 0.83s \tag{2.4}$$
$$endurance_{O,P} = (500 * 2)/150 = 6.6s \tag{2.5}$$

Lehký tank T: $strength_T = 300, health_T = 1000, armor_{T,O} = 2, armor_{T,P} = 0.5$

$$endurance_{T,O} = (1000 * 2)/100 = 20s \tag{2.6}$$
$$endurance_{T,P} = (1000 * 0.5)/150 = 3.33s \tag{2.7}$$

Protitankové vozidlo P: $strength_P = 150, health_P = 600, armor_{P,T} = 3, armor_{P,O} = 0.4$

$$endurance_{P,T} = (600 * 3)/300 = 6s \tag{2.8}$$
$$endurance_{P,O} = (600 * 0.4)/100 = 2.4s \tag{2.9}$$

Z příkladu je vidět, že obrněné vozidlo porazí protitankové, ale je snadno zničeno tankem. Na těchto principech je postavena podstata modelu vybalancování hry a způsob, jak mezi sebou hráči provádějí výměnu. V případě, že první hráč bude pouze produkovat jednotky tanku proti protihráči, který produkuje protitanková vozidla, tak dojdou zdroje mnohem dříve a prohraje.

Konečné vybalancování nicméně závisí i na dalších aspektech, jako dostřel, rychlost pohybu, plošné poškození – takže pro finální nastavení je také potřeba určité experimentování a testování.

2.5 Strategické interakce herních jednotek

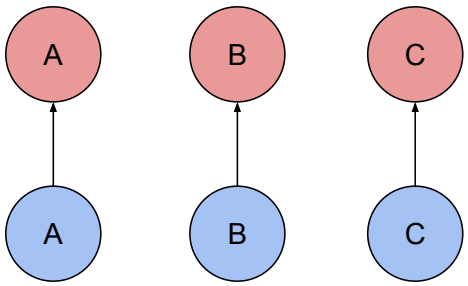
Hraní hry nespoléhá pouze na produkci odpovídajících jednotek, ale také na využití aspektu hry v reálném čase a tedy postřehu a reflexů hráče.

Pro úspěšnou strategii potřebuje hráč reagovat na preference jednotek protihráče a zvažovat vhodnou investici zdrojů v daný okamžik. V případě, že dojde ke střetu s protihráčem, pak má hráč různé možnosti reakce. Několik základních typů reakcí je popsáno v této části.

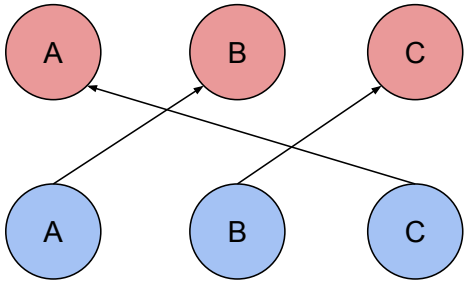
Následující příklady předpokládají jednotky A, B, C s cyklickými výhodami, kde A má výhodu proti B, B má výhodu proti C a jednotka C má výhodu proti A.

▶ **Příklad 2.1.** Neefektivní rozhodnutí (obrázek 2.1) – jednotky útočí na nepřátelské bez využití výhody – důsledkem je poté časová ztráta nebo prohra bitvy v případě, že oponent útočí s využitím bonusů.

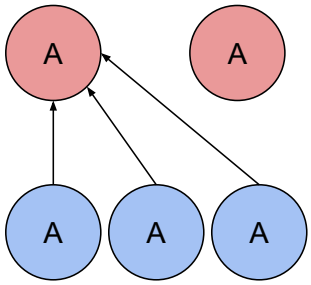
▶ **Příklad 2.2.** Efektivní rozhodnutí (obrázek 2.2) – jednotky útočí s využitím výhody – může dojít k vítězství v bitvě, nebo znemožnění oponentovi, aby bonus využil na jednotky v záloze.



Obrázek 2.1 Schéma neefektivního rozhodnutí



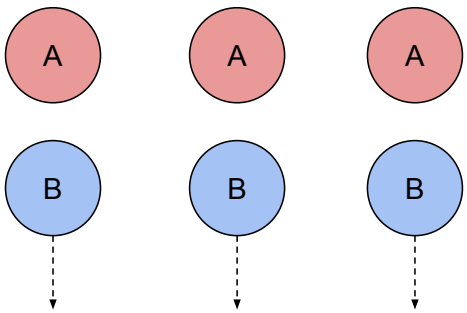
Obrázek 2.2 Schéma efektivního rozhodnutí



Obrázek 2.3 Schéma postupné eliminace

▶ **Příklad 2.3.** Postupná eliminace (obrázek 2.3) – pokud jednotky proti jednotkám oponenta nedisponují žádnou výhodou, může být vhodnou strategií postupná eliminace – rychlé zničení první jednotky zredukuje oponentovu celkovou palebnou sílu.

▶ **Příklad 2.4.** Ústup (obrázek 2.4) – v případě, že jednotky nemají žádnou výhodu a naopak oponentovy jednotky mají bonus maximální, pak jde o zcela nevýhodnou výměnu a v konečném důsledku plýtvání zdrojů – je tedy lepší se stáhnout.



Obrázek 2.4 Schéma ústupu

2.6 Stavba základny a těžba zdrojů

Nezbytnou součástí hry je stavba základny hráče, která umožňuje těžbu surovin, výrobu jednotek a obranu. Stavba budovy vyžaduje investici surovin a časový interval výstavby. Ve hře jsou dostupné následující budovy:

- **Hlavní budova** – umožňuje výstavbu dalších budov. Pokud je tato budova zničena, tak je výstavba dalších budov znemožněna.
- **Rafinerie** – budova, která umožňuje těžbu surovin. Společně s budovou je vytvořeno autonomní vozidlo realizující těžbu. Více postavených budov rafinerie vede k navýšení přísunu surovin.
- **Kasárna** – budova pro výrobu pěších jednotek.
- **Továrna** – budova pro výrobu mechanizovaných jednotek.
- **Laboratoř** – budova umožňující zvýšení technologické úrovně budov. Budova s vyšší technologickou úrovní umožňuje výrobu pokročilejších jednotek.
- **Obranná věž** – základní obranná věž a pokročilá věž.

Případný další vývoj hry předpokládá větší rozvoj této části a konkrétní integraci do modelu vybalancování. Aktuální verze projektu předpokládá pouze tento základní návrh základny hráče, jehož funkčnost vychází z dílčího experimentování a testování.

Kapitola 3
Vývojové prostředí

V kapitole je popsáno zvolené vývojové prostředí a programovací jazyk s představením výhod a nevýhod. Jsou demonstrovány základní principy optimalizací.

3.1 Jazyk Python a jeho výhody

Pro vývoj je zvolen jazyk Python [3]. Zásadní výhodou jazyka Python je rychlý a snadný vývoj, rychlý návrh a testování algoritmů a tedy rychlá výroba funkčního prototypu hry. Python je vysokoúrovňový jazyk s automatickou správou paměti a počítáním referencí instancí a souvisejícím garbage-collectorem[1]. Podporuje různá programovací paradigmata, jak objektový přístup tak funkcionální. Tyto přístupy je možné kombinovat podle vhodnosti řešení konkrétního problému. Jazyk je multiplatformní s poměrně snadnou přenositelností běhového prostředí. Python je dobře zdokumentovaný včetně nízko-úrovňových částí a tak umožňuje poměrně snadnou výrobu dalších rozšíření v C/C++ pro případné optimalizace. Dalším přínosem je vysoká popularita jazyka, která vede k dostupnosti užitečných nástrojů a knihoven nebo snazší zapojení dalších programátorů pro kooperaci na vývoji.

3.1.1 Nevýhody jazyka Python

Rychlý vývoj v jazyce Python nese daň v podobě poměrně nízkého výkonu. Jedná se o interpretovatný typově slabý jazyk a ani přímá kompilace problém rychlosti nesmazává. Dalším nemalým problémem je vlastnost jazyka nazvaná GIL [4] (Global Interpreter Lock), která znemožňuje plnohodnotné použití vláken. Ačkoliv vlákna jsou podporována, tak opatření GIL zabraňuje interpreteru jazyka, aby běžel paralelně. Jde o zásadní problém, který znemožňuje výkon programu efektivně rozložit do vláken – nicméně obejít tento problém umožňují další nástroje pro optimalizaci kódu (viz. 3.2).

3.2 Optimalizace pomocí nástroje Cython

Jednou z prvních možností, jak optimalizovat kód jazyka Python, je použití alternativního interpretru, jako který se nabízí PyPy [8]. Autoři tvrdí, že kód provádí just-in-time kompilaci a je až 4,5 krát rychlejší, než kód nativní implementace interpretru CPython. Bohužel, po provedení experimentů se ukázalo, že kód hry je naopak 2 krát pomalejší – pravděpodobně z důvodu použití OpenGL API.

[1] Automatický mechanismus, který zajišťuje uvolnění paměti instancí objektů, které se již nepoužívají.

Výpis kódu 3.1 Ukázka alokace pole

```
cdef struct PathFlowMapField:
        long flag
        long cost[2]
        int finish
        long steps

cdef size_t a_ptr = <size_t>malloc(width * height * sizeof(PathFlowMapField))
```

Výpis kódu 3.2 Celočíselná odmocnina [7] v Cythonu

```
cpdef long int_sqrt(long value):
    cdef long long left = 0
    cdef long long right = value + 1
    cdef long long mid

    while left != right - 1:
        mid = (left + right) // 2

        if mid * mid <= value:
            left = mid
        else:
            right = mid

    return left
```

Dalším zajímavým nástrojem je projekt Numba [9], který umožňuje optimalizovat konkrétní bloky kódu – bohužel se nástroj nepodařilo zprovoznit.

Zdá se tedy lepším přístupem nespoléhat na zázračná magická řešení a optimalizace raději provádět přímo pomocí nástroje Cython [5]. Jde o nástroj, který ze zdrojového kódu jazyka Python sestaví kód jazyka C/C++, ze kterého poté vytvoří kompilací funkční modul. Moduly Cython jsou z prostředí jazyka Python dostupné a plně kompatibilní.

Cython vedle samotné kompilace obsahuje jazykovou nadstavbu, pomocí které je možné psát de-facto kód jazyka C/C++. Tato možnost je pro optimalizaci velice zásadní, protože umožňuje postupy, které jsou v samotném jazyce Python nemožné. Jako podstatné je třeba zmínit alokace polí datových struktur 3.1, díky kterému je mimo jiné možné opustit mezi programátory oblíbený moloch Numpy, který v klasickém prostředí jazyka Python slouží pro efektivní a rychlou práci s poli a maticemi (knihovna Numpy představuje balíček o velikosti 100 MB a její použití v rámci nástroje Cython je komplikované a většina funkcionalit posléze nevyužitá).

Nezbytnou vlastností je možnost kompilace optimalizovaného Cython kódu do čistého C/C++. Příklad optimalizovaného Cython kódu 3.2 – výsledný kód v C/C++ sice není hezky čitelný, nicméně je efektivní. Kompilace neoptimalizovaného kódu obsahuje spousty kontrol a ošetření stavů interpretru a je řádově pomalejší.

Tyto optimalizace nicméně nelze provádět automaticky a je nutné tomu podřídit návrh celého projektu. Klíčovým optimalizačním prvkem je klauzule *cdef*, která zjednodušeným pohledem značí deklarace čistého C/C++ prosté od běhových kontrol jazyka Python. Výkonové problémy nastávají až při volání optimalizovaných funkcí z jazyka Python a předávání či navracení parametrů. Nejzásadnějším těžko řešitelným optimalizačním problémem jsou rozsáhlé kolekce objektů přímo v prostředí jazyka Python – vlastnosti a metody těchto objektů nejsou žádným jednoduchým způsobem dostupné z prostředí C/C++ např. pomocí ukazatele. Další problematický

důsledek optimalizací je skutečnost, že časté optimalizace smazávají výhodu rychlého psaní Python kódu.

Poslední velmi důležitou možností nástroje Cython je klauzule *nogil*, která umožňuje v rámci bloku kódu vypnout vlastnost GIL a využít výhody paralelního běhu vláken. Tento projekt tuto možnost prozatím nevyužívá a klauzule je zde zmíněna pro případ dalšího vývoje.

Možností optimalizace, které nástroj Cython poskytuje, je mnohem více – v práci jsou nicméně zmíněny především ty zásadní.

3.3 Pyglet a OpenGL

Projekt využívá knihovnu Pyglet [10] pro 2D grafiku. Jde o multiplatformní grafickou knihovnu postavenou nad OpenGL poskytující především základní funkcionalitu. Konkrétně inicializaci aplikačního okna pro vykreslení grafiky, načítání grafických zdrojů, kompletování atlas-map, systém spritů a efektivní komponování 2D mřížky pro dávkové vykreslování. Podporovány jsou grafická primitiva (linky, obdélníky apod.) a texty. Knihovna také podporuje přehrávání zvuků. Není podporováno uživatelské grafické rozhraní (GUI) – z této části je dostupné pouze editovatelné políčko. Pro uživatelské rozhraní je vytvořeno vlastní řešení (viz. kapitola 11.1.1). Společně s rozšířením PyShaders knihovna umožňuje a výrobu vlastních shaderů[2] a nasazení v OpenGL.

Nabízely se také knihovny Pygame nebo Godot, nicméně tyto knihovny jsou zbytečně komplexní a příliš odstiňují od samotného přístupu k programovaní grafické aplikace v OpenGL. Knihovna pyglet je tedy ideální s dostatečnou sadou funkcionalit.

[2]Program, který řídí detaily vykreslování v rámci jednotlivých fází procesů na grafické kartě.

Kapitola 4
Analýza a návrh

Kapitola popisuje sběr požadavků nezbytných pro kompletní funkční prototyp hry. Demonstruje základní návrh struktury aplikace a související problémy.

4.1 Funkční a nefunkční požadavky

Funkční herní požadavky:

- **F1**: reprezentace mapy a prostředí,
- **F2**: skrývání neodkrytých částí mapy mlhou,
- **F3**: stavba budov,
- **F4**: těžba surovin,
- **F5**: pohyb objektů jednotek,
- **F6**: výroba jednotek (v továrně výměnou za zdroje),
- **F7**: interakce jednotek (střelba, likvidace),
- **F8**: obranné věže,
- **F9**: vybalancování hry,
- **F10**: více hráčů na scéně a obarvení,
- **F11**: generátor map,
- **F12**: umělá inteligence (skriptována),
- **F13**: hra jednoho hráče proti AI (singleplayer),
- **F14**: hra více hráčů po síti (multiplayer).

Funkční požadavky ovládání:

- **F15**: menu hry,
- **F16**: menu nastavení,
- **F17**: menu spuštění nové hry,

- **F18**: menu hry více hráčů,
- **F19**: rozhraní pro ovládání hry (příkazy jednotkám, stavba atd.),
- **F20**: možnost herního chatu,
- **F21**: zobrazení mini-mapy.

Další funkční požadavky:

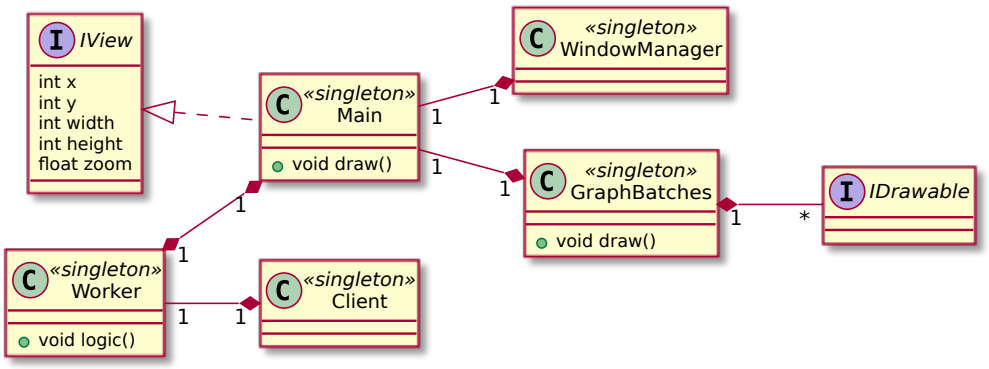
- **F22**: server pro hru více hráčů.

Nefunkční požadavky:

- **N1**: výkonové požadavky (optimalizace),
- **N2**: požadavek grafiky a vzhledu,
- **N3**: požadavek hratelnosti,
- **N4**: možnost přiblížení (zoom).

4.2 Základní doménový model

Aplikace je postavena s využitím principů OOP. Hlavním předpokladem je důsledné oddělení vykreslování a herní logiky. Program *Main* provádí inicializaci okna, poskytuje informaci o aktuálním viewportu a předává uživatelské vstupy jednotlivým komponentám. V rámci svého vlákna zajišťuje vykreslování přes *GraphBatches*. Třída *Worker* obstarává logiku spuštěné hry – v rámci samostatného vlákna. Návrh základní struktury znázorněn na obrázku 4.1.



Obrázek 4.1 Základní doménový model aplikace

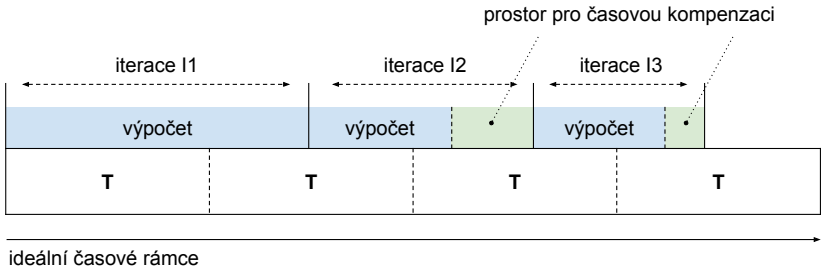
Celá aplikace je poměrně rozsáhlá (zhruba 150 tříd) – další doménové modely dílčích částí v základní podobě jsou znázorněny v jednotlivých kapitolách.

4.3 Herní smyčka a časování

Logika hry je rozdělena do více vláken a tedy více herních smyček – první smyčka obstarává vykreslování a je buď neomezená, tedy vykreslí maximum snímků, co zvládne GPU[1], nebo je

[1] Procesor na grafické kartě.

omezena pomocí vertikální synchronizace[2] (kterou je možné zapnout v nastavení). Druhá smyčka obstarává výpočty herní logiky, u které je z důvodu plynulosti důležité přesné časování – smyčka provádí přesně 50 iterací za sekundu a je klíčové, aby běh byl co nejvíce přesný. Naráží to na dva problémy a to rozdílné složitosti výpočtů mezi jednotlivými iteracemi a nepřesnost uspání vlákna. Tyto nepřesnosti je potřeba kompenzovat, jak je demonstrováno na obrázku 4.2.



Obrázek 4.2 Časové rámce iterací a kompenzace

Pro zajištění přesného časování 50 iterací za sekundu je použita metoda *sleep* – tedy uspání vlákna na krátký časový okamžik pro zamezení aktivního čekání. Funkce *sleep* sama o sobě vykazuje problém nepřesného časování, jehož minimální délka se liší v závislosti na OS[3]. Pro zachování přesného časování se udržuje průběžný průměr minimální doby uspání *min_sleep* a počet těchto uspání se spočítá: $sleeps\_in\_frame = (1/50)/min\_sleep$. Každá jednotlivá iterace provede celou část $sleeps\_in\_frame - (calc\_time/min\_sleep)$ jednotlivých uspání, kde *calc_time* je čas výpočtu v rámci poslední iterace a zbylou desetinnou část ponechá pro následující iteraci. Tímto principem je zajištěno, že se drží stabilně 50 iterací za sekundu nezávisle na OS.

Implementace více vláken se také potýká s problémem GIL a tedy implicitní nemožností paralelního běhu (jak je popsáno v kapitole 3.2). Tento problém se ale netýká návrhu aplikace v podobě multi-vláknové architektury. Případný další vývoj hry si klade za cíl výkonnostní problém odstranit pomocí klauzule *nogil* – současný prototyp to zatím neřeší.

[2] Synchronizace obnovovací frekvence monitoru s grafickou kartou.

[3] Zdroj k tomuto tvrzení není uveden, dohledat lze pouze příspěvky diskuzních fór – interval vychází z experimentů: min. 15 ms pro Windows a 1 ms pro Linux.

Kapitola 5
Grafické prostředí

Pro celkový dobrý dojem ze hry je důležitá konzistence grafického prostředí a základní smysl pro detail. Hra, která má sice jednoduchou grafiku, tak pokud je ucelená, stejného stylu a přehledná, tak může působit lépe, než překombinovaná grafika složená z 3D modelů využívající nejmodernějších technologií.

