
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**
**BRNO UNIVERSITY OF TECHNOLOGY**

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**
**FACULTY OF INFORMATION TECHNOLOGY**

**ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ**
**DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA**


# HRA V OPENGL

**GAME IN OPENGL**

**BAKALÁŘSKÁ PRÁCE**
**BACHELOR'S THESIS**


**AUTOR PRÁCE** **PAVEL HRANÁČ**
**AUTHOR**

**VEDOUCÍ PRÁCE** **Ing. TOMÁŠ STARKA**
**SUPERVISOR**



**BRNO 2021**

**Vysoké učení technické v Brně**
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2020/2021

# Zadání bakalářské práce


Student: **Hranáč Pavel**
Program: Informační technologie
Název: **Hra v OpenGL**
**Game in OpenGL**
Kategorie: Počítačová grafika

Zadání:

1. Nastudujte OpenGL, GPUEngine a další potřebné knihovny.
2. Nastudujte tvorbu herní AI.
3. Implementujte vlastní hru s možností AI protivníka.
4. Vytvořte plakát nebo video k práci.

Literatura:

- Po dohodě s vedoucím.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Starka Tomáš, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 30. října 2020

# Abstrakt

Cílem této práce je vytvořit vlastní hru se soupeřem s umělou inteligencí. Teoretická část práce se zabývá především umělou inteligencí ve hrách obecně. Grafika hry je trojrozměrná a využívá API OpenGL a grafickou knihovnu GPUEngine. Vytvořená hra je tahová strategie podobná šachům, která nabízí počítačové soupeře různých obtížností na bázi vlastní modifikace strategie Minimax s komplexní heuristikou. Hra také umožňuje lokální hru více hráčů. Výsledky této práce mohou sloužit jako inspirace nebo základ při zkoumání umělé inteligence ve hrách.



# Abstract

The goal of this thesis is to create a custom game with an AI opponent. The theoretical part of the thesis deals mainly with artificial intelligence in games in general. The game's graphics are three-dimensional and use the OpenGL API and the GPUEngine graphics library. The created game is a turn-based strategy game similar to chess, offering computer opponents of various difficulties using a custom modification of the Minimax strategy with a complex heuristic. The game also offers local multiplayer. The results of this thesis can serve as inspiration or a basis for the study of artificial intelligence in games.



# Klíčová slova

Hra, 3D, Grafika, Umělá inteligence, AI, Herní AI, Minimax, A*, A star, OpenGL, GPUEngine, QtQuick, QML

# Keywords

Game, 3D, Graphics, Artificial intelligence, AI, Game AI, Minimax, A*, A-star, OpenGL, GPUEngine, QtQuick, QML

# Citace

HRANÁČ, Pavel. *Hra v OpenGL*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Starka

# Hra v OpenGL

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Tomáše Starky. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Pavel Hranáč
12. května 2021

## Poděkování

Rád bych poděkoval vedoucímu práce Ing. Tomáši Starkovi za jeho vstřícnost a jeho rady během seminářů a osobních konzultací.

# Obsah

# Seznam použitých zkratek

| | | |
|---|---|---|
| UI | – | Umělá inteligence |
| RTS | – | Real-Time Strategy |
| MOBA | – | Multiplayer Online Battle Arena |
| GUI | – | Graphical User Interface |

# Kapitola 1

# Úvod

Cílem této bakalářské práce je vytvořit videohru s protivníkem ovládaným počítačem. Téma práce jsem si vybral především kvůli mému osobnímu zájmu o témata navrhování her a umělé inteligence. Fakulta informačních technologií Vysokého učení technického v Brně bohužel v oblasti herního průmyslu zaostává oproti zbytku světa, jelikož nenabízí žádné povinné ani volitelné předměty, které by s tímto dlouhodobě rostoucím průmyslem přímo souvisely. Studenti pak nemají jinou možnost než se s tématem seznámit ve volném čase nebo v rámci bakalářských a diplomových prací.

Vytvoření i jednoduché hry je docela obtížný počin, ale díky rostoucí popularitě tohoto průmyslu a s tím i růstu snadno dostupných kvalitních zdrojů a nástrojů se vytváření stává s časem stále snazší.

Tato bakalářská práce se kromě samotného vývoje hry věnuje blíže teorii umělé inteligence ve hrách. Kapitola 2 stanovuje co se rozumí některými základními pojmy jako například umělá inteligence a herní umělá inteligence. Kapitola 3 zkoumá různé druhy umělých inteligencí ve hrách jako třeba autonomní chování herních entit nebo simulace živých hráčů. Kapitola 4 vysvětluje některé existující metody, které nejsou specifické pro konkrétní druh herní umělé inteligence.

Pro správné pochopení návrhu a implementace hry je potřeba se seznámit se základními principy vytvářené hry, o čemž pojednává kapitola 5. Detailnější popis hry lze nalézt v příloze B. V kapitole 6 budou rozebrány návrhy jednotlivých částí hry. Kapitola 7 se ponoří do implementačních detailů. Kapitoly 8 a 9 zhodnotí výsledek práce.

# Kapitola 2

# Základní pojmy

V této kapitole bude čtenář seznámen se základními pojmy související s tématem herní umělé inteligence. Tyto pojmy zahrnují umělou inteligenci, herní umělou inteligenci a prohledávání stavového prostoru.

## 2.1 Umělá inteligence obecně

Umělá inteligence je pojem zajímavý a překvapivě vágní. V závislosti na kontextu, úhlu pohledu může mít velmi odlišné definice. Mezi možné významy umělé inteligence patří například „obor informatiky“, „soubor algoritmů“, „inteligenci připomínající chování strojů“ nebo „stroje, které prošly Turingovým testem“.

Nejobecnějším přístupem je nahlížet na umělou inteligenci jako na obor informatiky. Jedná se o všezahrnující definici. Nezáleží na účelu ani implementaci daných algoritmů a metod, ale pouze na tom, jestli se formálně asociují s tímto odvětvím informatiky.

Naproti tomu jiné definice se snaží umělou inteligenci specifikovat trochu úžeji.

- Umělá inteligence je schopnost počítače nebo stroje vykonávat činnosti, u kterých se běžně předpokládá, že k provedení daných úkolů vyžadují inteligenci.[11]
- Umělá inteligence je o přimění počítačů myslet jako lidé a zvířata.[6]
- Výzkum umělé inteligence je studie „inteligentních agentů“: jakékoliv zařízení, které vnímá své okolí a vykonává akce, které maximalizují jeho šanci úspěšné splnit svůj cíl.[12]
- Umělá inteligence je schopnost systému správně interpretovat vnější data, učit se z nich a použít tyto znalosti k dosažení specifických cílů pomocí flexibilní adaptace.[7]
- Umělá inteligence je studie o tom, jak vytvořit stroje, které mají některé z vlastností lidské mysli jako porozumění jazyku, rozpoznání obrazu, řešení problémů a učení se.[1]

Většina zdrojů má svou vlastní variantu definice umělé inteligence. V průběhu času se objevují nové definice, které posouvají laťku toho, co se dá počítat za UI a tím se snaží postavit některé starší algoritmy mimo tento obor. Zda se těmto algoritmům pak dá ještě říkat umělá inteligence nebo ne, je sice sporné a matoucí, ale ve výsledku jde jen o jméno a tyto algoritmy plní svůj účel ať už se jim říká jakkoliv.

[1] Cambridge Dictionary. Dostupné z: https://dictionary.cambridge.org/dictionary/english/artificial-intelligence

## 2.2 Herní umělá inteligence

Jedno z odvětví umělé inteligence je herní umělá inteligence neboli umělá inteligence ve hrách. Zmíněné hry jsou zpravidla elektronické videohry, ale algoritmy pro hraní deskových her jako šachy se také počítají do této oblasti. Opaku herní UI se někdy říká akademická UI (anglicky academic AI).

Ale ani v tomto případě není hranice mezi těmito druhy tak jednoznačná. Právě třeba UI pro hraní deskových her vykazují vlastnosti obou druhů. UI hrají hry, ale využívají k tomu metody akademické UI jako třeba evoluční algoritmy nebo strojové učení. Dalšími příklady překlenutí této hranice jsou umělé inteligence OpenAI Five [9] hrající online hru Dota 2 a AlphaStar [1] hrající hru Starcraft 2.