5.1 Doménový model mapy a grafiky

Vykreslování obstarává třída *GraphBatches*, která udržuje objekty k vykreslení implementující *IDrawable* a zajišťuje jejich vykreslení v rámci nastavených vrstev. Další třídy *RectCollector*, *EnvCollector*, *GroundCollector* zajišťují struktury mřížek[1] pro vykreslení efektů tříd *ParticleItem* a *Light*. Vykreslení herního prostoru mapy obstarává *GridArea* – pro kterou logickou strukturu udržuje *MapBase* včetně detailů políček *MapTile* a *MapPlace*. Třída *MapPlace* přes rozhraní *ITileCollector* poskytuje přístup k informacím mapy, které používají další třídy pro např. generování nebo skenování mapy. Model je znázorněn na obrázku 5.1.

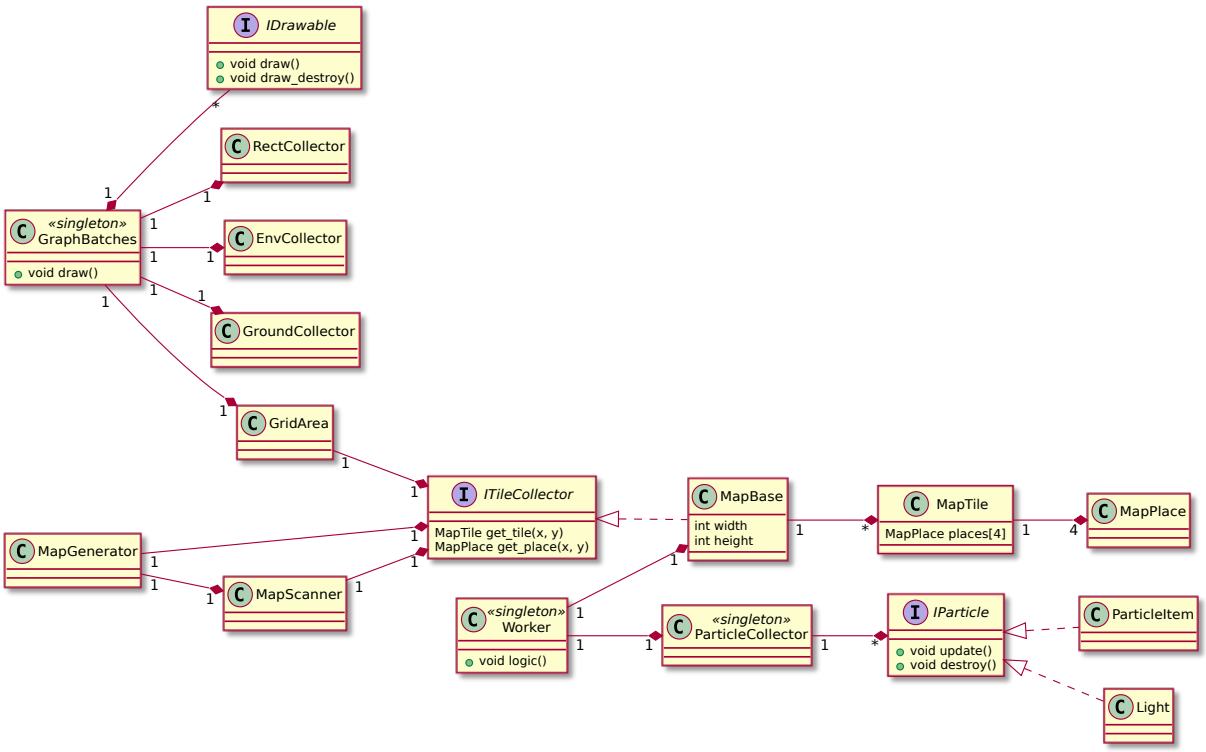
5.2 Herní prostor a mapa

Herní prostor je koncipován jako 2D plocha viděná ze shora. Hlavním přínosem tohoto přístupu je relativní jednoduchost výroby grafických podkladů. Nelze opomenout ani přehlednost takto zjednodušeného herního světa.

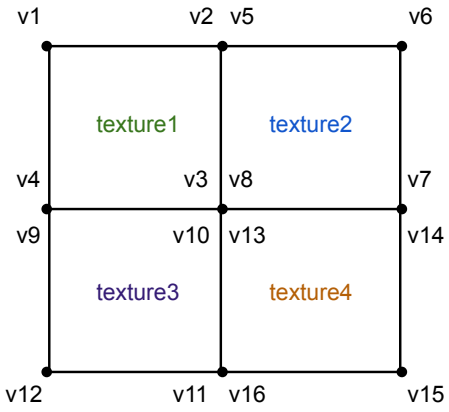
Základem je herní mapa skládající se z čtvercové mřížky, kde jsou jednotlivá políčka rozdělena na neprostupnou část (les, voda) a prostupnou (tráva, podklad). Logická struktura mapy každé políčko dělí na dalších 2*2 pod-políček, které mají význam pro pohyblivé objekty (jednotky).

Vykreslování mapy nepoužívá sprity knihovny Pyglet, ale vlastní vertexovou strukturu mřížky ze čtverců vykreslovanou pomocí GL_QUADS [11] jak je znázorněno na obrázku 5.2. Výhoda tohoto přístupu tkví ve vyšší rychlosti – v případě, že probíhá pouze přesun po mapě (nikoliv přiblížení), tak dochází pouze k posunu celé mřížky. Při posunu o celé políčko dojde k překonfigurování koordinátu textur (UV coordinates). Výhoda neplatí pro přiblížení (zoom) – v takovém případě je nutné přestavět celou mřížku mapy. Další výhodou tohoto přístupu je možnost použití vlastních shaderů.

[1] Mřížka aneb mesh je polygonová síť složená z vrcholů a ploch, která tvoří objekt v prostoru.



Obrázek 5.1 Doménový model mapy a grafiky



Obrázek 5.2 Mřížka herní mapy

5.2.1 Mlha neodkrytých oblastí

Důležitou součástí hry je skrývání neobsazených částí mapy, aby nebylo možné snadno odhalit plány a postup oponenta. Tato část mapy je zahalena mlhou. V rámci počítačových her se pro tuto vlastnost používá termín *fog of war*.

Mlha se skládá z částí, které jsou zcela neviditelné a zakrývají vše černou barvou a nebo částečně průhledné na pomezí viditelnosti a neviditelnosti. Tato možnost je řešena další mřížkou, která překrývá herní mapu a funguje na stejném principu – všechna políčka jsou ale černá s vari-

abilní průhledností rohů políčka. Zde vzniká problém, že GPU interně převádí GL_QUADS na trojúhelníky, u kterých neprobíhá korektní rovnoměrné rozložení průhlednosti v rámci čtverce (např. jeden roh průhedný, zbylé černé). Tento problém je vyřešen interpolací pomocí barycentrických souřadnic [2].

5.2.2 Generátor mapy

Přípravu mapy zajišťuje generátor se základními vstupy počtu hráčů, velikostí a inicializačním číslem náhodného generátoru – tzv. číslem *seed*. Generátor funguje v následujících krocích:

Příprava mapy – vyplnění prázdné mapy neprostupnými políčky vody a označení jako nenavštívené.

Rozmístění pozic hráčů – v rámci kruhu se středem ve středu mapy proběhne rovnoměrné rozdělení pozic hráčů (3 hráči pod úhlem 360/3 apod.).

Generování základních oblastí – na pozici hráče proběhne generování n políček trávy a $2 * q$ políček zdrojů k těžení. Generování probíhá upraveným algoritmem BFS[2] s určitým aspektem náhody při procházení prostoru. Již obsazené políčka se nepřekrývají.

Obtažení hran – vygenerované pozice hráčů se obtáhnou několika vrstvami trávy. Obtažení probíhá taktéž algoritmem BFS, který nepokračuje v místě, kde najde nenavštívené políčko (vyplní ho trávou a nevloží do fronty).

Spojení se středem – pozice hráčů se spojí tučnou linkou trávy do středu. Vyplňovány jsou opět nenavštívená políčka.

Generování zbytku trávy – doposud nenavštívená políčka se vyplní trávou s pravděpodobností 55 %. Nad tímto stavem mapy se spustí dvě iterace jednoduchého celulárního automatu [6], který u každého políčka, které má 5 a více okolních políček vodu, nastaví vodu a jinak trávu (z procesu jsou vyloučeny oblasti, které již jsou vytvořeny předchozími fázemi generátoru). Výsledek generuje poměrně obstojné ostrovní struktury. Jiná konstanta než 5 nevede k dobrému výsledku.

Generování lesů – podobným způsobem dojde k vytvoření lesů – náhodný šum stromů s pravděpodobností 50% se také zpracuje podobným celulárním automatem. Vynechány jsou již vytvořené pozice hráčů a spojnice do středu mapy, aby nedošlo k vytvoření neprostupného prostředí.

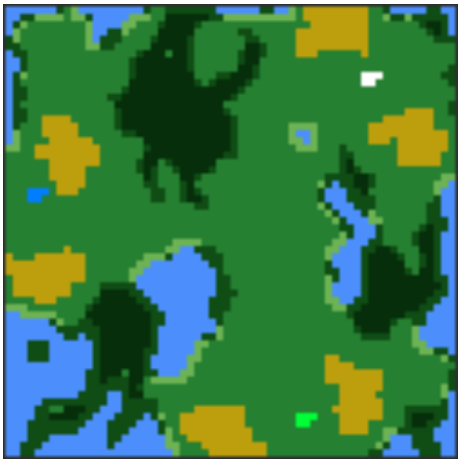
Finální úpravy – doplnění kosmetických změn, jako hlubokých velkých stromů uvnitř lesa a rozdílné textury na předělu trávy a vody.

Ukázkový výstup generátoru je znázorněn na obrázku 5.3 – zeleně je tráva, tmavě zeleně stromy a lesy, modře voda a oranžově zdroje k těžbě.

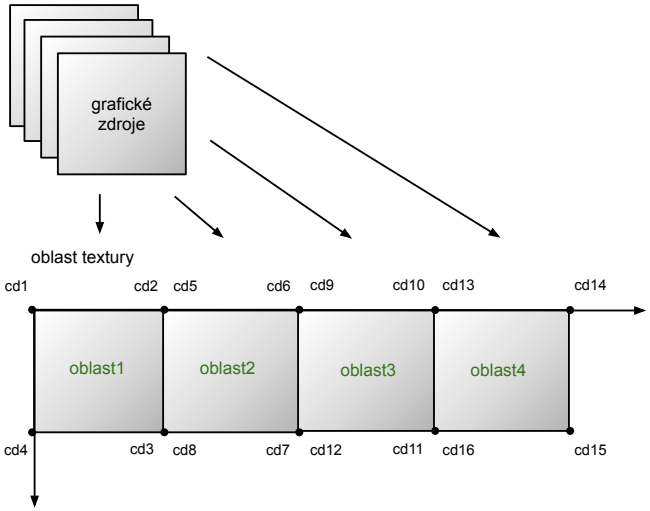
5.3 Grafické objekty

Ostatní grafické objekty jsou realizovány pomocí nástrojů knihovny Pyglet. Ta přes srozumitelné API realizuje správu mřížek pro jednotlivé vrstvy objektů a automatickou kompletaci alokovaných grafických zdrojů do textury ve formě atlas-mapy, která představuje texturu s efektivním rozmístěním jednotlivých spritů[3], jak je znázorněno na obrázku 5.4.

[2] Algoritmus prohledávání do šířky [33].
[3] Sprite – malý dvourozměrný obrázek, který je v podobě objektu umístěn v prostoru hry.



Obrázek 5.3 Ukázka generované mapy pro 3 hráče



Obrázek 5.4 Kompletace atlas mapy

Nad grafickými zdroji pomocí přímé modifikace binárních obrazových dat jsou vytvářeny následující varianty textur:

Konverze černobílých variant – verze obrázků pro zničené jednotky a budovy.

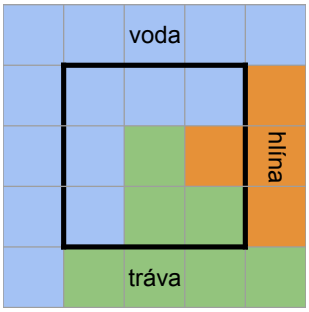
Konverze barevných variant – změna barvy pixelů modifikací složky Hue v prostoru HSV[4] (výběr pixelů probíhá podle masky).

Přidání okraje – texturám mapy se přidává 1px okraj se zkopírovanou vrstvou pixelů pro zamezení artefaktů při zvětšování nebo zmenšování mřížky vertexů.

Překryvové kombinace textur – pro políčko textury se v závislosti na okolních texturách sestavuje kombinovaná textura, rozdělená na části 3*3 s principem prolnutí – který je v této verzi jednoduchý pixelový šum. Znázorněno na obrázku 5.5.

[4]Barevný prostor skládající se ze složek odstínu, sytosti a jasu.

Zobrazení mini-mapy – mini-mapa herní plochy se provádí aktualizací jednotlivých pixelů textury s barevným rozlišením stromů, vody, budov, jednotek apod.

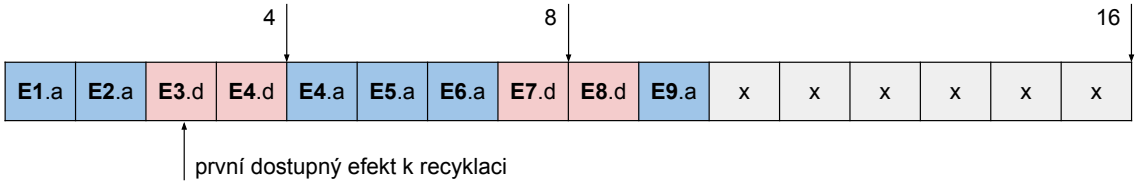


Obrázek 5.5 Příklad překryvové textury

5.4 Částicové efekty a světla

Potřeba grafických efektů souvisí s odměňování hráče za dosažení dílčích cílů – to může být výbuch tanku, efekt výstřelu více jednotek v jeden okamžik apod., ideálně doprovázené vhodnými zvukovými efekty. Vhodnou kombinací efektů se vytváří podvědomá odměna hráče, který je pak více motivován ve hře pokračovat a zkoušet plnit další cíle.

Knihovna Pyglet nativně nepodporuje žádný systém efektů a světel. Ve hře je proto implementován vlastní podsystém, postavený na správě mřížky složené taktéž z GL_QUADS. Objekty efektů mají implementovány rozšířené transformace, jako je konstrukce texturované linky mezi body (A, B) s proměnlivou šířkou, možnosti nezávislého pohybu všech čtyř bodů objektu a obarvování a průhlednost za pomocí vlastního shaderu. Vzhledem k tomu, že je výpočetně náročnější kompletní rekonstrukce celé mřížky, než jeho vykreslení, je potom jeho údržba založena na principu nafukovacího pole, jak je znázorněno na obrázku 5.6.



Obrázek 5.6 Mřížka s kolekcí efektů v podobě nafukovacího pole

Objekty efektů se po skončení animace přepnou ze stavu *active* do stavu *discarded* – ve kterém objekt stále figuruje v rámci struktury mřížky, ale má maximální průhlednost. Při vytvoření nového efektu se zahozený objekt recykluje a nastaví se mu nové parametry. Při delší ne-aktivitě probíhá přestavba celé mřížky – pole se zredukuje na velikost aktivních efektů, nebo kompletně promaže, pokud efekty nejsou žádné.

V projektu se používá několik druhů efektů:

Světelné efekty – světelné záblesky jsou řešeny objektem s kruhovou texturou s animací expanze a kontrakce a změnou barvy nebo průhlednosti.

Efekty střel – texturovaná obdélníková linka ze dvou bodů s proměnlivou šířkou a animovaným začátkem a koncem.

Objekty prostředí – ve hře jsou použity statické objekty kmenů stromů a dílčích korun stromů s efektem větru.

Stopy vozidel – jízda vozidla po trávě za sebou zanechává stopy v podobě průhledných texturovaných obdélníků, které na sebe průběžně navazují (vozidla jsou tímto efektem lépe zasazeny do prostředí hry).

Animované sekvence – efekty, které se skládají ze sekvence animovaných obrázků (kouř, výbuch) – jsou řešeny klasickými sprity knihovny Pyglet.

5.5 Výroba grafiky

Veškerá grafika ve hře je autorským dílem. Byla vytvořená metodou pixel-art pomocí softwaru Aseprite. Demonstrace nakreslených budov a jednotek je na obrázku 5.7.

Obrázek 5.7 Přehled grafických spritů základen a jednotek a některých efektů

Kapitola 6

Herní objekty a interakce

Zde jsou popsány typy herních objektů v podobě jednotek a budov a jejich možné vlastnosti, které jsou v prototypu hry implementovány. Je znázorněno, jak mezi sebou objekty interagují a jak se v herním prostoru hledají. Důležitou součástí je, jak hráč objekty ovládá a jaké příkazy může zadávat.

6.1 Doménový model herních objektů

Herní objekty se dělí na jednotky, budovy a střely. Jednotky obsluhuje třída *MoveableCollector* a objekt jednotky *MoveableItem* – který používá několik vlastností: *MoveablePath* pro pohyb, *MoveableTarget* pro cílení. Další třídy *MoveableMiner*, *MoveableBuilder* a *MoveableUpgrade* obsahují logiku chování speciálních vlastností (těžič surovin, stavební jednotka, technologická jednotka). Budovu představuje třída *BuildItem* s obsluhující *BuildCollector*. Budova může spravovat několik obranných věží *BuildTurret*. Třída *ShotBase* představuje objekt střely, ze které dědí třídy s konkrétními vlastnostmi střel. Třída *ShotSplash* realizuje plošné poškození. Model znázorněn na obrázku 6.1.

6.2 Objekty jednotek a budov

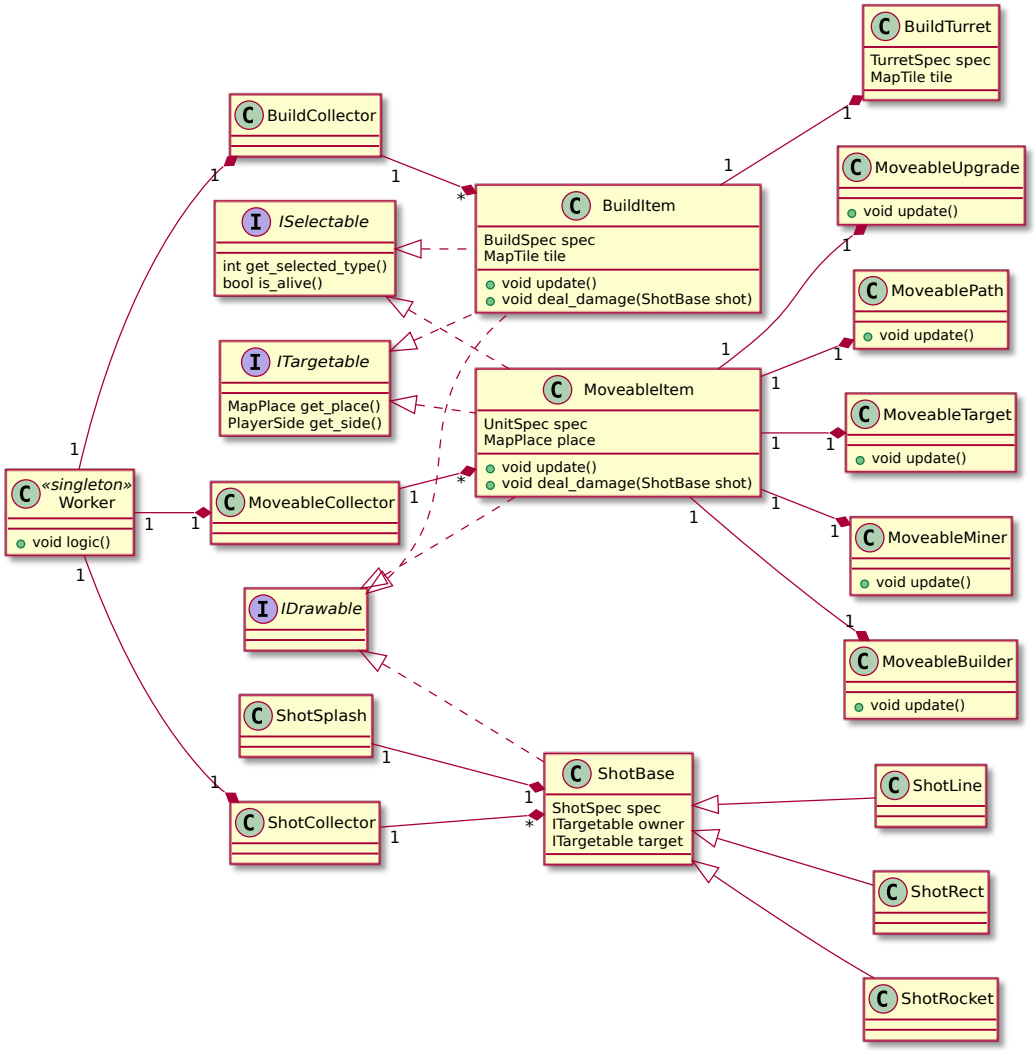
Ve hře jsou rozlišeny dva základní typy objektů, se kterými hráč může přímo interagovat a mají rozhodující vliv na průběh a výsledek hry – a tím jsou budovy a jednotky. Budovy zabírají obdélníkovou část mapy $width * height$ – a skládají se ze spritu podkladu a spritu samotné budovy. Budovy mohou mít na každém políčku postavenou obrannou věž (v současné verzi hry je nastavena pouze jedna věž pro speciální budovy obranných věží rozměru 1*1).