Nepočítaje tyto okrajové případy, klasická herní umělá inteligence se zásadním způsobem liší od té akademické. Cílem herní UI většinou není co nejrychleji a nejkvalitněji vyřešit problém, ale maximalizovat hráčův požitek ze hry. To opět může podle případu znamenat velmi odlišné věci. Mnohé z těchto případů jsou implementací podobné akademické UI, ale některé z nich jsou spíše iluze a klamy, které mají pouze navodit zdání inteligentního chování. Více o jednotlivých typech herní UI lze nalézt v kapitole 3.

## 2.3 Prohledávání stavového prostoru

*Prohledávání stavového prostoru je skupinou metod řešení úloh, spadající do oblasti umělé inteligence. Jejich princip spočívá ve vhodném procházení stavů řešené domény za účelem nalezení požadovaného stavu.*[13]

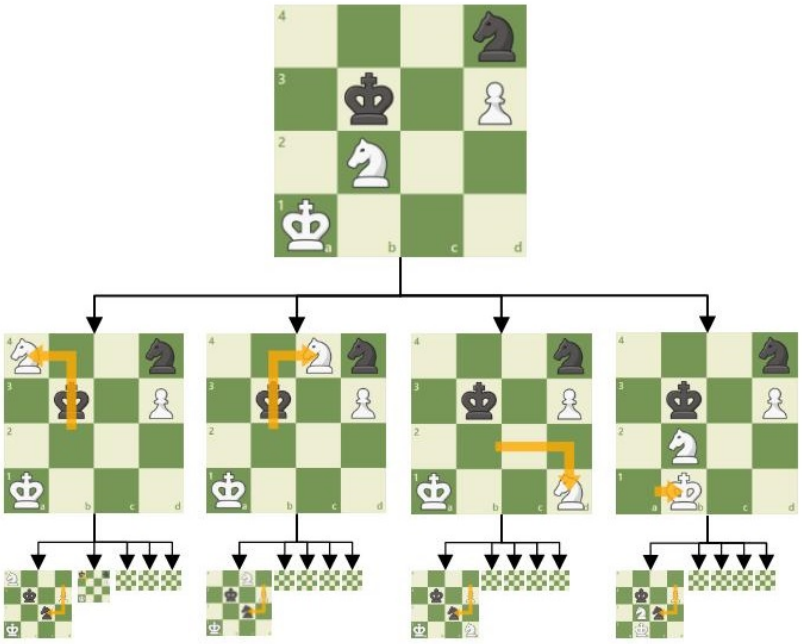
V kontextu herní UI jsou tyto metody důležité zejména pro dva účely. Prvním jsou již zmíněné algoritmy hrající deskové hry. Stavy řešené domény zde představují možné stavy hry čili různá rozmístění herních kamenů. Požadovaný stav pak většinou znamená výherní stav pro hrajícího bota. Simulací možných tahů ze současného herního stavu a ze všech dalších zkoumaných stavů vzniká větvící se stromová struktura.

Prohledávající algoritmy zkoumají jednotlivé stavy, hodnotí je (např. pomocí heuristické funkce) a nakonec vybírají nejvhodnější bezprostřední tah, který pak simulovaný hráč provede. Pokud je stavový prostor malý (málo se větví), lze prohledávat takzvaně hrubou silou (anglicky brute force).

Prohledávat hrubou silou znamená, že algoritmus zkoumá do určité hloubky stromu všechny stavy, které mohou ve zbytku probíhající hry nastat. Většinou však je stavový prostor příliš velký a je potřeba využít pokročilejší prohledávací algoritmy.

### 2.3.1 Hledání cest

Druhým hlavním využitím prohledávání stavového prostoru v kontextu herní UI je hledání cest (anglicky pathfinding). Algoritmus v tomto případě hledá optimální cestu z bodu A do bodu B. Prohledávání stavového prostoru probíhá podobně jako u hraní tahových her, ale místo zkoumání platných herních tahů, algoritmus provádí elementární tahy se zkoumanou postavou či jednotkou ve videohře. Stěžejní algoritmus pro tento účel je A* (A star, podkapitola 4.1).