Jednotky jsou o velikosti malých políček 1*1 nebo 2*2 a nabývají konfigurací znázorněných na obrázku 6.2 – **A**: pěší jednotka z jednoho spritu, **B**: mechanizovaná jednotka skládající se z podstavce, věže a děla, **C**: mechanizovaná jednotka z jednoho kusu spritu.

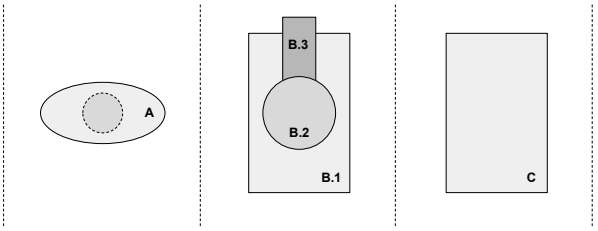
Objekty jednotek nabývají těchto konkrétních vlastností:

Střelba – jednotka analyzuje okolní políčka a v případě dostřelu střílí (funkcionalitu poskytuje *MoveableTarget*).

Těžba surovin – v okolním prostředí hledá zdroje a když je najde, naviguje se na jejich místo a provede těžbu. Poté vyhledá nejbližší budovu rafinerie, ve které suroviny promění ve zdroje (funkcionalitu poskytuje *MoveableMiner*).



Obrázek 6.1 Doménový model herních objektů



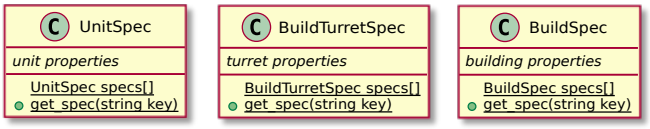
Obrázek 6.2 Konfigurace objektů jednotek

Stavba budovy – jednotka vyjede ze základny a na políčkách cílové budovy provede stavbu a vytvoří objekt budovy. Poté zajede zpátky do základny (funkcionalitu poskytuje *MoveableBuilder*)

Technologické vozidlo – k cílové budově přepraví balíček technologických komponent a poté zajede zpátky do laboratoře (funkcionalitu poskytuje *MoveableUpgrade*).

Návrh programu umožňuje i kombinování těchto vlastností – např. těžební vozidlo, které střílí – nicméně v současné verzi hry tato možnost využita není.

Konkrétní kombinace vzhledu, vlastností a jejich nastavení jsou zahrnuty pod specifikací jednotek. Podobně existují specifikace budov a specifikace obranných věží. Specifikace je možné snadno měnit a rozšiřovat a jsou zahrnuty v rámci katalogů specifikací znázorněných na obrázku 6.3.



Obrázek 6.3 Model specifikací vlastností objektů

6.3 Střely a kolize

Střely jsou objekty, které zprostředkovávají poškození od jednotky, která pálí, na cílovou jednotku oponenta. Střely provází různorodé grafické efekty, nicméně zásadní technické rozlišení je ve způsobu kolize střel:

Instantní kolize – střela po nebo během animace efektu vždy implicitně koliduje a zasáhne cíl.

Trajektorie – případ střely, která představuje objekt putující po trajektorii linky nebo s mechanikou naváděné střely. Kolize probíhá buď na konci trajektorie, nebo při prostupu kolizní oblasti jednotky oponenta v případě volné kolize. Test kolize probíhá pouze na políčkách mapy, kde se vyskytuje střela.

Oblast – při kolizi je poškození rozšířeno na cíle umístěné v rámci okolních políček podle rozsahu a míry poškození (tzv. splash-damage).

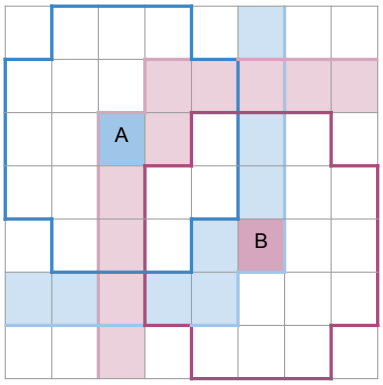
6.4 Detekce objektů

Herní objekty spolu interagují především střelbou (pohyb objektů popsán viz. 7) – pro takovou interakci musí být nejdříve nalezen cílový objekt. Hledání využívá mřížky herní mapy do které veškeré objekty svoje pozice registrují. Střely mohou relevantní políčka najít podle pozice $[\lfloor x/size \rfloor, \lfloor y/size \rfloor]$ kde $size$ je velikost políčka. Na obrázku 6.4 jsou objekty A, B – v rámci tučné linky je znázorněn dostřel objektu a barevná políčka zvýrazňují viditelný dosah.

Probíhá tedy prohledávání pouze omezeného prostoru, které je navíc zredukováno prohledáváním větších políček (políčko mapy se skládá z 2*2 logických políček). Stále zde ale zůstává možnost pro optimalizace na principu quad-tree [12] struktury – tedy vytvoření n úrovní větších políček pro registraci pozic objektů. Současná implementace je nicméně dostatečná a navíc prohledávání probíhá po delších intervalech – nikoliv v rámci každé iterace hry.

6.5 Ovládání jednotek, příkazy a interakce

Ovládání hry je navrženo na použití myši s usnadněním pomocí klávesových zkratek. Klíčovým prvkem je zadávání příkazů jednotkám (přesun, útok) a budovám (stavba, výroba). Hráč pracuje s kontextem výběru objektů, který vytvoří buď kliknutím na objekt, nebo kliknutím a tažením a vytvořením výběrového rámce, který umožní výběr více objektů zároveň. V rámci tohoto kontextu výběru jsou dostupné následující příkazy:



Obrázek 6.4 Hledání vzájemných objektů

Příkaz přesunu – jednotky provedou přesun na cílové místo s rychlostí nejpomalejší jednotky ve skupině. Pokud během přesunu narazí na nepřátelskou jednotku, tak se nezastavují a pokračují v přesunu. Některé typy jednotek s otočnou věží provádějí střelbu během přesunu – stále ale platí, že plní příkaz přesunu na cílové místo a nezastavují se.

Příkaz přesunu s útokem – příkaz je v základu stejný jako přesun s rozdílem, že jednotky analyzují okolní prostředí a pokud objeví nepřátelskou jednotku, tak ji přidají jako cíl přesunu a jakmile se vyskytne v bezprostředním dostřelu, tak se zastaví a střílí. Pokud je nepřátelská jednotka zničena, tak pokračují na původní místo přesunu.

Příkaz cíleného útoku – provádí se kliknutím na nepřátelskou jednotku a je tím iniciován přesun k této jednotce a její zničení. Probíhá útok pouze na vybranou cílovou jednotku.

Příkaz zastavení – označené jednotky okamžitě zastaví jakoukoliv akci přesunu nebo útoku.

Při provedení příkazu *přesun s útokem* jednotky automaticky neprovádí žádné strategické interakce (jak je uvedeno v kapitole 2.5) a pouze útočí na nejbližší nepřátelskou jednotku. Jakékoliv složitější strategické interakce jsou na rozhodnutí hráče kombinováním přesunu, přesunu s útokem a cíleným útokem popř. zastavením.

Vedle těchto interakcí jednotky disponují také automatickým útočením na nepřátelské jednotky ve viditelném dosahu v případě, že nemají zadaný žádný jiný příkaz. Pokud chce hráč tomuto chování zamezit, tak přepne označené jednotky do režimu čekání (*hold*), kdy takto označené jednotky stojí na místě a střílí pouze v rámci bezprostředního dostřelu.

Všechny příkazy počítají s aspektem reálného času – je tedy možné zadávat spousty příkazů různým objektům i v množství desítek příkazů za sekundu. Vykonávání libovolného příkazu je ze strany hráče možné kdykoliv přerušit, upřesnit nebo změnit.

Kapitola 7

Hledání cest skupin objektů a navigace

Objekty ve hře, které představují jednotky hráče, mají definovanou pozici v rámci mřížky herní mapy. Pohyb objektu z místa A na místo B spočívá v sekvenci posunů mezi všemi políčky naplánované cesty. Přesun mezi políčky probíhá plynule s akcelerací, rozdílnou rychlostí a plynulým natáčením. V této kapitole je popsáno, jakým způsobem je vyřešen problém hledání cesty mezi body (A, B) a jak jsou navigovány a jak mezi sebou kolidují.

7.1 Objekty různých rozměrů a navigace po mapě

Objekty mohou nabývat různých čtvercových rozměrů. Použity jsou rozměry 1*1 a 2*2, ale lze implementovat další rozměry. Pozice objektu je definována levým horním rohem. V mřížce je poté pro každé políčko spočítáno, jak maximálně velký objekt se může na políčko umístit, aby nepřekrýval nedostupnou oblast. Číslo představuje nejbližší vzdálenost neprůchozího políčka od pravé nebo dolní strany, jak je znázorněno na obrázku 7.1 – modrý objekt o velikosti 2*2 se může pohybovat pouze po políčkách s cenou 2 a více. Zelený objekt 1*1 se může pohybovat po všech přístupných políčkách.

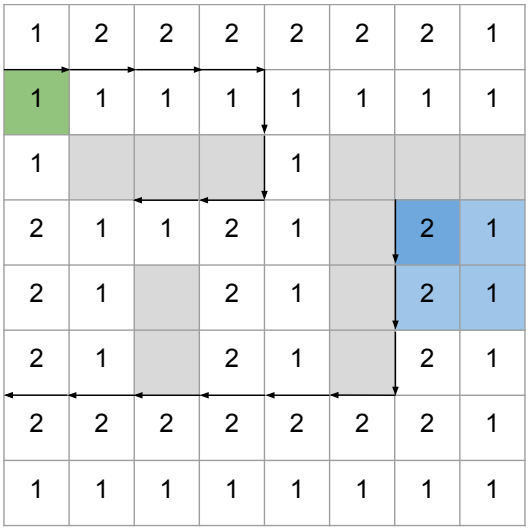
Implementačním problémem poté je, aby se ohodnocení v mřížce v reálném čase aktualizovalo v závislosti na pohybu okolních objektů. Přítomnost objektu 1*1 zamezuje výskytu objektu 2*2 vlevo apod.

7.2 Problém hledání cest

Klíčovou mechanikou hry je přesun objektů (jednotek) na cílové místo. Nález cesty z místa A na místo B v rámci mřížkové mapy. Hra vyžaduje pohyb skupin několika objektů různých velikostí zároveň, kolize objektů a efektivní rychlý algoritmus. Není požadavkem vyhledávat ideální nejrychlejší cesty v rozsáhlém bludišti. Je ale velmi důležité, aby se skupiny objektů chovaly očekávaně, rozumně a snadno se ovládaly – což má klíčový dopad na hratelnost hry.

7.3 Základní princip hledání cesty

Hra pro specifické případy, jako je jízda jedné jednotky, kde je vyžadována nejvyšší přesnost (např. těžba těžiče surovin), podporuje klasické optimalizované hledání pomocí algoritmu A* [14]. Přitom řeší několik situací:



Obrázek 7.1 Ocenění políček a pohyb

- Optimalizuje prohledávání odhadem vzdálenosti.
- Řeší kolize až když nastanou - buď vysláním příkazu kolizní jednotce, ať odjede stranou, nebo aktualizací hledané cesty.
- Do určité míry zvládá přesun skupin jednoduchým řešením kolizí – objekty, které jsou více vzadu, se nesnaží objíždět, a čekají, až předchozí objekty vykonají pohyb.

Ačkoliv hledání cest tímto způsobem má ve skupinách objektů relativně obstojné výsledky, vykazuje několik zásadních problémů:

- Je obtížné predikovat, který objekt přijede na cílové místo první a jaké bude rozmístění objektů při dosažení cíle.
- Výsledný pohyb objektů je řídký – objekty mají tendencí se roztáhnout přes celou cestu a trvá dlouho, než všechny dojedou do cíle. Což je poměrně významný problém při např. koordinaci útoku – kdy útočící jednotky jsou postupně masakrovány obrannou silou, neboť nemohou zaútočit zároveň (tento problém vykazují leckteré staré RTS).
- Při pohybu vyšších desítek objektů je tento způsob neoptimální a pomalý.

7.4 Hledání cesty pomocí vektorového pole

Cesty nalezené pomocí A* [14] mají nízkou provázanost a nedostatečně uvažují cesty ostatních objektů. Tento problém by patrně šel řešit buď nějakou optimalizací všech nalezených cest, nebo jedním spuštěním BFS se všemi zahrnutými objekty a další optimalizací. Tyto experimenty volně navádějí k použití vektorového pole [13].

7.4.1 Stavba vektorového pole

Vektorové pole (vector field) definuje pro každé políčko v mřížce vektor pohybu tak, aby pohybující objekt směroval k cíli. Základem pro vektorové pole je mapa ocenění políček podle počtu kroků od cíle jak je znázorněno na obrázku 7.2.

6	5	4	3	2	2	2	2
6	5	4	3	2	1	1	1
6	5					0	1
6	6		7	7		1	1
7	7		6	6		2	2
8	8	7	6	5		3	3
9	8	7	6	5	4	4	4
9	8	7	6	5	5	5	5

Obrázek 7.2 Ocenění políček vektorového pole

Z jakéhokoliv místa v mřížce je poté možné najít cestu k cíli postupným krokem z dražšího políčka na levnější (schéma na obrázku je zjednodušeno – ve hře mají diagonály větší ocenění).

7.4.2 Pohyb ve vektorovém poli

Objekt pohybující se po vektorovém poli může určit jako vhodné políčko to s nejmenší váhou. Problém nastává, když se pohybuje více objektů zároveň a cílové pole je již obsazeno. V takovém případě je třeba zkusit zvolit políčko podle jiného vektoru.

Jednou z možností je určení směrového vektoru $[v_1, v_2]$ z políčka $[a_1, a_2]$ pomocí váhy ocenění, kde $[o_1, o_2, o_3]$ je okolní políčko s parametry souřadnic o_1, o_2 a cenou o_3:

$$v_1 = \sum_{i=1}^{8} \frac{o_{1,i} - a_1}{o_{3,i}} \tag{7.1}$$

$$v_2 = \sum_{i=1}^{8} \frac{o_{2,i} - a_2}{o_{3,i}} \tag{7.2}$$

Výsledný vektor $[v_1, v_2]$ je poté nutné normalizovat. Takový vektor má tu vlastnost, že u vzdálenějšího políčka se pomaleji natáčí k cíli cesty. Vektor se použije k určení dalšího možného políčka k přesunu. V případě, že je v okolí neprostupné políčko, tak vektor určený váhou není definován a výsledný vektor se určí podle levnějšího políčka.

Různě rozmístěné skupiny objektů se poté pohybují plynuleji. K plynulosti pohybu skupiny objektů přispívá také skutečnost, že vektor vzdálenější od cíle určený váhou je odlišný od vektoru určeného levnějším políčkem – objekt tak může vybrat vhodnější vektor, pokud je cílové místo obsazeno.

Pro celočíselný výpočet se zvětšením z je výpočet následující (nutnost celočíselných výsledků je popsána v kapitole 8.2.1). V projektu použito $z = 10000$.

$$v_1 = \sum_{i=1}^{8} \left\lfloor \frac{(o_{1,i} - a_1) * z}{o_{3,i}} \right\rfloor \tag{7.3}$$

$$v_2 = \sum_{i=1}^{8} \left\lfloor \frac{(o_{2,i} - a_1) * z}{o_{3,i}} \right\rfloor \tag{7.4}$$

Celočíselně normalizovaný vektor $[n_1, n_2]$, kde int_sqrt je celočíselná odmocnina [7] v násobku z:

$$n_1 = \left\lfloor \frac{v_1 * z}{\text{int\_sqrt}(v_1^2 * v_2^2)} \right\rfloor \tag{7.5}$$

$$n_2 = \left\lfloor \frac{v_2 * z}{\text{int\_sqrt}(v_1^2 * v_2^2)} \right\rfloor \tag{7.6}$$

Během pohybu jsou objekty průběžně řazeny podle ocenění pro dosažení efektu, že vykročí jako první vždy ten, kdo je blíže k cíli. Ostatní čekají na pohyb objektu před nimi. Algoritmus pohybu znázorněn na obrázku 7.3 (problém překážek popsán níže).

7.4.3 Překážky a kolize

Pohybující se objekty mohou narazit na překážky v podobě jiných pohyblivých objektů, kterým se potřebují vyhnout. Tato problematika se dotýká komplikovaných problémů jako je simulace davu [15] – nicméně vzhledem k rozsahu se projekt tímto tématem detailně nezabývá a řeší překážky relativně jednoduchými způsoby.

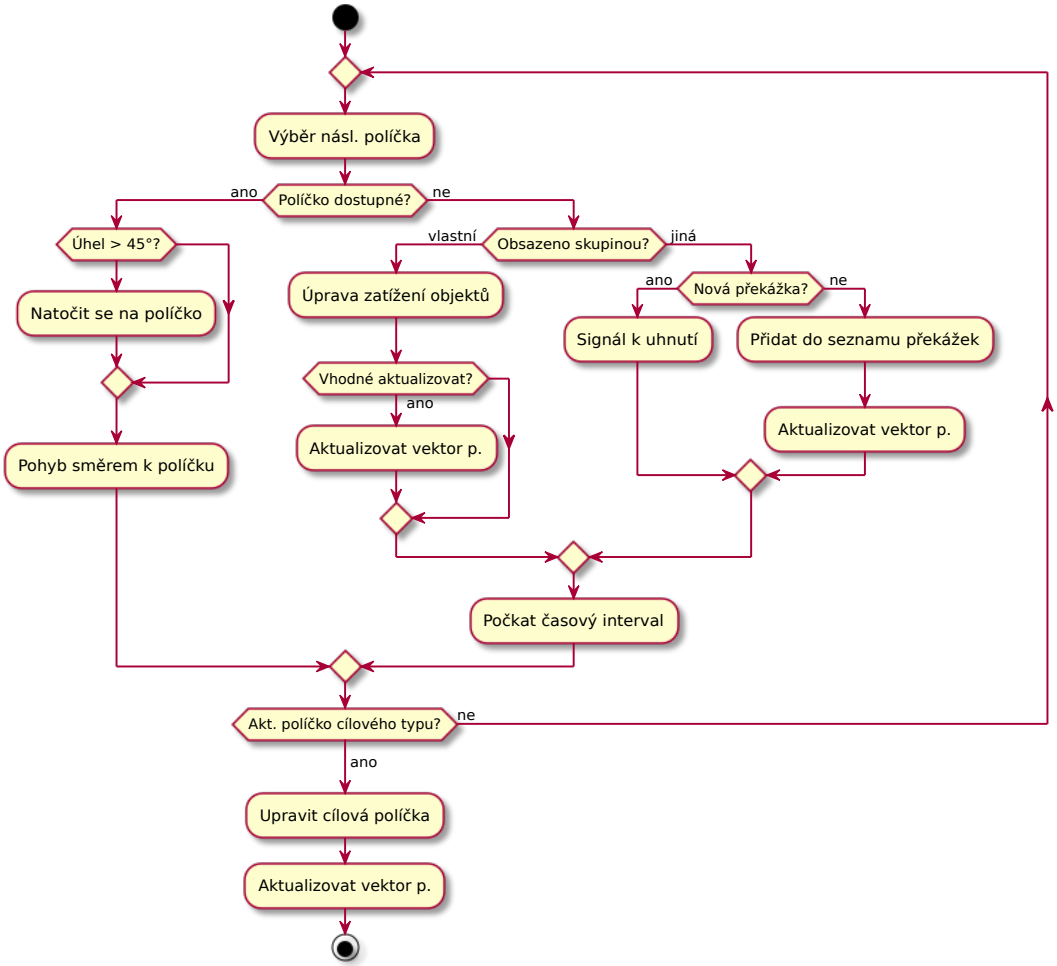
Sestavování vektorového pole v implicitním případě neuvažuje dynamické objekty jako překážky – uvažuje pouze terén a budovy. Na dynamické překážky reaguje až v průběhu pohybu.

Překážky z jiné skupiny objektů – pokud jde o překážku v podobě jednoho objektu, tak je překážce vyslán signál k uhnutí. Pokud do určitého časového limitu není cesta průjezdná, pak se všechny okolní objekty přidají do seznamu překážek a pomocí BFS se najdou sousedící překážky. Vektorové pole se poté aktualizuje s úvahou těchto překážek. Libovolná další aktualizace vektorového pole promaže seznam překážek jak je ukázáno na obrázku 7.4 – červeně jsou znázorněny objekty jako překážky ve skupině b a modře pohybující se objekty ve skupině a.