Obrázek 2.1: Příklad stromové struktury stavového prostoru ve hře šachů

## 2.4 Stromy chování

Stromy chování (anglicky behavior tree, BT) je způsob jak strukturovat přepínaní mezi různými úkoly autonomního agenta jako je například robot nebo virtuální entita v počítačové hře. Stromy chování jsou efektivní způsob vytváření komplexních systémů, které jsou zároveň modulární a reaktivní. Tyto vlastnosti jsou kritické v mnoha použitích, což vedlo k rozšíření stromů chování z oblasti vývoje počítačových her do mnoha odvětví umělé inteligence a robotiky. Stromy chování slouží jako náhrada za dříve používané konečné automaty.[3]

Na rozdíl od rozhodovacích stromů, které se prochází vždy od kořenového uzlu až po listový uzel, aktivita ve stromech chování se může přesouvat všemi směry a navíc nezaniká po průchodu stromem. Stromy chování mohou mít podle jejich konkrétní implementace různé výběry druhů uzlů, ale několik klíčových druhů lze očekávat v každé variantě. Jedná se o uzly typu Sequence, Selector a Task.

- *Sequence – provádí sekvenci uzlů zleva doprava. Pokud všechny uzly skončí úspěchem, skončí Sequence uzel úspěchem. Jakmile jakýkoliv uzel neuspěje, celá sekvence neuspěje.*[14]
- *Selector – provádí sekvenci uzlů zleva doprava. Při prvním úspěchu končí selector úspěchem. Pokud neuspějí všechny uzly, končí selector neúspěchem. Na rozdíl od Sequence uzlu, selector pokračuje po neúspěchu jednoho uzlu dalším uzlem v řadě.*[14]
- Task – představuje konkrétní aktivitu, kterou bude autonomní agent provádět. Každá aktivita musí být definované vlastní podmínky úspěchu a neúspěchu.[14]

# Kapitola 3

# Typy herní umělé inteligence

Herní umělou inteligenci můžeme rozdělit na několik typů. Použitý typ částečně souvisí s žánrem hry.[14] Každá hra může obsahovat vícero druhů herní UI, ale stejně tak nemusí obsahovat žádnou.

## 3.1 Autonomní chování herních entit

Postavy, zvířata, příšery a jiné herní entity, ať už přátelské, nepřátelské či neutrální, většinou potřebují být ovládány počítačem, aby se jevily jako živé a inteligentní. Hráč se pak snáze ponoří do herního světa, když se entity chovají v rámci jeho očekávání, což pomáhá plnit primární funkci her. Naopak chyby v chování entity nebo její neschopnost vyrovnat se s nečekanou situací mohou způsobit ztrátu hráčova zájmu, což může mít negativní dopad na úspěch hry, děje-li se tak hromadně.

Umělá inteligence neutrálních entit je z implementačního hlediska nejjednodušší. Neutrální entity jsou většinou zvířata, od kterých není očekáváno mnoho interakcí s hráčem. Příklady autonomního chování neutrálních entit jsou:

- Zvíře se nahodile pohybuje po okolí.
- Domestikované zvíře utíká před hráčem, když na něj hráč zaútočí.
- Divoké zvíře (srnka) utíká před hráčem, když ho spatří.
- Zvíře následuje hráče, když hráč drží krmivo v ruce.

Umělá inteligence nepřátelských entit má velký rozptyl v komplexitě mezi hrami. Nepřátelské entity představují překonatelnou překážku pro hráče a u mnoha her je interakce s nepřátelskými entitami hlavní náplní hry. Správná funkcionalita těchto entit je tedy kritická pro mnoho her.

Velká část herních mechanik vyžaduje, aby s nimi uměly správně interagovat i nepřátelské entity. Dá se tedy říct, že čím více herních mechanik hra má, tím složitější je implementace nepřátelských entit. Příklady autonomního chování nepřátelských entit jsou:

- Nepřítel útočí na hráče.
- Nepřítel se přesouvá do pozice, ze které může zaútočit na hráče.
- Nepřítel se brání před útoky hráče.

- Nepřítel se adaptuje na akce hráče.
- Nepřítel hlídkuje.
- Nepřítel hledá schovaného hráče, který na sebe upozornil.

Umělá inteligence přátelských entit bývá přinejmenším o něco složitější než UI nepřátelských entit. Krom toho, že přátelské entity zpravidla potřebují umět všechno, co umí nepřátelské entity, tak od nich může být vyžadováno umět například:

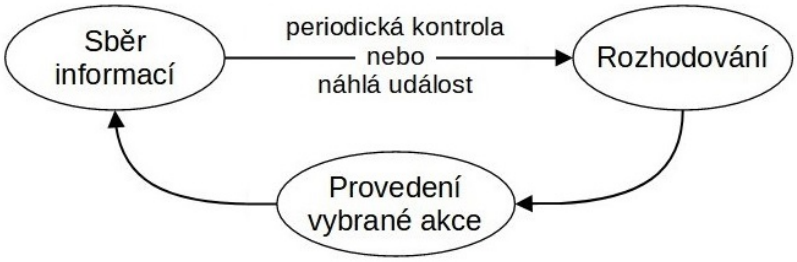
- Následovat hráče a vyhýbat se u toho překážkám.
- Nepřekážet hráči.
- Vykonávat hráčovy příkazy.

Za speciální případ autonomního chování entit by se dalo označit chování jednotek ve hrách žánru RTS (strategické hry probíhající v reálném čase) a MOBA (herní žánr, který vznikl z žánru RTS). Od jednotek v těchto hrách se očekává schopnost se přinejmenším autonomně bránit, pokud zrovna neprovádí jiné příkazy vydané živým či simulovaným hráčem.


Obrázek 3.1: Panel příkazů ve hře Age of Empires II: Definitive Edition

Například ve hře Age of Empires II: Definitive Edition (obrázek 3.1) má hráč k dispozici celou řadu příkazů ovlivňujících autonomní chování jednotek:

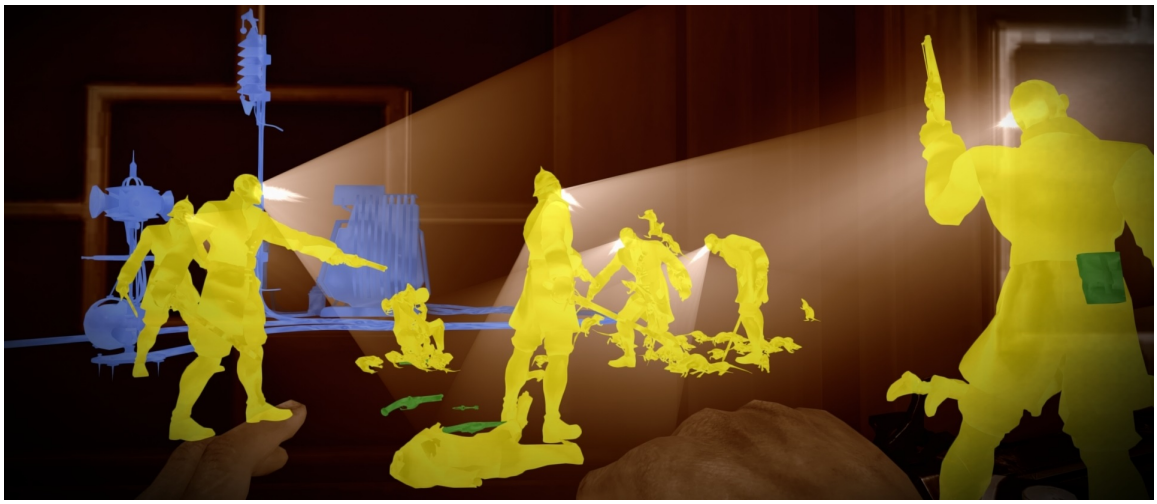
- Hlídkování mezi body.
- Střežení jednotky či budovy.
- Sledování nepřítele z bezpečné vzdálenosti.
- Pohyb s útokem na případné nepřátele po cestě.
- Čtyři módy autonomní sebeobrany (agresivní, defenzivní, drž pozici, pasivní).
- Automatické prohledávání mapy.
- Čtyři bitevní formace armád.




Obrázek 3.2: Diagram herní UI entit [14]

Minimální model autonomního chování entit je koloběh tří fází (obrázek 3.2). Entita nejdříve sbírá informace, na základě získaných informací se rozhodne provést nějakou akci, tuto akci provede a nakonec se vrátí ke sběru informací.