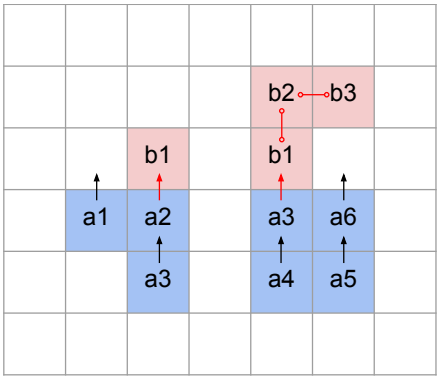
Překážky v rámci vlastní skupiny – objekty mohou během pohybu také překážet samy sobě. Typickým příkladem je úzké hrdlo jak je ukázáno na obrázku 7.5 – objekt $a3$ je zatížen objekty $(a2, a4, a5, a7)$, objekt $a2$ je zatížen objektem $a5$ apod. Všechny objekty v rámci skupiny ukládají reference objektů, kterými jsou zatíženy. V případě velké zátěže (např. 10 objektů) se objekt pokusí vybrat jiné cílové pole, které nemusí mít ideální směrový vektor (nicméně v případě příkladu na obrázku žádné takové pole neexistuje).

7.4.4 Doménový model principu hledání cest

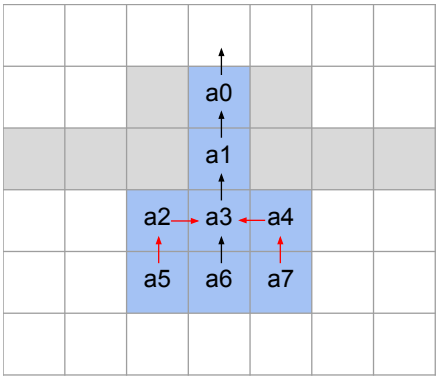
Doménový model znázorněn na obrázku 7.6 – jednotky *MoveableItem* jsou zahrnuty v rámci své skupiny *GotoGroup*. Hledání cesty pomocí A* [14] poskytuje třída *PathFinder*. Sestavení vektorového pole zprostředkovává *PathFinderFlow* navázaná na skupinu *GotoGroup* s datovou strukturou *PathFlowMap*. Každá jednotka ve skupině udržuje stav cesty ve třídě *PathFlowEntity*.



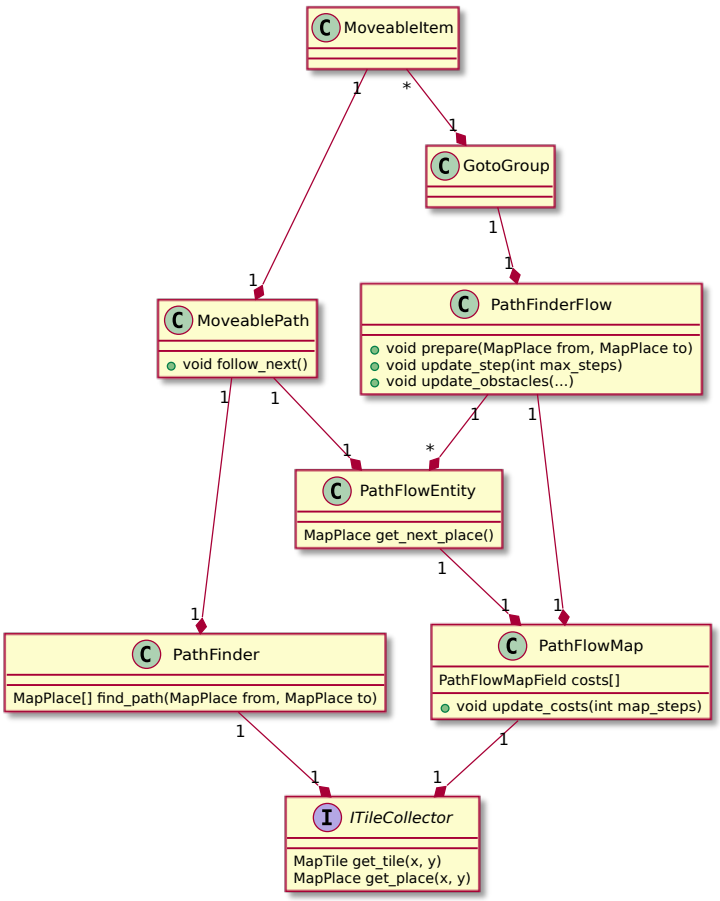
Obrázek 7.3 Algoritmus pohybu objektu



Obrázek 7.4 Detekce překážek během pohybu



Obrázek 7.5 Překážky úzkého hrdla

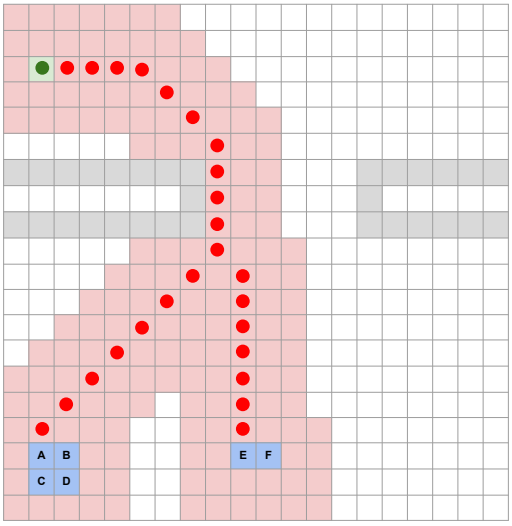


Obrázek 7.6 Doménový model principu hledání cest

7.4.5 Omezení prohledávání vektorového pole a optimalizace

Pro pohyb masy objektů není efektivní sestavovat vektorové pole pro celou mapu. Optimalizace probíhá v několika fázích. Nejdříve se skupina objektů rozdělí na dostatečně odlehlé podskupiny,

pro které se najde cesta klasickým algoritmem A* [14] s heuristikou manhattanské[1] vzdálenosti od cíle. Vektorové pole se poté sestavuje pro oblast do určité vzdálenosti od takto nalezených cest (podle odmocniny počtu a velikosti objektů). Demonstrováno na obrázku 7.7.



Obrázek 7.7 Optimalizace oblasti pro sestavení vektorového pole

Podstatná vlastnost, která pomáhá efektivnímu nasazení, je rozložení výpočetní zátěže. Algoritmus je tedy navržen tak, že lze realizovat hledání cesty s omezením na n iterací v rámci všech fází – od hledání cesty po sestavování vektorového pole. V každé logické iteraci hry se provede n kroků hledání cest. Požadavky hráčů na hledání cesty jednotlivých skupin objektů jsou prováděny v rámci fronty, kde provedení n iterací přesune skupinu na konec fronty, aby bylo garantováno, že nebude upřednostněn hráč s požadavkem na přesun více objektů.

Další důležitou optimalizační mechanikou je double-buffering vektorového pole. Případná aktualizace pole probíhá na novém bufferu – zatímco objekty pokračují v pohybu podle původního bufferu. Tento princip přispívá k pocitu plynulosti ovládání – při častém zadávání příkazů nedochází k zasekávání apod.

7.4.6 Omezení aktuálního řešení hledání cest

Během testování a vývoje řešení hledání cest se ukázalo několik problémů:

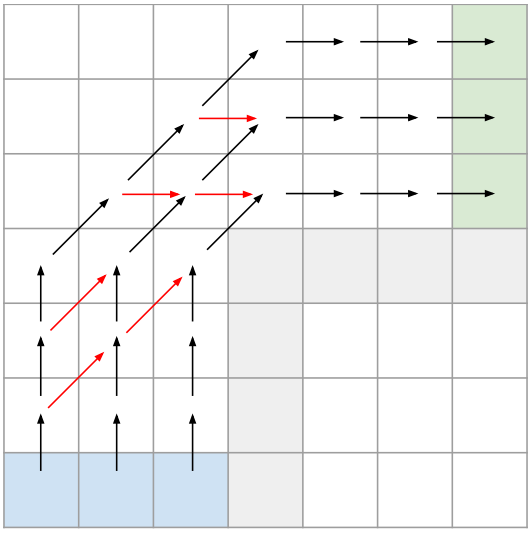
Ačkoliv implementace využívá optimalizace pomocí nástroje Cython a rychlou alokaci vektorového pole, tak některé dílčí části jednoduchým způsobem optimalizovat nelze (znamenalo by to zásadní refaktoring). Projevuje se to např. pomalou reakcí v případě přesunu 100 a více jednotek a pro finální řešení by byla vhodná další optimalizace (lze navýšit počty iterací – to je pak nicméně na úkor výkonu celé hry).

Restrikce pozic objektů na mřížkovou mapu není ideální – pohyb větších skupin by mohl být plynulejší. Navigace jednotek po mapě je sice dostatečná, ale při volném pozicování by mohl být výsledek lepší. Znamenalo by to ale zásadní změnu přístupu k řešení problémů – od hledání cest přes pohyb objektu, kolize a práci s vektorovým polem.

Současné řešení vektorového pole má nevýhodu, že v základu nedostatečně pracuje se skupinovými cíli. Pole vždy vykazuje buď diagonální problém, nebo vertikální problém, kdy se cíle sbíhají (v případě změny nastavení), jak je demonstrováno na obrázku 7.8. Ideální pole by fungovalo podle černých šipek, červené šipky jsou defektní. Projekt experimentoval s jednoduchým řešením

[1] Manhattanská vzdálenost mezi body $[a1, a2]$ a $[b1, b2]$ představuje $|b1 - a1| + |b2 - a2|$.

v podobě krabicového nebo gaussovského rozostření [35] ohodnocení polí, které sice v některých případech (např. zatáčky) vedlo k lepším vektorům, ale v rovných úsecích naopak způsobovalo problémy. Případný další vývoj hry by se tedy měl zaměřit na vyřešení tohoto problému (pokud nepůjde úplně jinou cestou).



Obrázek 7.8 Ideální vektorové pole

Kapitola 8
Hra více hráčů

Tato kapitola se zabývá hrou více hráčů a nezbytnou analýzou problému synchronizace stavu hry. Jsou nastíněny některé možné přístupy a je rozepsáno finální řešení sdílených deterministických simulací. Dále je v kapitole znázorněna architektura client-server a princip komunikace s přenosem instrukcí jednotlivých hráčů.

8.1 Problém synchronizace stavu hry

V případě hry dvou a více hráčů je nutné zajistit, aby každý hráč v průběhu času pracoval se stejným stavem hry – viděl tedy stejné prostředí, stejné rozložení herních objektů apod. V případě, že by hráč nemohl do hry zasahovat, pak by se jednalo pouze o jednosměrnou distribuci stavu a problém je pouze o přenosu informací ze strany serveru na stranu klienta. To by bylo vhodné např. v případě nějaké simulace, která probíhá na serveru. V případě, kdy hráč dostává možnost hru ovlivnit (např. pohnout objektem), nastává problém, jak toho docílit s ohledem na různé aspekty, jako je spolehlivost, přesnost, nároky na klienta a nároky na server.

V kapitole následuje několik úvah ohledně obecných přístupů k synchronizaci stavu hry [21]. Nicméně podstatou práce není jejich detailní popis – ten je rozvinut až u poslední metody použité v tomto projektu.

8.1.1 Synchronizace tahové hry

Čistě tahová hra, která neprobíhá v reálném čase, nahrává nejjednoduššímu a nejjistějšímu řešení problému. Hra začíná tím, že na základě nějakého principu je vybrán jeden hráč a ten obdrží od serveru potvrzení ke hře a ostatním jsou rozeslány pokyny k čekání. Vybraný hráč začne provádět sérii tahů a měnit stav hry. Jakmile tahy dokončí, odesílá na server jejich seznam, nebo přímo kompletní stav své hry a přepíná se do režimu čekání. Server distribuuje změnu stavu ostatním hráčům a vybere dalšího hráče, který pokračuje ve hře. Během této hry se nemůže stát, že by hráči táhli zároveň a narušili si navzájem stav hry. Lze si také snadno představit implementaci bez serverové části.

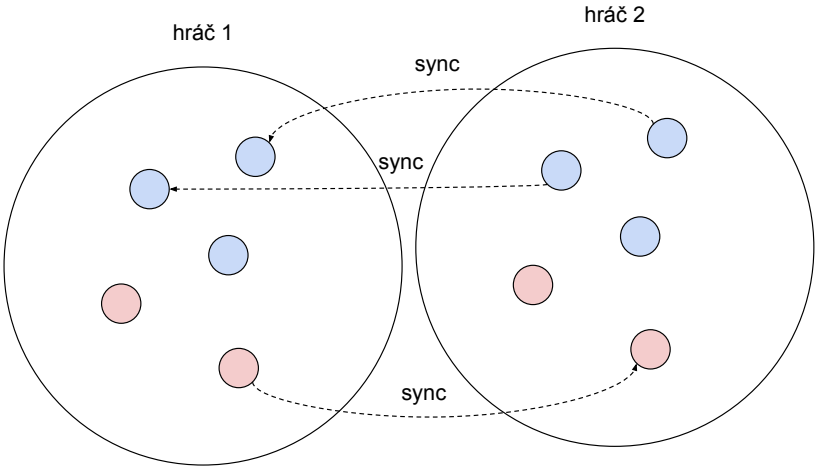
8.1.2 Synchronizace hry v reálném čase – málo objektů

Tento přístup předpokládá několik málo objektů, které hráč ovládá a ostatní objekty, se kterými interaguje. Objekty nabývají vlastností např. rychlost, velikost, kolize apod. s rozlišením objektů:

- ty, které jsou ovládány hráčem,

- ty, které ovládají ostatní hráči,
- ty, které nikdo neovládá ale interagují s pohybem hráče a kolidují s prostředím.

V tomto případě je nutné distribuovat informaci o pozicích, směrových vektorech a rychlostech ovládaných objektů ostatním hráčům. V případě, že hráč v rámci svého stavu hry koliduje s jiným objektem, posílá informaci o kolizi a pozice a vektory kolidujících objektů ostatním hráčům. Ostatní hráči tyto informace zpracovávají a upravují stavy dotčených objektů a případné prodlevy a nepřesnosti aproximují viz. obrázek 8.1.



Obrázek 8.1 Synchronizace hry v reálném čase

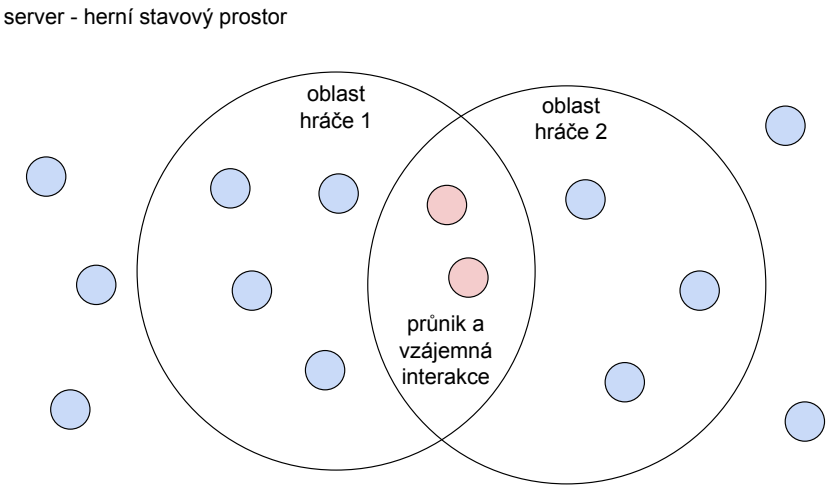
Řešení je také teoreticky možné bez nezbytné zastřešující serverové části – klienti si rozesílají stavy objektů mezi sebou např. negarantovaným protokolem UDP – v případě, když se paket ztratí, dojde k aktualizaci v dalším cyklu.

8.1.3 Synchronizace hry v reálném čase – mnoho objektů bez nezbytné závislosti

Problém navazuje na hru v reálném čase v případě, když hra obsahuje rozsáhlé světy s tisíci až deseti-tisíci interagujícími objekty. Toto řešení předpokládá, že hráč má v rámci hry svoji oblast zájmu v závislosti např. na pozici – tedy malou výseč světa, se kterou může interagovat. Jeho hra není nezbytně závislá na tom, co se děje na druhé straně světa. Řešení vyžaduje výkonnou serverovou část, která udržuje stavy všech objektů a provádí synchronizaci všem hráčům v rámci jejich průniků oblastí zájmu. Problém tohoto řešení může nastat v případě, kdy příliš mnoho hráčů má stejnou oblast zájmu viz. obrázek 8.2.

8.1.4 Synchronizace hry v reálném čase - mnoho objektů s nezbytnou závislostí

Případ, kdy mohou navzájem interagovat stovky objektů a oblast zájmu má v pokročilejší fázi hry neustálé průniky, je případ real-time strategie [34]. Pro zajištění funkčnosti je nezbytné, aby hra fungovala plně deterministicky bez narušení vnějšími vlivy. Stav hry A, který představuje

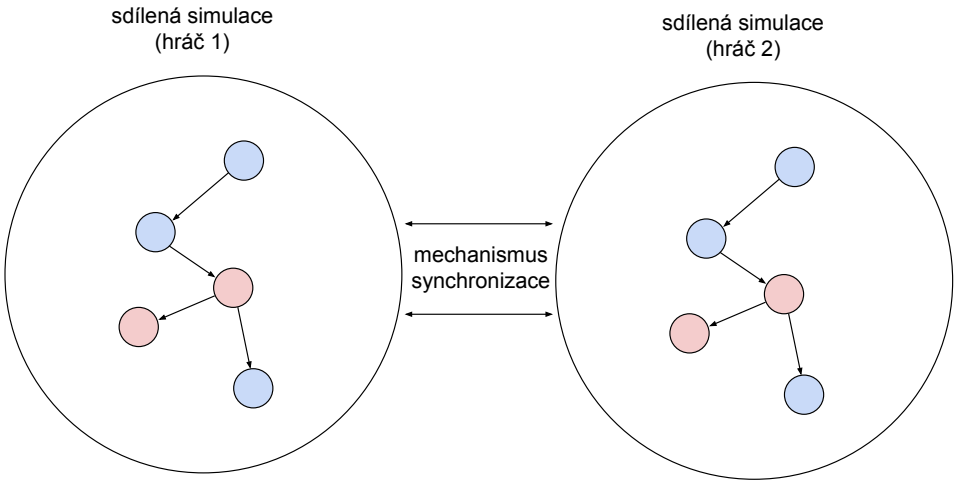


Obrázek 8.2 Synchronizace herního prostoru na serveru

konfiguraci herních objektů, se bez zásahu hráče po n iteracích dostane do stavu B. Tento princip musí být reprodukovatelný nezávisle na hardwaru nebo operačním systému.

Stav hry představuje takovou konfiguraci herních objektů, které mají vliv na rozhodující výsledek hry – např. objekt protihráče byl přesunut, zničen, poškozen apod. Stav hry nemusí obsahovat např. grafické efekty, animace a jiné detaily.

Jakmile je tento deterministický princip zajištěn, pak je pro synchronizaci nezbytné, aby instrukce, které hráč ve hře zadává, byly provedeny u všech hráčů v rámci stejného čísla iterace hry. Toto zajišťuje serverová část, která nepracuje přímo se stavem hry, pouze přeposílá instrukce připojeným klientům ve správnou chvíli viz. obrázek 8.3.



Obrázek 8.3 Synchronizace simulací

Projekt pro synchronizaci stavu hry používá právě tuto metodu.

8.2 Deterministická simulace

Zvolenou synchronizační metodu lze zobecnit jako synchronizaci několika deterministických simulací – tedy takových, které mají při stejných vstupech a výchozích podmínkách stejný reprodukovatelný výstup. Simulace se mění pouze při obdržení příkazu od hráče, který musí všechny ostatní simulace realizovat ve stejný okamžik.

8.2.1 Problémy zachování determinismu

Podstatu determinismu narušuje automatické spoléhání na vždy stejné výsledky operací s čísly s plovoucí řádovou čárkou. Princip počítání s řádovou desetinnou čárkou je sice ze samé podstaty deterministický, nicméně výsledky mohou být v detailech rozdílné, pokud se liší hardware nebo operační systém [16]. Existují nízkoúrovňové metody, které determinismus čísel s řádovou desetinnou čárkou zajistí platformě nezávisle[1] – nicméně tato práce se touto cestou nevydává a upřednostňuje jistý způsob použití celočíselných hodnot.

Dále jsou problematické iterace přes kontejnery, které jsou nestabilní a negarantují tedy uspořádání prvků. Konkrétně kolekci *set* (množina) je třeba nahradit kolekcí *OrderedSet*.

V případě kolekce *dict* (slovník) – u klíčů slovníku nemusí být garantováno pořadí ve všech implementacích jazyka Python. Je třeba použít *OrderedDict*.