Sběr informací může mít mnoho podob. Většinou se jedná o napodobení nebo přímo simulaci lidských smyslů, zejména zraku (obrázek 3.3) a někdy také sluchu. Ostatní lidské smysly se ve hrách objevují jen zřídka. Chuť a hmat mají smysluplné využití pro hráče, ale vyjma vibrací u konzolových ovladačů zatím nejsou rozšířené periferní zařízení, které by tyto smysly pro hráče zprostředkovaly. Pro herní entity by navíc neměly využití téměř žádné.



Obrázek 3.3: Vizualizace zorných polí nepřátel ve hře Dishonored 2

Čich využití sice má (např. zvíře ucítí hráčovu postavu), ale čichové herní mechaniky se ve hrách moc neobjevují. Důvodem je pravděpodobně opět fakt, že nejsou periferie, které by takovou mechaniku zprostředkovaly hráči. Čich se prozatím simuluje formou vizualizovaných oblaků (obrázek 3.4), ale čichové herní mechaniky zpravidla interagují pouze s hráčem a ne s herními entitami.

Po sběru informací přechází systém UI do fáze rozhodování. Děje se tak buď periodicky, nebo na základě přednastavených událostí jako třeba hráč vstupující do zorného pole entity. Systém rozhodování bývá v dnešní době implementován pomocí stromů chování (podkapitola 2.4). Tyto stromy na základě sesbíraných dat z první fáze a také na základě vnitřních stavů dané entity rozhodují, kterou akci nebo sekvenci akcí má entita provádět. Během


Obrázek 3.4: Vizualizace pachů ve hře The Witcher 3: Wild Hunt

provádění akcí může entita znova sbírat nové informace a rozhodovat o dalším vývoji jejího chování.

## 3.2 Skriptované chování

Za herní UI se dá označit i předem naskriptované chování herních entit.[14] Naskriptované chování je sekvence akcí, která je buď pevně daná, nebo obsahuje minimální množství variací. Používá se většinou pro příběhové části hry, ve kterých je interakce s hráčem omezená nebo irelevantní. Technicky vzato vůbec nejde o umělou inteligenci, protože nedochází ke sběru informací a entita nereaguje na své okolí nebo reaguje jen velmi omezeně.

## 3.3 Simulovaný hráč

Hry pro více hráčů, kde hráči hrají proti sobě, zpravidla nabízí možnost nahradit hráče simulovaným hráčem, kterého ovládá počítač. Typickými žánry pro tento typ herní UI jsou hry strategické (Age of Empires, Heroes of Might & Magic, ...), bojové (Tekken, Mortal Kombat, ...) a kompetitivní online hry (Counter-Strike, League of Legends, ...).

Detaily fungování této herní UI se liší mezi jednotlivými žánry. Obecně se dá říct, že čím více ve hře záleží na rychlé akci a koordinaci rukou a očí (anglicky hand-eye coordination), tím jednodušší je vytvořit silnou simulaci hráče. Důvodem je to, že počítač koordinovat své oči a ruce nemusí. V tomto ohledu je počítač dokonalý nadčlověk.

Na základě tohoto parametru se dají hry rozdělit do tří kategorií – akční, kombinované a čistě strategické. Hra šachy a jim podobné hry jsou speciálním případem čistě strategických her.

Akční hry, které převážně závisí na koordinaci očí a rukou, nepředstavují příliš velkou výzvu při implementaci umělé inteligence díky již zmíněné výhodě počítačů nad živými hráči. Tato výhoda musí být uměle omezována, aby simulovaní hráči nebyli neporazitelní a hra proti nim více připomínala hru proti živým hráčům. Z implementačního hlediska lze tuto UI vnímat jen jako trochu složitější verzi autonomního chování entit.