Posledním důležitým zdrojem problémů, který je třeba důsledně ošetřit, je síťová komunikace a pořadí paketů[2]. Pro správné fungování musí být vždy zajištěno stejné pořadí na straně všech komunikujících klientů.

8.2.2 Celočíselné principy a algoritmy

Projekt v zájmu vyvarování se použití čísel s řádovou desetinnou čárkou v částech, které souvisí se stavem hry, používá následující celočíselné principy a algoritmy:

Celočíselné dělení – informace pozic objektů $[x, y]$ je uložena v násobcích 20-ti a pro jejich konverzi se používá celočíselné dělení. Tento postup simuluje určitou malou přesnost desetinného čísla. Pro rozsáhlé prostory může být limitující omezená velikost celočíselných typů, nicméně pro tuto hru je dostatečné i použití 32-bit čísla (některé výpočty pracují i s 64-bit číslem).

Mapa goniometrických funkcí – mapa celočíselných úhlů mezi dvěma body, včetně reverzních hodnot vzdálenosti a úhlu (omezeno na určitou výseč bodů, která je pro hru dostatečná).

Celočíselná odmocnina – používající princip binárního hledání [7].

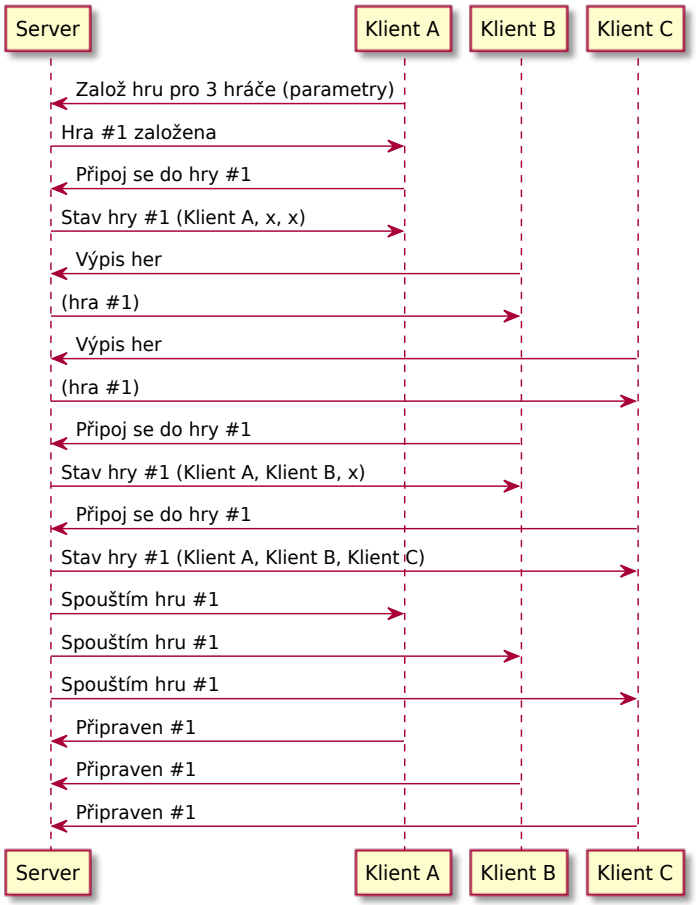
Rasterizace úsečky – používající Bresenhamův algoritmus [17]. Princip použit např. pro trajektorie střel.

8.3 Architektura client-server a doménový model

Projekt je rozdělen na serverovou část a klientskou část. Pro hru více hráčů je nutné spustit server, který obstarává veškerou výměnu informací mezi připojenými klienty. Server běží zcela autonomně a poskytuje i příkazy typu výpis her, založení hry, připojení do hry apod. Komunikace je demonstrována na obrázku 8.4.

Doménový model představuje třídu *Server*, která obstarává libovolné množství *Connection* s příchozími zprávami *Command*. Katalog těchto zpráv udržuje také server, který je předává

[1] Při striktním dodržování standardu IEEE 754.
[2] Blok dat přenášený v rámci počítačové sítě.



Obrázek 8.4 Demonstrace síťové komunikace se založením hry

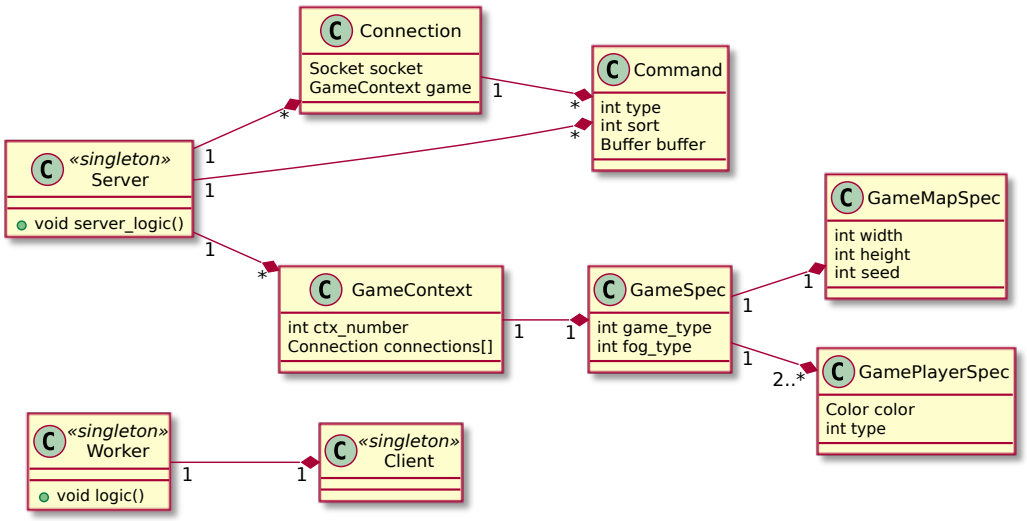
všem relevantním připojeným klientům. Server také udržuje kolekci založených her *GameContext* společně se specifikací hry *GameSpec*, *GameMapSpec* a *GamePlayerSpec*. Zprávy rozesílané jednotlivým připojeným klientům jsou rozlišeny podle relevantního herního kontextu, který klienti v rámci jedné hry sdílejí viz. obrázek 8.5.

8.3.1 Moderní přístup k síťové hře

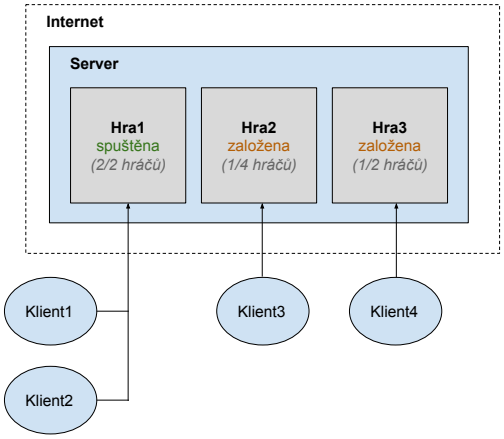
Historie hraní her po síti je protkána mnohými problémy – od nefunkčností lokální sítě, problémy s porty, firewally a podobně. Moderní návrh architektury síťové hry tyto problémy eliminuje způsobem, že od všech hráčů vyžaduje internetové připojení i pro hru v malých komunitách (obrázek 8.6). V případě nenáročného serveru není takové řešení ani příliš nákladné a otevírá lepší dostupnost hry po síti a tím i lepší herní zážitek. Podobná řešení často také integrují dodatečné nástroje, jako jsou chaty, voice-chaty, katalogy přátel apod. pro podporu komunit a socializovaného hraní, které v konečném důsledku prospívá samotnému produktu hry.

8.3.2 Princip přenosu herních instrukcí

Vhodné pro toto řešení je garantovaný protokol TCP. Server periodicky po např. 5-ti logických iteracích hry (1 krok) posílá všem klientům SYNC[3], který obsahuje příkazy všech hráčů pro



Obrázek 8.5 Doménový model client-server



Obrázek 8.6 Server na veřejném internetu

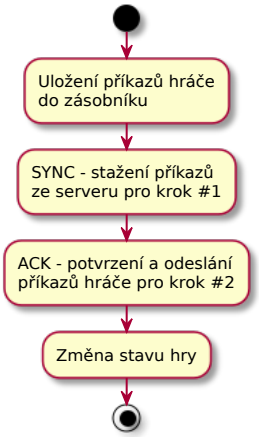
krok 1 a od každého klienta očekává ACK[3], který obsahuje provedené příkazy hráče k provedení v dalším kroku 2, jak je demonstrováno na obrázku 8.7.

Server SYNC nepošle, dokud neobdrží od všech připojených hráčů ACK. Klientská část provádí příkazy podle přesně stanoveného čísla iterace. Je tím tedy zajištěn konzistentní běh hry u všech připojených klientů. V případě, když klient neobdrží včas SYNC, tak čeká. Příčinou může být např. vysoká latence připojení. Pro balíček paketů s instrukcemi je klíčové, aby byly na straně každého klienta provedeny ve stejném pořadí – jinak může dojít k de-synchronizaci.

8.3.3 Problém de-synchronizace

Jednou z hlavních nevýhod tohoto řešení je riziko de-synchronizace. Malá změna na straně jednoho klienta může mít na základě efektu motýlích křídel dalekosáhlé důsledky. Herní objekt se posune o malý kousek jinam a za chvíli dva hráči hrají zcela rozdílnou hru.

[3]Příkazy SYNC, ACK představují interní příkazy protokolu hry postaveného nad TCP.



Obrázek 8.7 Synchronizační krok

Příčinou tohoto problému je vždy nějaký zdroj nedeterministického chování (viz. problémy zachování determinismu 8.2.1). Odhalení těchto chyb je poměrně náročné. Pro účely ladění má program možnost zapnout režim kontroly, který odesílá stav všech objektů na server a serverová část stavy porovná a případně určí, u kterého objektu nastal problém. Ve finální verzi hry je nicméně tato kompletní kontrola zredukována na občasné odeslání pozic vybraných objektů, které poté serverová část zkontroluje, zda jsou všechny shodné. V případě problému zašle SYNC_ERROR a hra se ukončí. Nutno dodat, že ve finální verzi hry jsou všechny problémy odladěny a problém de-synchronizace již nenastává.

8.3.4 TCP a UDP

Zdá se jako všeobecná shoda, že jediný vhodný protokol pro počítačovou hru je UDP. Je o tom napsáno spousty článků, které ujišťují o správnosti použití UDP, především z důvodu nízké latence a komplikovanosti protokolu TCP [18]. Je pravdou, že obvykle jde o střílečky nebo jiné hry vyžadující co nejkratší odezvu. V případě použití TCP je riziko latence až ve stovkách msec. Lze nicméně nalézt dost případů her, které přesto TCP úspěšně využívají (především hry typu simulátor, strategie apod.).

Pro tento projekt je limitovaná odezva protokolu TCP dostatečná. Navíc při použití konfigurační klauzule soketu TCP_NODELAY[4] je riziko vyšší latence zredukováno.

Samotné použití UDP je problematické, neboť je pro hru, vzhledem ke způsobu synchronizace, nezbytná garance paketů – a tak použití UDP je v zásadě podobné, jako snaha o opakované vynalézání TCP protokolu. Pro hru navíc není rozhodnutí mezi TCP a UDP architektonicky klíčové a případně lze implementaci předělat.

[4]Dochází k vypnutí Naglova algoritmu, který může vést k zadržováním paketů a jejich odesíláním ve větších dávkách a tím zvyšování latence.

Kapitola 9
Umělá inteligence

Umělá inteligence představuje v rámci počítačové hry simulaci chování lidského hráče. Vývoj umělé inteligence je složitý problém a jeho analýza a implementace by mohla být v rozsahu celé práce. V kapitole je rozepsán princip fungování umělé inteligence, která je implementována v rámci projektu – tedy jakými informacemi disponuje, jak je sbírá a jak se rozhoduje.

9.1 Cíl a přístupy vývoje umělé inteligence

Dostatečný cíl funkční umělé inteligence je pro projekt stanoven následovně: AI průběžně staví svoji základnu a budovy s produkcí jednotek se snaží vylepšit na maximální technologickou úroveň. Staví obranné věže na místech, kde je větší riziko útoku. Vedle stavby základny také paralelně produkuje jednotky. Průběžně prozkoumává prostředí mapy a zaznamenává pozice a výskyty oponentů. V případě napadení protihráčem mobilizuje jednotky k obraně. Vytváří útočné skupiny jednotek, přesouvá je na vhodná místa a provádí útoky. Aktuální fáze projektu nepočítá se samostatným učením a změnou komplexní strategie podle situace. Předpokládá základní ovládání skupin jednotek a určováním jejich cílů.

Rozhodovací přístupy vývoje umělé inteligence:

Konečné automaty – rozhodovací princip, který obsahuje jasně definované stavy, ve které se část hry může nacházet, a podmínky přechodů mezi těmito stavy. Chování automatu je tedy jasně dané. Nevýhodou může být předvídatelnost a také komplikovanost až nemožnost použití pro rozsáhlé stavové prostory. Hra tohoto principu využívá.

Rozhodovací stromy – struktura, která stavy a podmínky organizuje do souvislého orientovaného grafu bez cyklů. V jednoduché podobě může být přístup zaměněn konečným automatem, nebo přepínáním konečných automatů. Rozhodovací stromy jsou implementovány s dalším větvením a učením se změnou podmínek. Implementace AI se tohoto dotýká velice okrajově – tedy jednoduchými stromy, které snesou záměnu s několika stavovými automaty.

Fuzzy logika – podobor matematické logiky, kde jsou jednotlivé výroky ohodnoceny mírou pravdivosti (nepřítel je blízko, zaútočilo hodně jednotek apod.). Význam výroků se navíc může měnit v závislosti na změně kontextu – např. „hodně jednotek“ má jiný význam na začátku hry, uprostřed hry a na konci hry. Pro konverzi neurčitého výroku na určitý, který může zpracovat konečný automat, se používá termín fuzzyfikace – AI tento přístup využívá pro zhodnocení některých stavů.

Markovův rozhodovací proces – matematický rámec, který umožňuje nalézt vhodnou reakci v prostředí, které je z části dáno náhodně a z části vlastním rozhodnutím [28]. Implementace AI se o tento problém opírá také velice okrajově – v rovině heuristiky (formálním popisem se nezabývá).

Plánovací přístupy umělé inteligence:

Cílem orientované plánování – přístup, který stanovuje dílčí cíle pro naplnění hlavního cíle bez závislosti na pořadí nebo konkrétní implementaci stavového automatu [30]. AI tento přístup používá na ovládání skupin jednotek.

Monte Carlo plánování – plánování metodou Monte Carlo [29] – tedy výběr náhodných plánů, jejich validace a finální výběr nejlepšího plánu. Tento přístup může být vhodný pro tahové hry, nebo rozhodování v zjednodušeném modelu hry nebo omezeném stavovém prostoru. Vzhledem k charakteru hry a potřebě rychlého rozhodování v reálném čase se pro tento projekt příliš nehodí.

9.2 Návrh umělé inteligence

Hra se tedy zaměřuje na umělou inteligenci na bázi jednoduchých rozhodovacích stromů a stavových automatů s potenciálem dalšího rozšíření.

Návrh AI stojí na předpokladu, že nepodvádí a má k dispozici stejné prostředky, jako lidský hráč. Princip AI je tedy nadstavbou herního systému. Základní koncept spočívá v simulaci očí – tedy skenování viditelného prostředí – a rozhodovací logiky, která posbírané informace vyhodnocuje.

Doménový model je znázorněn na obrázku 9.1 – kde *AiContainer* obstarává rozhodovací logiku na principu stavového automatu, *AiMapScan* sbírá informace o stavu okolního prostředí, *AiGroup* představuje skupinu jednotek, *AiChoices* spravuje jednotky k výrobě a *AiMoney* obstarává správu zdrojů. Informace z uživatelského prostředí jsou předávány pomocí registrovaných událostí.

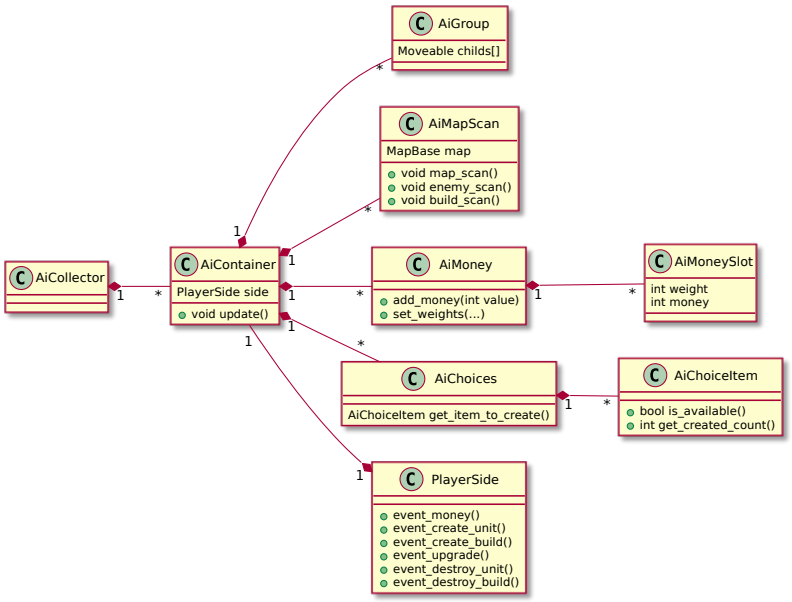
9.2.1 Alokace zdrojů a stavba základny

Elementární mechanikou je stavba základny, bez které se hráč nemůže posunout dále – k tomu AI rozděluje natěžené zdroje do základních skupin: rafinerie (*resources*), budovy (*build*), obrana (*defense*), technologie (*upgrade*), jednotky (*units*) – a těmto skupinám přikládá váhu podle situace ve hře (váha 0 redistribuuje zbylé suroviny do ostatních skupin). Stavba základny je z části skriptovaná sekvence, kde postavení první rafinerie má prioritu 100% – po jejím postavení se priorita mění na stavbu budov, tedy kasárny, továrny a poté laboratoře, kde každá budova je podmíněna sadou konkrétních cílů – např. pokud oponent útočí, tak je upřednostněna produkce jednotek a tedy stavba odpovídajících budov – pouze ale v případě, pokud je již postavena alespoň jedna rafinerie.

Lokalizování umístění budov probíhá na stejném mechanismu, jako princip v rámci standardního uživatelského rozhraní, který vyhodnocuje vhodnost umístění budovy (max. 2 políčka od jiné budovy). V případě stavby rafinerie hledá takové vhodné místo, které je co nejblíže surovinovým zdrojům.

9.2.2 Skenování okolního prostředí

Pro analýzu situace ve hře provádí AI skenování okolního prostředí mapy. Provádí několik rozdílných fází:



Obrázek 9.1 Návrh struktury tříd AI

Analýza viditelného okolí – od jednotek a budov prochází viditelné prostředí a ukládá pozice zdrojů surovin a nepřátelských jednotek nebo budov. Navštívená políčka ukládají míru návštěvy – která se postupně redukuje, když se políčko ztratí z dosahu. Je tím zajištěna možnost hledání naposledy navštívených oblastí.

Označení nepřátelského prostředí – kolem nalezených nepřátelských jednotek označuje omezenou okolní oblast a vytváří určitou paměť, kde byl naposledy nepřítel k vidění. Tuto oblast po určitém časovém intervalu cíleně „zapomíná“.

Analýza základny – okolí vlastní základny označuje vzdáleností od poslední budovy pro vymezení oblasti obrany a případné stavby věží.

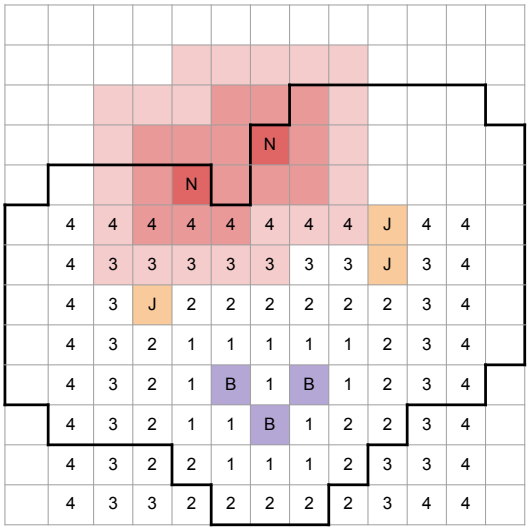
Interní paměť je znázorněna na obrázku 9.2 – kde N představuje nepřítele, J vlastní jednotku, B budovu. Černě je ohraničena viditelná oblast, číselně oblast základny a červeně paměť výskytu nepřátelských jednotek.