Kombinované hry jsou hlavně hry žánru RTS. Úspěch v těchto hrách závisí na relativně ekvivalentní kombinaci rychlé akce (běžně nazývané micro) a strategických rozhodnutích vyžadujících inteligenci (nazývané macro). Herní UI simulovaných hráčů v těchto hrách je

daleko složitější než v případě akčních her, ale i přesto bývá v dnešní době docela kvalitní a představuje přijatelnou obtížnost proti živým hráčům. Velkou roli zde opět hraje fakt, že počítačový hráč má nadlidsky dokonalé micro. Zaprvé má vždy všechny dostupné informace o svých jednotkách, což hráč musí zjišťovat postupným přesouváním kamery po mapě. Zadruhé může provádět relativně neomezené množství akcí za minutu, v čemž bývají i nejlepší hráči omezeni řádově na několik stovek akcí za minutu.

Nedávnou výjimkou je umělá inteligence AlphaStar od DeepMind pro hru Starcraft 2, která byla v průběhu vývoje modifikována tak, aby byla při hře omezená podobně jako živí hráči. Informace musela získávat stejným způsobem jako hráči a stejně tak i její akce za minutu byly omezené na úroveň podobnou nejlepším živým hráčům.[1][1]

Standardní implementace zahrnuje většinou nějaký flexibilní naskriptovaný herní plán, který určuje co a za jakých podmínek má UI provádět a jak má reagovat na různé události způsobené například ostatními hráči.

Některé hry obsahují velmi malé množství akčních prvků a místo toho závisí pouze na strategii hráče (např. hry Civilization, Stellaris, Heroes of Might & Magic). Kvůli vysokým nárokům na skutečně inteligentní chování a často i dlouhodobé plánování je velmi obtížné vytvořit pro tyto hry kvalitní herní UI. Proto bývá u těchto her zvykem, že obtížnost UI je uměle zvýšená různými výhodami jako třeba bonusem do produkce zdrojů. Jinak se UI pro tyto hry příliš neliší od UI pro hry kombinované, ale důsledkem často větší komplexity těchto her musí být naskriptované herní plány daleko rozsáhlejší.

Speciálním případem čistě strategických her jsou tahové hry s omezeným herním prostorem (např. šachy, Go). Tyto hry mají daleko jednodušší pravidla, menší volnost a většinou relativně malou herní plochu. Díky tomu a díky tomu, že jsou hry tahové, lze současný herní stav a jeho možné budoucí vývoje snadno převést do stromové struktury (podkapitola 2.3) a celou UI řešit například strategií Minimax nebo jejími variacemi, případně v kombinaci se metodami strojového učení. Díky tomu UI v těchto hrách už dlouhodobě dominuje nad lidskými hráči, přestože nemá žádnou neférovou výhodu jako v případě akčních her.

V Šachách tomu tak je od roku 1997 (Deep Blue vs. Garry Kasparov) a ve hře Go od roku 2016 (AlphaGo vs. Lee Sedol).[1] Velký časový rozdíl mezi těmito událostmi je dán především rozdílem v míře větvení stavového prostoru her. Šachy průměrně umožňují desítky různých tahů každému hráči, zatímco hra Go nabízí na začátku hry až stovky možných tahů. Překonání tohoto rozdílu jednoho řádu vyžadovalo 16 let vývoje umělé inteligence a exponenciálního růstu výkonu výpočetní techniky.

## 3.4 Řízení hry

Některé hry obsahují umělou inteligenci, která dynamicky upravuje obtížnost hry. Tento systém sleduje, jak se hráči momentálně daří, a podle toho zvyšuje nebo snižuje obtížnost tak, aby hráči vyhovovala. Klasickým použitím této UI jsou závodní hry. Soupeři jsou zrychlováni a zpomalováni tak, aby se stále drželi v okolí hráče. Dalším často zmiňovaným případem je takzvaný AI Director ze hry Left 4 Dead, který dynamicky upravoval vytváření nových zdrojů a nepřátel.[14]

[1] Celý příběh UI AlphaStar popsaný přímo autory této UI lze najít na adrese https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii

# Kapitola 4

# Obecně důležité metody

V této kapitole budou popsány vybrané algoritmy a metody, které jsou důležité pro téma herní umělé inteligence a pro vytváření hry.

## 4.1 Algoritmus A*

V problematice hledání cest je algoritmus A* dlouhodobě nejpoužívanější.[4] Důvodem je především jeho konzistentní rychlost a spolehlivost.