Nad touto pamětí jsou postaveny rozhodovací principy – zejména kombinace aktuálního výskytu nepřítele (červeně) a okolí základny (číslo) představuje signál obranným jednotkám. Paměť výskytů nepřátel slouží k vhodnému rozmístění obranných věží. Na základě detekce ostatních budov probíhá lokalizace základny nepřítele. Útočné skupiny se přesouvají na místa posledního výskytu nepřátelských jednotek.

Případná možnost, jak přiblížit tento princip chování skutečného lidského hráče, by byl pohyb viewportu[1] a omezení skenování na aktuálně viditelný viewport. V aktuální fázi projektu AI skenuje vždy celé dostupné okolí a to i takové, které „nevidí“ – dá se tedy říci, že v tomto případě lehce podvádí.

Je třeba zmínit, že skenování prostředí může být výpočetně náročné – nicméně kód je plně optimalizovaný pomocí nástroje Cython včetně rychlých datových struktur. Také je algoritmus skenování navržen tak, že může být omezen na maximálních n iterací – takže kompletní skenování

[1] Viewport je oblast obrazovky určitého rozměru, která je namířena na konkrétní výseč světa – v tomto případě mapy hry.



Obrázek 9.2 Skenování okolního prostředí

může být rozloženo přes několik logických cyklů hry. Výsledek je více než dostatečný co se týče výkonu a rychlosti.

Dalším prostorem pro případnou optimalizaci je paralelní zpracování logiky umělé inteligence – projekt se tím ale zatím nezabývá (především kvůli omezení GIL).

9.2.3 Skupiny jednotek a reakce

Logika AI v okamžiku, kdy se jí podaří postavit kasárnu nebo továrnu, začíná vyrábět jednotky. Postup výroby jednotek stojí na základě kolekce s nabídkou aktuálně dostupných jednotek – kde volba k výrobě je podmíněna principem postaveným na počtu již vyrobených jednotek a stanoveném maximálním počtu. Případný další vývoj AI by měl pokračovat s rozšířením tohoto principu – např. reflexí volby jednotek oponenta, nebo úspěšností jednotek v boji. Vyrobené jednotky AI rozděluje do několika typů skupin:

Průzkumná skupina – v rámci této skupiny je udržována jedna jednotka, která jezdí po mapě s cílením políček nejmenšího indexu návštěvy. Jinými slovy prozkoumává a objevuje mapu a hledá oponenty. Skupina nemá za cíl útočení a snaží se vyhýbat nepřátelským jednotkám.

Obranná skupina – skupina, do které AI vkládá nově vytvořené jednotky. Má omezenou velikost a když je překročena, vytvoří se skupina další. Jednotky v rámci těchto skupin jsou rozmísťovány kolem základny v závislosti na ohodnocení cílového místa rizikem výskytu nepřátel (políčka takto označená v rámci skenování mapy).

Obranně-útočná skupina – v případě, že je narušen prostor základny (dle stavu skenované mapy), tak se vybere odpovídající množství obranných jednotek, které se vloží do obranně-útočné skupiny, které se zadá signál k útoku na detekované místo narušení – příkaz *attack-move*.

Útočná skupina – v okamžiku vyhodnocení správného momentu AI vybere určité množství obranných jednotek a přesune je do nové útočné skupiny. Tato skupina poté útočí na dříve lokalizované místo základny oponenta.

V rámci těchto skupin umělá inteligence v současné verzi zhodnocuje několik aspektů vývoje hry a snaží se na ně adekvátně reagovat těmito způsoby:

Cílení s využitím výhody – na základě analýzy bezprostředního okolí jednotek provádí zhodnocení, na které jednotky je vhodné útočit a proti kterým má výhodu a na tyto jednotky se snaží efektivně cílit. Zhodnocuje také, zda je nějaká oponentovat jednotka poškozena a považuje ji tak jako primární cíl.

Ústup ze zničení – v případě, že dojde ke zničení kritického množství jednotek útočící skupiny, tak je proveden ústup – buď ve formě, že se sloučí více útočících skupin, nebo se skupiny stáhnou do základny.

Ústup z nevýhody – pokud útočící skupina narazí na výraznou množstevní přesilu, tak se stahuje do základny. Zde je prostor pro složitější analýzu jednotek oponenta a stažení v případě, když je výměna zjevně nevýhodná. Aktuální verze AI toto nezhodnocuje.

Režim paniky – v případě, že jsou v oblasti základny nepřátelské jednotky v množství větším, než jsou jednotky obranné, tak nastává režim paniky – a AI investuje veškeré zdroje do výroby jednotek.

Podstatným pilířem vývoje umělé inteligence je snadné testování, které je popsáno v kapitole 13.1.3.

9.2.4 Další možnosti rozšíření

Případný budoucí vývoj by měl počítat s vyhodnocením úspěšnosti nasazených jednotek a přizpůsobení reakce na konkrétní jednotky oponenta. Lze také pracovat s možnostmi defenzivní nebo ofenzivní strategie a oddalováním produkce se stavbou více továren a čekání na adekvátní reakci. Další přístupy ale také souvisí s rozšířením obsahu jako takového a dalších možností v rámci hry.

Experimentování různých implementací AI vs AI může vést k zajímavým poznatkům a také k dalším možnostem vybalancování hry. Za zvážení také stojí prozkoumání možnosti implementace strojového učení na některé části umělé inteligence (např. interakce jednotek).

Kapitola 10

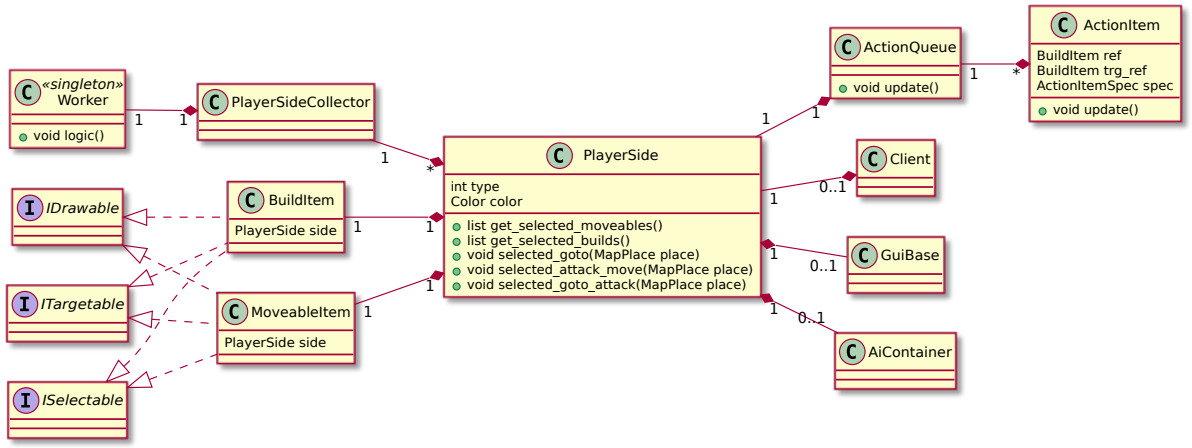
Prostředí hráče a možnosti kombinování

Zde je znázorněno, jak návrh hry pracuje s pojmem „hráč“ a jakým způsobem je v prostoru hry zajištěno, že může hru ovládat více hráčů zároveň, ať už lidských, nebo v podobě umělé inteligence.

10.1 Doménový model struktury hráče

Základní prostředí hry, které se skládá z objektů, budov a střel, funguje jako autonomní simulace a její stavový prostor přímo nesouvisí s entitou hráče. Pojem „hráč“ je tedy pouze vlastností, kterou nabývají jednotlivé jednotky a budovy.

Doménový model této části je znázorněn na obrázku 10.1 – kde entitu hráče představuje *PlayerSide*, která je navázána na herní objekty např. *MoveableItem*, *BuildItem*. Jediná stavová vlastnost je katalog akcí ke stavbě budov a jednotek *ActionQueue* a *ActionItem*. Samotná entita hráče udržuje seznam svých jednotek a budov a zprostředkovává jejich příkazy. Entita hráče je ovládána pomocí uživatelského rozhraní *GuiBase* s využitím / nebo bez využití *Client* (pro síťovou hru) – nebo je ovládána umělou inteligencí *AiContainer*.



Obrázek 10.1 Doménový model struktury hráče

10.2 Možné režimy hry

Hra, podle předchozích kapitol, podporuje dva základní režimy hry – **hru jednoho lidského hráče** (single-player) a **hru více lidských hráčů** (multi-player). Vzhledem ke charakteru hry v podobě sdílené simulace je hra jednoho lidského hráče detailem, kdy je pouze vypnuta klientská část, která obstarává komunikaci se serverem – a instrukce zadané hráčem jsou realizovány přímo (nečeká se, až je obdrží všichni připojení hráči). Hra pochopitelně poté umožňuje hru pouze proti umělé inteligenci.

10.3 Kombinování typů hráčů

Jak již bylo zmíněno, tak entitu hráče může ovládat buď **lidský hráč** (LH) nebo **umělá inteligence** (AI). Možnosti jejího nastavení se liší podle toho, jaký režim hry je zvolen – v případě hry jednoho lidského hráče, jsou praktické možnosti LH + libovolný počet hráčů AI. Reálná možnost ovšem je, nastavit klidně např. LH + LH + LH – znamená to potom, že hraje lidský hráč proti lidskému hráči a ve hře lze mezi jednotlivými hráči přepínat. Tato možnost je určena pouze pro testování – praktické využití nemá. Lidský hráč je v tomto případě jinými slovy hráč, kterého neovládá umělá inteligence.

Skutečná možnost kombinace lidských hráčů nastává v případě, kdy je spuštěna a založena hra více lidských hráčů na serveru, ke které se mohou připojit. Scénář LH + LH + LH představuje několik hráčů, kde je každý připojen ze svého zařízení. V případě nastavení umělé inteligence např. LH + LH + AI, se hráč s umělou inteligencí chová podobně, jako hráč na serveru – je tedy spuštěnou instancí hry, která je pomocí komponenty *Client* připojena na server a zároveň má aktivovanou komponentu *AiContainer*. Hru je možné spustit vícekrát i na stejném zařízení – je možné uvedené režimy libovolně kombinovat.

Ačkoliv návrh hry je koncipován tak, že umožňuje běh herní logiky bez vykreslování a je realistické spustit samostatný kontejner s umělou inteligencí, tak tato přímočarou možnost server zatím nepodporuje – a dodělání této možnosti je naplánováno na případný další vývoj.

Kapitola 11
Uživatelské rozhraní

V kapitole je rozepsán problém uživatelského rozhraní a možnosti jeho implementace ve zvoleném vývojovém prostředí.

11.1 Prostředí uživatelského rozhraní

Grafická knihovna Pyglet disponuje pouze omezenou možností výroby uživatelského rozhraní. Poskytuje nástroje na zobrazení formátovaného textu a textových vstupů. Částečnou podporu uživatelského rozhraní poskytují až poslední verze knihovny a to v omezené míře (panel, tlačítko, textbox). Je tedy vhodné se ohlédnout za možnostech nadstaveb s uživatelským rozhraním pro knihovnu Pyglet.

Za zmínku stojí knihovna Glooey [19] – která se zdá dostatečná a dobře konfigurovatelná. Po otestování bohužel nastává několik problémů – knihovna neumožňuje snadné ošetření kontejneru v případě přetečení prvků a program s vyvoláním výjimky spadne. Zásadnějším problémem se ale ukazuje, že je knihovna příliš pomalá. Její nasazení snižuje hodnotu FPS na polovinu. Knihovna by šla použít pro základní herní menu, které se při spuštění hry vypne. To nicméně nestačí – v této hře je vyžadováno použití uživatelského rozhraní i během hry.

Knihovna Kytten – pro Pyglet má vytvořen novou větev, ačkoliv pro něj původně nebyla stavěná. Nicméně tento fork je příliš starý – a nestojí tedy ani za experimentování.

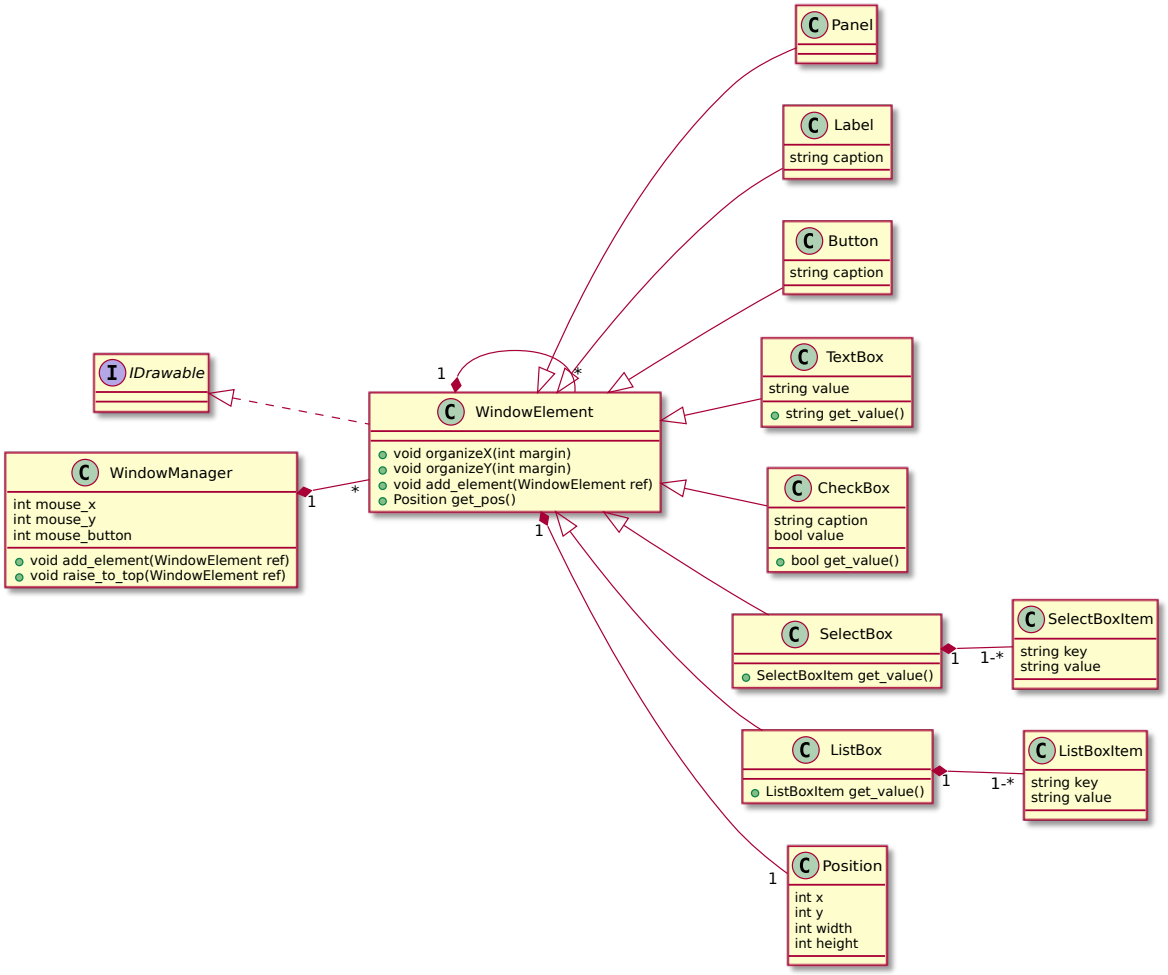
Podobně jsou na tom další knihovny, které představují spíše experimenty jednotlivců, než plnohodnotné knihovny. Zdá se tedy jako vhodné řešení vytvořit vlastní podsystém pro uživatelské rozhraní.

11.1.1 Doménový model systému uživatelského rozhraní

Požadavky na systém uživatelského rozhraní nejsou příliš dramatické – je potřeba: panel, label, checkbox, tlačítko, selectbox/listbox, textbox.

Doménový model je na obrázku 11.1 – který funguje na bázi hierarchie obdélníkových objektů. Třída *WindowElement* umí zároveň fungovat jako neviditelný kontejner, který organizuje prvky vertikálně nebo horizontálně. Viditelnou plochu okna představuje *Panel*. Události myši a další orchestraci zajišťuje *WindowManager*. Funkci editace textového políčka poskytuje knihovna Pyglet. Systém uživatelského rozhraní nevyžaduje příliš složité interakce ve formě správy otevřených oken apod. – systém pracuje především s jedním otevřeným oknem a případným modálním dialogem[1]. Systém umožňuje také volné pozicování prvků uživatelského rozhraní relativně k pravému / levému / hornímu / spodnímu okraji obrazovky nebo panelu.

[1] Modální dialog představuje okno uživatelského rozhraní, které vyžaduje bezprostřední vstup uživatele.



Obrázek 11.1 Doménový model uživatelského rozhraní

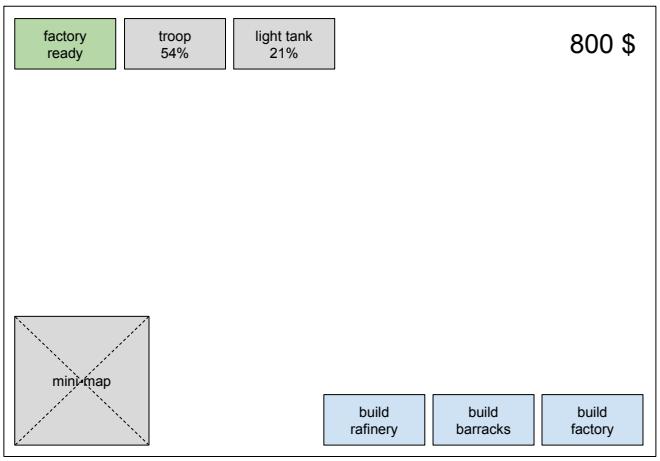
11.2 Návrh uživatelského rozhraní

Uživatelské rozhraní se skládá ze dvou zásadních částí – a to **rozhraní menu hry** (UC.a) a **rozhraní pro ovládání hry samotné** (UC.b). Toto zahrnuje několik uživatelských scénářů:

- **UC.a1** – uživatel nastaví a spustí hru jednoho hráče proti počítači.
- **UC.a2** – uživatel nastaví hru pro více hráčů po síti a připojí se do ní.
- **UC.a3** – uživatel změní nastavení prostředí hry.
- **UC.b1** – hráč postaví budovu základny.
- **UC.b2** – hráč těží surovinové zdroje.
- **UC.b3** – hráč postaví jednotky.
- **UC.b4** – hráč zvolí jednotky a zadá příkaz pro přesun nebo útok.
- **UC.b5** – hráč vylepší technologickou úroveň továrny.

- **UC.b6** – hráč vyhraje nebo prohraje hru.
- **UC.b7** – hráč reaguje na chybu (typicky odpojení od sítě v případě síťové hry).

Uživatelské rozhraní je navrženo v zájmu jednoduchosti a uživatelské přívětivosti. Návrh rozhraní uvnitř hry je znázorněn na obrázku 11.2 – které dělí plochu okna na 4 části. V pravém horním rohu jsou zobrazeny dostupné zdroje, za které je možné stavět jednotky nebo budovy. V pravém dolním rohu je kontextové menu – obsah menu se mění podle toho, zda je vybrána budova (zobrazí se nabídka ke stavbě) nebo jednotka či skupina jednotek (zobrazí se nabídka akcí jednotek). V levém dolním rohu se zobrazuje mini-mapa. V levém horním rohu je fronta aktuálně zpracovávaných akcí – přičemž akce, které vyžadují další interakci (stavba budovy) zezelenají a fungují jako tlačítko. Ostatní akce, jako výroba jednotky, po dovršení 100% z intervalu výroby se dokončí automaticky a jednotka vyjede z továrny.



Obrázek 11.2 Návrh uživatelského rozhraní uvnitř hry

Vzhledem ke charakteru problému je další uživatelské rozhraní v podobě hlavního menu, nastavení a spuštění hry popsáno v uživatelské dokumentaci a ukázáno v rámci demonstračních videí.

Lehce problematická část je nastavení a spuštění síťové hry – současné řešení předpokládá, že síťová hra bude realizována centrálním serverem na veřejném internetu – tak, jak to je u moderních her běžné. Neboť se ale stále jedná o prototyp, tak pro síťovou hru je nutné spustit server, odblokovat příchozí port a nastavit správnou IP adresu v nastavení.

Kapitola 12
Spustitelné balíčky

Kapitola se zabývá výrobou samostatně spustitelných balíčků s integrovaným prostředím jazyka Python se všemi nezbytnými závislostmi.

12.1 Možnosti spouštění

Prostředí jazyka Python je samo o sobě snadno dostupné na platformě Linux (např. přes balíčkovací systém apt) a také na platformě MS Windows (např. rámci Microsoft store). Je i pravděpodobné, že uživatel bude mít na své platformě prostředí jazyka Python již nainstalované. Je ale rizikové na toto spoléhat – může nastat nekompatibilita verzí, mohou se stát zastaralé některé dílčí balíčky a postup instalace může být zbytečně komplikovaný. Python sice podporuje konfigurace virtuálního prostředí specifických verzí balíčků (conda), nicméně i tento přístup může být rizikový z pohledu snadné přenositelnosti. Ideální možností by bylo vytvoření samostatného spustitelného balíčku s již integrovaným prostředím jazyka Python s potřebnými závislostmi.

12.2 Postup kompletace balíčků

Python je otevřený interpretovaný Jazyk, pro který lze psát rozšíření v C/C++, nebo ho použít jako interpretr v rámci projektu napsaném v jazyce nižší úrovně. Tato skutečnost nahrává výrobě samostatně spustitelného balíčku. Vstupním bodem je kód, který implementuje hlavičkový soubor *Python.h* společně s nezbytnými závislostmi (dynamické knihovny .so nebo .dll). Následuje inicializace prostředí jazyka Python a spuštění vstupního modulu *main*.

Tímto prvním krokem nicméně pouze jiným způsobem spouštíme interpreter jazyka Python – program stále vyžaduje původní zdrojové kódy všech modulů – což není pro balíček distribuovaný koncovému uživateli žádoucí (uživatel by mohl zdrojové kódy měnit). V případě menších programů se nabízí metoda *freeze* – která spojí všechny zdrojové kódy do jednoho modulu (souboru) jazyka Python, který se následně zkompiluje do C/C++ pomocí nástroje Cython a výsledek se spouští v rámci funkce *main*. Nejedná se ovšem o nijak standardní řešení a u větších projektů to není reálné – nelze se vyhnout načítání jednotlivých modulů. Zde nastává poměrně problematická část, jakým způsobem interpretru jazyka Python „podstrčit“ již zkompilovaný a integrovaný modul uvnitř binárky – a ačkoliv se projekt tímto problémem zabýval a výsledek byl kromě malých detailů funkční, tak se stále nejedná o standardní a jisté řešení (řešení bylo na bázi *cdef extern* entry-pointů modulů a vlastních module-laderů v jazyce Python [20]). Pro finální řešení je tedy nakonec použit zavedený nástroj PyInstaller [22].

PyInstaller je distribuován jako balíček jazyka Python. Klíčovým předpokladem před kompletací balíčku hry je vytvoření virtuálního prostředí jazyka Python (venv), které je odstíněno od již nainstalovaných modulů v systému a tváří se jako nová instalace. V tomto prostředí se nainstalují nezbytné závislosti nutné pro spuštění hry včetně nástroje PyInstaller – bez tohoto kroku by byly zahrnuty všechny Python balíčky nainstalované v OS a výsledný soubor by byl zbytečně velký a spouštěl by se velice pomalu díky načítání nabytečných závislostí.

Samotné vytvoření balíčku je už poměrně snadné – definuje se vstupní bod programu a nástroj se pokusí detekovat všechny použité moduly a výsledek sestaví do jednoho balíčku. Pro závislosti, které nejsou snadno detekovatelné (jsou např. importované z modulu nástroje Cython), se použije přepínač *–hidden-import*. Pro připojení zkompilovaných modulů (dynamických knihoven .so nebo .dll) slouží přepínač *–collect-binaries*. Demonstrace použití ukázána ve fragmentu 12.1.

Výpis kódu 12.1 Použití nástroje PyInstaller

```
pyinstaller gam_main.py -F --hidden-import cython --hidden-import deepdiff^
--hidden-import lib.pyshaders^
--hidden-import game.image.texturebuilder^
--collect-binaries game.image.texturebuilder^
--hidden-import game.image.imagecollector^
--collect-binaries game.image.imagecollector^
...
```

Výsledný balíček je po otestování funkční na platformě, která žádnou instalaci jazyka Python dostupnou nemá (pro MS Windows je nutné mít nainstalovaný Microsoft Visual C++ Redistributable, který již je standardem).

Kapitola 13
Testování

Kapitola diskutuje úroveň funkčnosti a chybovosti výsledného produktu hry a popisuje metody testování, které byly použity. Součástí je také detailní přehled problémů vycházející z uživatelského testování a způsoby, kterými byly vyřešeny.

13.1 Problém testování počítačové hry

Není sporu o tom, že počítačovou hru je nutné testovat jako jakýkoliv jiný software. Přístup k testování hry má nicméně zásadní specifika. Při srovnání s testováním informačního systému, který plní jasně definovaný set případů užití, lze první rozdíl spatřit s ohledem na možná rizika – nemožnost v informačním systému realizovat use-case, nebo zapřičinění ne-konzistence dat, může být prakticky ekvivalentní s nefunkčností celého systému. V případě počítačové hry určitá drobná ne-funkcionalita nebo závada (pro hru typicky *glitch*) nemusí nutně znamenat nefunkčnost celé hry. V rámci hry navíc probíhá interakce mnoha objektů v rámci složitých pravidel (v případě 3D hry už běžně interakce v rámci fyzikálního světa) a plnění use-casů zahrnující „hraní hry“ nepředstavuje nikdy stejnou sadu interakcí a stejný výsledek – každá odehraná hra je jiná. Lze tedy nakreslit určitou nepřímou úměru mezi možností hrát hru a její hratelností a počtem chyb.

Jednou ze skutečností, která hraje v neprospěch testování je situace na trhu počítačových her – které jsou především spotřebním zbožím ve formě „zahraj a zapomeň“ a her, které jsou dlouhodobě vyvíjeny a vylepšovány příliš mnoho není. Vývojář je obvykle nucen včas dodávat výsledek na úkor kvality kódu a důkladného testování. Nelze popřít, že tento projekt díky své ambicióznosti tento stav z části simuluje.

13.1.1 Unit testy

Jednotlivé herní komponenty by měly být ideálně navrženy testovatelné se snahou o dodržováním principů GRASP a SOLID [31, 32] – lze říci, že kód, který je od začátku testovaný a jsou k nim psány unit-testy, tak kód k těmto principům nutně inklinuje. V případě rozšíření týmu vývojářů je psaní unit-testů přímo nezbytné.

Na druhé straně je třeba zdůraznit, že i důsledné psaní jednotkových testů neeliminuje riziko chyb v celém systému – a mnohé ani nelze tímto způsobem odhalit, neboť kombinací interakcí jednotlivých komponent v rámci počítačové hry je příliš mnoho. Případně se může stát, že jsou testovány kombinace, které v rámci celku příliš velkou relevanci nemají.

13.1.2 Náhodné testování

Jeden z možných přístupů, který projekt používá, je náhodné testování. Tento způsob slouží k odhalení klíčových chyb a nedostatků. Způsob stojí na předpokladu, že program odděluje vykreslování grafiky a herní logiku. Následně jsou definovány předpoklady, které jsou nežádoucí. Poté se spustí herní logika bez časování a s náhodnými vstupy a sleduje se naplnění předpokladů. Příkladem může být testování příjezdu skupiny jednotek na určené místo:

- Vytvoříme herní mapu.
- Umístíme skupinu jednotek na náhodné dostupné místo.
- Zadáme příkaz skupině na přesun na jiné náhodné místo.
- Kontrolujeme, zda se některá z jednotek nezasekla, zda nebyla nalezena cesta, zda vzdálenost od jednotek není příliš velká apod.
- Pokud po n iteracích nenastal nežádoucí předpoklad, pak lze testovaný mechanismus hry označit za funkční.
- Není tím samozřejmě vyloučeno, že v jiné konfiguraci se nemůže problém detekovat. Nebo se objeví po $n + x$ iteracích.

Tomuto způsobu testování navíc nahrává předpoklad tohoto projektu, že je herní prostor deterministický a nespoléhá na náhody ani jiné vlivy – lze tedy nežádoucí stav nasimulovat a jednoznačně opravit.

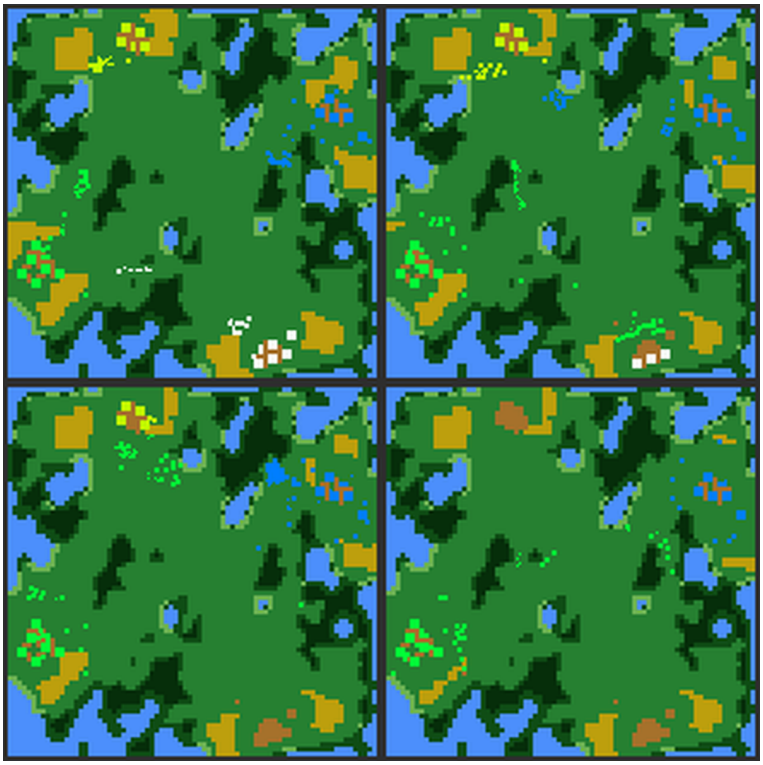
13.1.3 Testování pomocí skriptované umělé inteligence

Při testování hry lze využít přítomnost umělé inteligence. Jedním z jistých způsobu testování je spouštění různých konfigurací mapy s umělou inteligencí a pozorování případných defektů. Pro odhalení konkrétních problémů je ale třeba taktéž zanést určité nežádoucí předpoklady. Bez nich se jedná o velmi hrubé testování se zjištěním, že hra „něco dělá“ – a tedy když nepadá, tak funguje. Předpoklad, že jedna strana AI vyhraje, nemusí být vždy platný – neboť mohou dojít zdroje a hra skončit nerozhodně. V každém případě lze tímto způsobem odhalit chyby typu přetečení bufferu, neplatný ukazatel apod. – tedy jakékoliv příčiny pádu programu, neboť hra AI proti AI projde velmi rozsáhlý stavový prostor.

Důležitou částí je testování umělé inteligence jako takové – a to především pozorováním, zda se AI chová podle očekávání. Bylo by velmi zdlouhavé toto pozorování provádět v reálném čase. Zde pro testování pomůže opět možnost spustit herní logiku bez vykreslování a časování a pro zobrazení stavu hry pomůže projekce na mini-mapu, jak je znázorněno na obrázku 13.1, kde barevné tečky uvnitř prostředí mapy znázorňují budovy a jednotky jednotlivých hráčů, kteří jsou ovládáni umělou inteligencí. Animovaný průběh hry je uložen na přiloženém médiu.

13.2 Uživatelské testování

Velmi důležitou součástí je uživatelské testování. Absence uživatelského testování v případě informačního systému je sice tragická, nicméně uživatel se v konečném důsledku bude muset se systémem seznámit, jinak nebude mít možnost realizovat scénáře, ke kterým je systém určen. V případě počítačové hry je uživatelské testování zásadnější – hru, která nepůjde snadno ovládat a nebude nijak zábavná, jednoduše nikdo hrát nebude. Bylo by ideální definovat cílové skupiny a testovat na jejich základě – nicméně vzhledem k cílům a povaze projektu to není příliš realistické. Přesto se testování zaměřilo na typ hráčů, pro které hraní her není nic neznámého. Testování na



Obrázek 13.1 Mini-mapy průběhu hry umělé inteligence

ne-hráčích nebo občasných hráčích (tzv. casual hráčích) naopak příliš velký smysl nedává - neboť jsou vzdáleni od cílové skupiny hráčů strategií a hra je pro ně příliš složitá.

13.2.1 Výstup uživatelského testování

Hra byla testována na čtyřech subjektech – kde všichni zúčastnění měli s hraním her tohoto typu zkušenosti a někteří byli přímo aktivní hráči. Test probíhal způsobem, že byla nejdříve hra ukázána a vysvětlena a poté subjekt hrál hru proti umělé inteligenci, během které byl pozorován, zda se daří plnit scénáře a cíle hry a kde nastávají problémy. V další fázi byla vyzkoušena hra více hráčů po síti. Poznatky jsou rozepsány v sjednocené podobě, neboť se opakovaly. Po úpravách u některých subjektů byl proveden test znovu s pozitivní zpětnou vazbou, že změny přispěly k lepší hratelnosti hry.

- Problém s ovládání myši – díky častému pohybu myši se chybně detekuje dvojklik nebo klik a spustí se režim plošného označení. **Vyřešeno**.
- Problém s umístěním budov – chybí srozumitelné upozornění, zda budovu lze postavit nebo nelze. **Vyřešeno** – režim umístění budovy nezmizí při pokusu o umístění a při chybném umístění zobrazí upozornění.
- Postřeh s ne příliš srozumitelným označením budov – není zřejmé k čemu je konkrétní budova určena. **Částečně vyřešeno** – nabídka budov má ikonky a stejné ikonky jsou vidět při označení budovy. Ikonky nicméně nelze integrovat tak, aby byly viditelné stále.
- Fronta stavby budov není srozumitelná – hráč má tendenci stavět budovu opakovaně, přestože již je připravena ke stavbě. **Vyřešeno** – budovu lze v jeden okamžik stavět pouze jednu. Při dalším pokusu se zobrazí upozornění.

- Bylo by vhodné se přesouvat po mapě kliknutím do mini-mapy a rolování při najetí kurzorem na kraj mapy. **Vyřešeno** – mini-mapa zobrazuje, na jakém místě je hráč narolován a také se lze přesouvat kliknutím do mini-mapy.
- Jednotky by měly samy útočit na nepřátelské v dosahu. **Vyřešeno** – automatický útok povolen pro lidské hráče a přidán příkaz jednotkám *hold* – kdy takto označené jednotky neútočí a stojí na místě.
- Doporučení lepšího ovládání jednotek – sloučit příkazy *attack* a *attack move* a cílený útok provést po kliknutí na nepřátelskou jednotku. **Vyřešeno** – nabídka příkazů je *move*, *attack*, *hold* a *stop* (včetně klávesových zkratek). Rozlišení příkazů *attack* a *attack-move* je zrušeno.
- V nabídce továrny na výrobu jednotky by pro přehlednost měly být ikonky nebo obrázky. Také by pomohla nápověda s bonusy. **Vyřešeno** – nabídka výroby jednotek obsahuje jejich obrázky. Doděláno také informační okno, které zobrazuje základní údaje jednotky (poškození, výdrž) a bonusy proti ostatním jednotkám.
- Kliknutí na jednotku s klávesou CTRL by mělo vybrat všechny jednotky stejného typu analogicky, jak se děje pomocí dvojkliku. **Vyřešeno**.
- Mělo by se ukázat upozornění na nedostatek surovin při pokusu o výrobu. **Vyřešeno**.
- Chyba, kdy při nastavení „human“ jako druhého hráče je po spuštění hry zacílen první hráč s umělou inteligencí. **Vyřešeno**.
- Hra je příliš rychlá – hráč má příliš brzy mnoho zdrojů a nestíhá je utrácet. **Vyřešeno** – těžba zdrojů je mnohem pomalejší (hru lze nicméně spustit s režimem rychlé těžby).
- Fronta stavby jednotek je nepřehledná a nelze zrušit omylem zadaný příkaz ke stavbě jednotky nebo budovy. **Vyřešeno** – fronta stavby obsahuje také obrázky jednotek a více příkazů stejného typu se sjednocuje do jednoho tlačítka. Taktéž lze kliknutím zadaný příkaz zrušit.
- Nabídka stavby budov nebo jednotek by se měla automaticky aktualizovat po vylepšení technologické úrovně. **Vyřešeno**.
- Ve hře by měla hrát hudba. **Nelze snadno vyřešit** – najít vhodnou hudbu s odpovídající licencí by bylo pracné.
- Hra je pomalá na slabším PC a trhá se. **Problematické** – částečné optimalizace byly ještě udělány, nicméně obecně je třeba většího úsilí věnovaného optimalizacím pomocí Cythonu a to již není v této fázi práce splnitelné díky nedostatku času. Cíl je to nicméně reálný a v případě dalšího vývoje nezbytný.
- Na pomalejších PC je problém s efektem dojezdu při rolování po mapě. **Vyřešeno** – efekt dojezdu zrušen.
- Hru proti AI lze vyhrát defenzivní strategií s postupnou výrobou co nejvíce jednotek. **Problematické** – vyžaduje další vývoj a odladění mechanik hry, které by vysoké množství jednotek nějakým způsobem penalizovaly.
- Hru lze vyhrát maximalizací stavby rafinerií na těžbu zdrojů. **Částečně vyřešeno** – maximální počet rafinerií a těžičů surovin je omezen. Přidána také je možnost cílené destrukce budovy – pro poskytnutí možnosti postavit rafinerii blíže ke zdrojům surovin. Pro hru by nicméně bylo lepší rozšíření těžby surovin o další mechaniky.
- Pád hry nebo zamrznutí na některém testovaném PC – **Částečně vyřešeno** – některé problémy odladěny nicméně plná oprava vyžaduje další testování.

Výstup uživatelského testování lze stručně shrnout tak, že se hra líbila a bylo zábavné ji hrát. Pozitivní je přehlednost a relativní jednoduchost. Problémem pak je obecně malý rozsah hry a málo herního obsahu – hráč tedy nemá motivaci hru hrát příliš dlouho. Určitý potenciál hra nicméně má.

Kapitola 14
Závěr

Vývoj náročného projektu se podařilo dokončit s výsledkem funkčního a hratelného prototypu. Hra disponuje ucelenou grafikou a přehledným grafickým prostředím s funkčním generátorem mapy. Ve hře funguje stavba budov základny, těžba surovin a výroba jednotek. Je implementován dostatečně efektivní algoritmus hledání cest a objekty lze intuitivně navigovat po herní mapě a hru ovládat a hrát. Jsou funkční interakce jednotek, jako je střelba nebo těžba surovin, s doprovodnými částicovými grafickými efekty.

Podařilo se dokončit a funkční client-server mechanismus pro síťovou hru více hráčů fungující na bázi deterministických simulací, kde je odladěn a ošetřen problém de-synchronizace. Síťová hra byla úspěšně otestována i přes internet se spuštěným veřejným serverem.

Také se podařilo vytvořit skriptovanou umělou inteligenci, která simuluje chování lidského hráče a provádí stavbu základny, těžbu zdrojů, výrobu jednotek, obranu základny a útoky. Provádí také analýzu viditelného okolního prostředí a snaží se přizpůsobit svoji reakci aktuálním podmínkám ve hře.

Prototyp podporuje hru více než dvou hráčů a umožňuje různé kombinace lidských hráčů a umělé inteligence. Pro snadné spuštění hry jsou připravené a otestované balíčky pro platformy Windows a Linux. Výsledek byl úspěšně otestován s několika uživatelskými testery.

Vývoj hry pomohl autorovi nabýt další cenné zkušenosti v této oblasti.

14.0.1 Možnosti dalšího vývoje

Ačkoliv vývoj dosáhl vytyčených cílů a je k dispozici funkční hra, tak se jedná o prototyp a pro finální konkurence-schopnou verzi hry by byla potřeba další práce. Dílčí možnosti dalšího vývoje jsou mimo jiné uvedeny v rámci jednotlivých kapitol.

Hra vyžaduje další optimalizace a to ideálně přepis všech modulů do kódu nástroje Cython, které jsou stále v čistém jazyce Python, včetně dílčích optimalizací. Důležitým prvkem také je odblokování limitace GIL.

Obsah hry jako takový je v rámci prototypu limitovaný a obsahuje málo budov, několik jednotek a omezené herní principy a mechaniky. Dalším obsahem by mohlo být pestřejší a komplikovanější prostředí mapy, více jednotek (letadla, lodě), rozvinutější základna s rozšířenými možnostmi stavby. Více herních mechanik a režimů hry (single-player mise, tower-defense mód apod.). Výroba dalšího obsahu je nicméně velmi časově náročná.

Grafický koncept hry není překážkou a má pozitivní ohlasy především od starších lidí vzhledem ke svému nostalgickému nádechu, který připomíná hry z devadesátých let.

Literatura

[1] *ŽÁRA, Jiří. Moderní počítačová grafika. 2.*, přeprac. a rozš. vyd. Brno: Computer Press, 2004. ISBN 8025104540.

[2] *AdamSawicki: How to Correctly Interpolate Vertex Attributes on a Parallelogram Using Modern GPUs?* [online]. [cit. 2022-04-21]. Dostupné z: `https://asawicki.info/news_1721_how_to_correctly_interpolate_vertex_attributes_on_a_parallelogram_using_modern_gpus`

[3] *Python 3: Documentation* [online]. [cit. 2022-04-21]. Dostupné z: `https://docs.python.org/3/`

[4] *Python: Global Interpreter lock* [online]. [cit. 2022-04-21]. Dostupné z: `https://wiki.python.org/moin/GlobalInterpreterLock`

[5] *Cython: C-Extensions for Python* [online]. [cit. 2022-04-21]. Dostupné z: `https://cython.org/`

[6] *Envato Tuts+: Generate Random Cave Levels Using Cellular Automata* [online]. [cit. 2022-04-22]. Dostupné z: `https://gamedevelopment.tutsplus.com/tutorials/generate-random-cave-levels-using-cellular-automata--gamedev-9664`

[7] *Wikipedia: Integer square root using binary search* [online]. [cit. 2022-04-21]. Dostupné z: `https://en.wikipedia.org/wiki/Integer_square_root#Algorithm_using_binary_search`

[8] *PyPy: A fast, compliant alternative implementation of Python* [online]. [cit. 2022-04-21]. Dostupné z: `https://www.pypy.org/`

[9] *Numba: Numba makes Python code fast* [online]. [cit. 2022-04-21]. Dostupné z: `https://numba.pydata.org/`

[10] *Pyglet: cross-platform windowing and multimedia library for Python* [online]. [cit. 2022-04-21]. Dostupné z: `https://pyglet.readthedocs.io/en/latest/`

[11] *Khronos: OpenGL® and OpenGL® ES Reference Pages* [online]. [cit. 2022-04-21]. Dostupné z: `https://www.khronos.org/registry/OpenGL-Refpages/`

[12] *Wikipedia: Quadtree* [online]. [cit. 2022-04-21]. Dostupné z: `https://en.wikipedia.org/wiki/Quadtree`

[13] *Envato Tuts+: Understanding Goal-Based Vector Field Pathfinding* [online]. [cit. 2022-05-09]. Dostupné z: https://gamedevelopment.tutsplus.com/tutorials/understanding-goal-based-vector-field-pathfinding--gamedev-9007

[14] *Wikipedia: A* search algorithm* [online]. [cit. 2022-04-21]. Dostupné z: https://en.wikipedia.org/wiki/A*_search_algorithm

[15] *Adrien Treuille, Seth Cooper, and Zoran Popović. 2006. Continuum crowds. ACM Trans. Graph. 25, 3* (July 2006), 1160–1168. DOI:https://doi.org/10.1145/1141911.1142008

[16] *Crafter on Games: Floating Point Determinism* [online]. [cit. 2022-04-21]. Dostupné z: https://gafferongames.com/post/floating_point_determinism/

[17] *GeeksForGeeks: Bresenham's Line Generation Algorithm* [online]. [cit. 2022-04-21]. Dostupné z: https://www.geeksforgeeks.org/bresenhams-line-generation-algorithm/

[18] *Gaffer On Games: UDP vs. TCP* [online]. [cit. 2022-04-21]. Dostupné z: https://gafferongames.com/post/udp_vs_tcp/

[19] *Glooey: An object-oriented GUI library for pyglet* [online]. [cit. 2022-04-21]. Dostupné z: https://glooey.readthedocs.io/en/latest/

[20] *Stackoverflow: Collapse multiple submodules to one Cython extension* [online]. [cit. 2022-04-21]. Dostupné z: https://stackoverflow.com/questions/30157363/collapse-multiple-submodules-to-one-cython-extension

[21] *Github: GameNetworkingResources* [online]. [cit. 2022-03-05]. Dostupné z: https://github.com/ThusWroteNomad/GameNetworkingResources

[22] *PyInstaller: Using PyInstaller* [online]. [cit. 2022-04-21]. Dostupné z: https://pyinstaller.org/en/stable/usage.html

[23] *Wayward Strategy: What Is A Real-Time Strategy Game? An Exploration and Definition* [online]. [cit. 2022-05-03]. Dostupné z: https://waywardstrategy.com/2015/09/25/what-is-an-rts-game/

[24] *Blizzard Entertainment Inc.: StarCraft: Remastered* [online]. [cit. 2022-05-03]. Dostupné z: https://starcraft.com/en-gb/

[25] *EA Official Site: Command & Conquer Remastered* [online]. [cit. 2022-05-03]. Dostupné z: https://www.ea.com/games/command-and-conquer/command-and-conquer-remastered

[26] *Age of Empires II: Definitive Edition* [online]. [cit. 2022-05-03]. Dostupné z: https://www.ageofempires.com/games/aoeiide/

[27] *Department of Play: The Mathematics of Game Balance* [online]. [cit. 2022-05-04]. Dostupné z: https://departmentofplay.net/the-mathematics-of-balance/

[28] *GeeksForGeeks: Markov Decision Process* [online]. [cit. 2022-05-04]. Dostupné z: https://www.geeksforgeeks.org/markov-decision-process/

[29] *Wikipedia: Monte Carlo method* [online]. [cit. 2022-05-04]. Dostupné z: https://en.wikipedia.org/wiki/Monte_Carlo_method

[30] *Envato Tuts+: Goal Oriented Action Planning for a Smarter AI* [online]. [cit. 2022-05-04]. Dostupné z: https://gamedevelopment.tutsplus.com/tutorials/goal-oriented-action-planning-for-a-smarter-ai--cms-20793

[31] MARTIN, Robert C. *Čistý kód: [návrhové vzory, refaktorování, testování a další techniky agilního programování]*. Brno: Computer Press, 2009. ISBN 978-80-251-2285-3.

[32] PECINOVSKÝ, Rudolf. *Návrhové vzory: [33 vzorových postupů pro objektové programování]*. Brno: Computer Press, 2007. ISBN 978-80-251-1582-4.

[33] *Wikipedia: Breadth-first search* [online]. [cit. 2022-05-09]. Dostupné z: `https://en.wikipedia.org/wiki/Breadth-first_search`

[34] *Game Developer: 1500 Archers on a 28.8: Network Programming in Age of Empires and Beyond* [online]. [cit. 2022-05-09]. Dostupné z: `https://www.gamedeveloper.com/programming/1500-archers-on-a-28-8-network-programming-in-age-of-empires-and-beyond`

[35] *Wikipedia: Kernel (image processing)* [online]. [cit. 2022-05-09]. Dostupné z: `https://en.wikipedia.org/wiki/Kernel_(image_processing)`

Příloha A

Uživatelská příručka

V této části je obsažena příručka s návodem, jak se program spustí, jak se zahájí hra jednoho hráče proti umělé inteligenci a jak spustit server pro hru více hráčů a následné připojení do hry. Je zde také uveden návod samotné hry.

Veškerý potřebný obsah je umístěn na přiloženém médiu, který je dostupný také na následujících umístění:

Webová adresa – na adrese `http://fit.mazec.org/rts` je snadno přístupný balíček s výsledným prototypem hry a prezentační videa. Dostupnost této adresy je garantována do 24.6.2022.

Repozitář GitLab fakulty ČVUT FIT – v repozitáři `https://gitlab.fit.cvut.cz/hrabajak/bp_realtime_strategy_game` jsou dostupné zdrojové soubory. Není zde uveden dodatečný obsah přiloženého média.

A.1 Spuštění programu hry

Spustitelné balíčky jsou umístěny v adresáři *dist*. Na platformě MS Windows se hra spustí pomocí *game_win.exe* a na platformě Linux pomocí *./gamerts_linux*. Možné problémy se spuštěním a jejich řešení jsou uvedeny v sekci A.6.

A.2 Hlavní menu

Po spuštění hry se zobrazí hlavní menu, jak je znázorněno na obrázku A.1.

Standard game – spuštění hry jednoho hráče.

Online server game – menu s možností připojení do serverové hry a její vytvoření.

Settings – nastavení s hlasitostí zvuku, rozlišením obrazovky a IP adresou serveru. Alternativně lze zapnout možnost *Allow cheats* – která umožní vkládat jednotky na místo kurzoru pomocí číselných kláves a numerické klávesnice.

About – zobrazení informací o hře.

Units overview – zobrazení přehledu jednotek a jejich parametrů (kliknutím na jednotku lze zobrazit detaily).

Quit – ukončení hry.

Obrázek A.1 Hlavní menu hry

A.3 Spuštění nové hry

Dialog s vytvořením nové hry, po zvolení možnosti *Standard game*, je znázorněn na obrázku A.2.

Player count – nastavení počtu hráčů.

Map size, Seed – nastavení rozměru mapy a čísla *seed*, které určuje vstup náhodného generátoru.

Fog – nastavení mlhy. Při zvolení *disabled* je mlha zcela vypnutá a vše je viditelné.

Tabulka s hráči – u každého hráče se nastavuje postupně: typ civilizace (*first type* – tanky, klasické zbraně, *second type* – energetické zbraně), lidský hráč nebo AI (podle obtížnosti), barva hráče, množství zdrojů, které jsou ihned dostupné po startu hry.

Quick progress – po zapnutí je produkce jednotky nebo výroba budovy okamžitá.

Quick build – po zapnutí je budova po umístění hned postavená (nečeká se na práci stavitele).

Quick harvest – rychlejší těžba surovin.

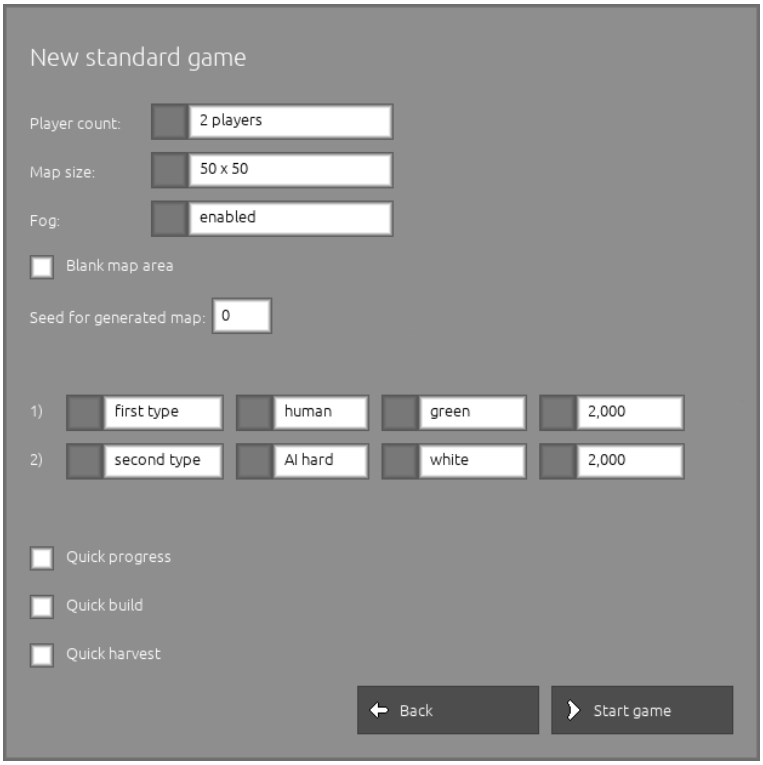
A.4 Hra více hráčů

Dialog s vytvořením a připojením do síťové hry, po zvolení možnosti *Online server game*, je znázorněn na obrázku A.3. Program musí mít v nastavení *Settings* zadanou korektní IP adresu spuštěného serveru.

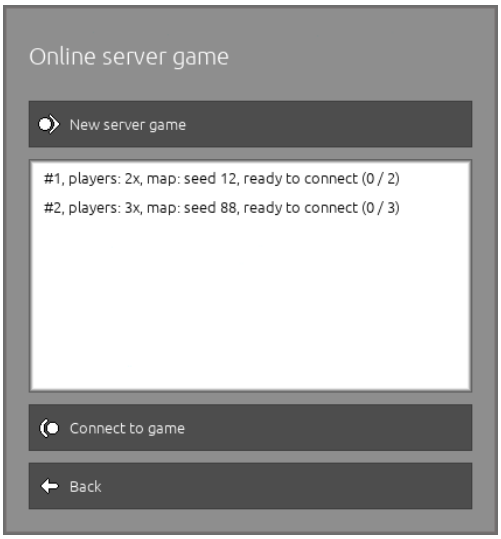
Server se na platformě MS Windows spustí pomocí *server_win.exe* a na platformě Linux pomocí *./server_linux* (řešení problémů s připojením je popsáno v sekci A.6).

New server game – založení nové hry na serveru. Následuje dialog, který je stejný jako v případě spuštění nové hry.

Connect to game – připojení do zvolené hry v nabídce. Vybraná hra musí být ve stavu *ready to connect*. Před spuštěním hry se čeká na připojení všech hráčů.



Obrázek A.2 Spuštění nové hry



Obrázek A.3 Hra více hráčů

A.5 Přehled budov základny

Budovy základny, které je možné postavit, jsou znázorněny na obrázku A.4.

Construction base – (číslo 1) hlavní budova s nabídkou výstavby dalších budov (maximální technologická úroveň je 1 – vylepšení umožní stavbu silnějších obranných věží).

Refinery – (číslo 2) budova rafinerie, která umožňuje těžbu surovin. Stavba jedné rafinerie odemkne možnost stavby dalších budov.

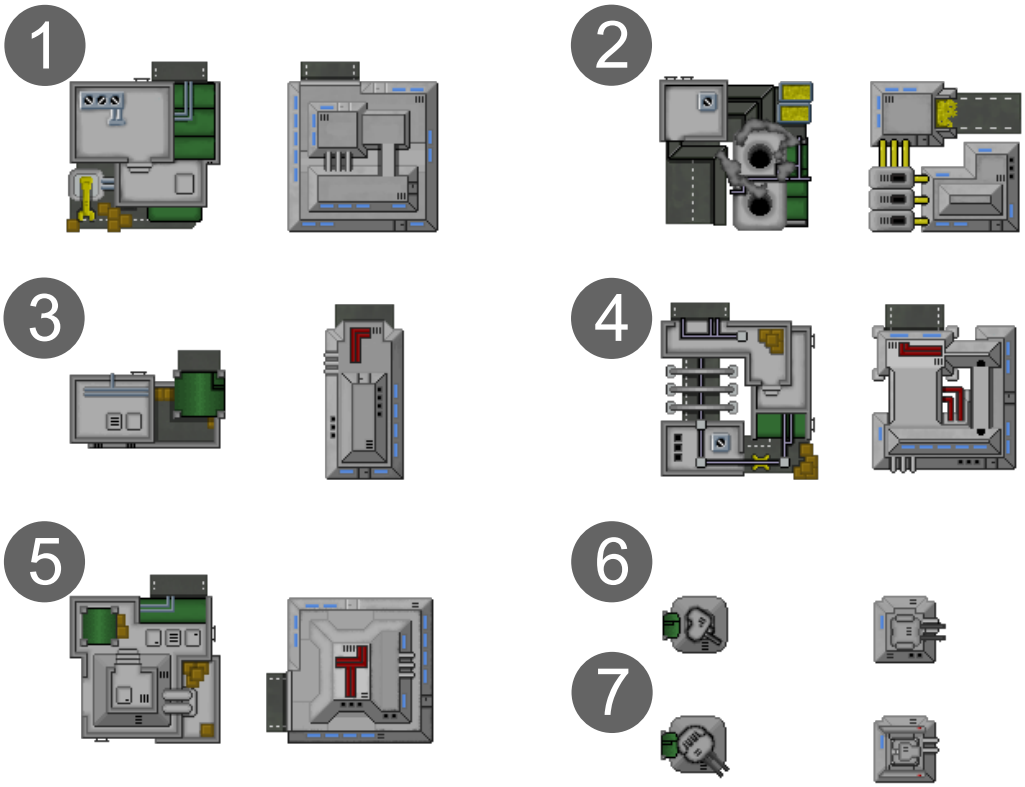
Barracks – (číslo 3) budova na produkci pěších jednotek (maximální technologická úroveň je 2).

Factory – (číslo 4) budova na produkci mechanizovaných jednotek (maximální technologická úroveň je 3).

Laboratory – (číslo 5) budova laboratoře s nabídkou vylepšení technologické úrovně ostatních budov.

Turret – (číslo 6) základní obranná věž.

Advanced turret – (číslo 7) pokročilá obranná věž (vyžaduje první technologickou úroveň hlavní budovy).



Obrázek A.4 Nabídka budov základny

A.6 Možné problémy a jejich řešení

Pro spuštění na platformě MS Windows je nutné mít nainstalované Microsoft Visual C++ Redistributable: `https://docs.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist?view=msvc-170`.

Běh hry (nikoliv serveru) vyžaduje podporu OpenGL verze alespoň 3.1 (vydaná v roce 2009).

Hra více hráčů vyžaduje povolit naslouchání portu 65111 na firewallu. V případě přístupu z vnější sítě je potřeba pomocí překladu adres (NAT) přesměrovat port 65111 tak, aby byl přístupný z veřejné adresy.

Pokud je na serveru nainstalováno více síťových adaptérů, tak může pomoci zadat explicitní adresu v kolonce „Listen IP“ v nastavení hry (soubor nastavení je sdílený mezi serverem a klientem).

A.7 Ovládání hry

Jednotky je možné ovládat následujícími způsoby:

Označení jednotky – kliknutím levým tlačítkem myši na jednotku dojde k jejímu označení a poté je možné jednotce zadávat příkazy.

Označení více jednotek – tažení se stisknutým levým tlačítkem myši vytvoří výběrový obdélník, pomocí kterého je možné označit více jednotek zároveň – příkazy jsou pak zadávány všem označeným jednotkám.

Dvojklik na jednotku – dvojklik označí všechny jednotky stejného typu na obrazovce (stejnou funkci provede jednoduché kliknutí se stisknutou klávesou *CTRL*).

Příkaz označeným jednotkám – kliknutím pravým tlačítkem myši na místo na mapě provede přesun jednotek. Kliknutí pravým tlačítkem se stisknutou klávesou *CTRL* provede přesun s útokem. Kliknutí pravým tlačítkem na nepřátelskou jednotku provede cílený útok na tuto jednotku.

Přesun po mapě – tažení myší se stisknutým pravým tlačítkem – nebo klávesy šipek. Další možností je kliknutí do oblasti mini-mapy.

Chat – stisknutí klávesy *ENTER* během hry otevře dialogové okno s možností zadat textový vzkaz, který se ukáže ostatním hráčům.

A.8 Spuštění hry přímo v prostředí jazyka Python

Hra je vyvíjena a odladěna na verzi Python 3.6. Jako první krok je třeba uvnitř adresáře *src* vytvořit virtuální prostředí:

```
cd src
/usr/bin/python3.6 -m venv rts
source rts/bin/activate
```

Poté nainstalovat potřebné balíčky:

```
python -m pip install wheel
python -m pip install pyglet==1.5.17
python -m pip install pyshaders==1.4.1
python -m pip install ordered_set
python -m pip install cython
python -m pip install deepdiff
```

Spustit kompilaci nástrojem Cython:

```
python setup_gam_b.py build_ext --inplace
```

Spustit hru:

```
python gamerts.py
```

Nebo spustit server:

```
python server.py
```

A.9 Použité nástroje a zdroje

Pro vývoj herního prototypu a dalších souvisejících částí byly použity následující nástroje a zdroje:

Python – jazyk Python optimalizovaný pomocí nástroje Cython. Použité knihovny: Pyglet, PyShaders, OrderedSet, DeepDiff.

PyCharm IDE – vývojové prostředí použité pro psaní kódu.

Bash – linuxový shell použitý pro dodatečné kompilační skripty, spuštění apod.

GIT – systém pro správu verzí.

PyInstaller – nástroj pro vytvoření samostatně spustitelného balíčku.

Audacity – open source editor digitálního zvuku, který byl použitý pro lehkou korekci zvukových efektů.

Aseprite – grafický editor pro výrobu grafiky metodou pixel-art.

GIMP – rastrový grafický editor.

Freesound – portál s volně dostupnými zvukovými efekty (`https://freesound.org/`).

Obsah přiloženého média

- `readme.txt` ... stručný popis obsahu média
- `video` ... adresář s demonstračními videi
  - `stavba zakladny.mkv` ... návod na stavbu základny hráče
  - `hra proti 2 AI.mkv` ... demonstrace hry jednoho hráče proti 2 AI
  - `hra vice hracu po siti.mkv` ... demonstrace hry více hráčů po síti
  - `demonstrace efektu.mkv` ... ukázka efektů
  - `testovani hry 4 AI.mp4` ... demonstrace testování hry 4 umělých inteligencí
- `dist` ... adresář se spustitelnými balíčky s výslednou hrou
  - `gamerts_win.exe` ... hra pro platformu MS Windows
  - `gamerts_linux` ... hra pro platformu Linux
  - `server_win.exe` ... server pro platformu MS Windows
  - `server_linux` ... server pro platformu Linux
- `src` ... zdrojové kódy implementace
  - `readme.txt` ... stručný komentář ke zdrojovému kódu
- `latex` ... text práce
  - `ctufit-thesis.pdf` ... text práce ve formátu PDF