A* je možné použít k řešení mnoha problémů, hledání cest je pouze jedním z nich. V případě hledání cest je podstatou algoritmu opakované zkoumání nejslibnějšího bodu z těch, se kterými se algoritmus setkal. Klíčový je zde způsob určování toho, který ze zbývajících bodů má být právě ten nejslibnější.[4]

Každému potkanému bodu je přiřazena hodnota $f(n) = g(n) + h(n)$, kde $n$ značí zkoumaný bod, $g(n)$ značí přesnou cenu cesty ze startovního bodu do bodu $n$ a $h(n)$ je heuristická funkce, která určuje odhad ceny cesty z bodu $n$ do cílového bodu.

Pro heuristickou funkci existuje jedno omezení, které musí splňovat, aby nalezená cesta byla optimální. Odhad vytvořený funkcí $h(n)$ nikdy nesmí být větší než skutečná nejkratší cesta z bodu $n$ do bodu cílového.

Krom správné heuristické funkce potřebuje algoritmus dva seznamy uzlů - otevřený a uzavřený. Dále datová struktura uzlů musí pro každý uzel umožňovat ukládat reálné číslo (výsledek funkce $f(n)$) a odkaz na rodičovský uzel, jenž reprezentuje předchozí krok v cestě na daný uzel.

Samotný algoritmus pro v obecném případě funguje následovně:

1. Přidej startovní uzel do otevřeného seznamu.
2. Opakuj následující kroky:
   (a) V otevřeném seznamu najdi uzel s nejnižší hodnotou $f(n)$. Tento uzel označ za současný uzel.
   (b) Odstraň současný uzel z otevřeného seznamu a přidej ho do uzavřeného seznamu.
   (c) Pro každý sousední uzel k uzlu současnému:
      i. Pokud je sousední uzel v uzavřeném seznamu, ignoruj ho a pokračuj dalším uzlem.
      ii. Pokud sousední uzel není v otevřeném seznamu, přidej ho tam a učiň současný uzel jeho rodičovským uzlem. Vypočítej pro něj hodnoty $g(n)$, $h(n)$ a $f(n)$.

iii. Pokud už sousední uzel v otevřeném seznamu byl, zkontroluj, jestli cesta ze současného uzlu není levnější než z rodičovského uzlu. Pokud ano, vypočítej nové hodnoty *g(n)* a *f(n)*.

(d) Zastav, když je:

i. Cílový uzel je přidán do uzavřeného seznamu. Cesta nalezena.

ii. Otevřený seznam je prázdný. Cesta neexistuje.

3. Výsledná cesta je získána vracením se po rodičovských uzlech z uzlu cílového.

## 4.2 Strategie Minimax

Minimax je důležitá strategie pro takzvané hry s nulovým součtem (anglicky zero-sum games). To jsou hry více hráčů (nejčastěji dvou), ve kterých zisk jednoho hráče je ekvivalentní ztrátě ostatních hráčů, takže spolupráce v těchto hrách nemá smysl.[10]

Ve zkratce je cílem strategie Minimax minimalizovat ztráty pro nejhorší možný scénář. Jinými slovy - snažit se ztratit co nejméně za předpokladu, že soupeř či soupeři hrají optimálním způsobem. Minimax lze použít jak na hry v reálném čase, tak i na hry tahové. V případě tahových her je ale princip strategie jednodušší, a proto bude vysvětlena její varianta pro tahové hry.

Základem Minimaxu je heuristická funkce, která je schopná jakémukoliv stavu hry přiřadit číslo, vyjadřující který z hráčů má v tomto stavu hry výhodu. Heuristická funkce může k tomuto účelu využívat libovolné herní informace (dostupné hrací kameny, rozestavení kamenů, ohrožení kamenů, ...) a může být jakkoliv komplexní.

Při použití Minimaxu v jeho základní podobě se nejdříve vytvoří stromová struktura ze současného herního stavu způsobem popsaným v podkapitole 2.3. Struktura se vytváří pouze do předem stanovené hloubky, protože jinak by se mohla větvit donekonečna. Ohodnocení stavů se pak provádí pouze pro listové uzly stromu. Hodnota uzlů vyšších úrovní je vypočítána jako minimální či maximální hodnota ze všech hodnot potomků daného uzlu. Zda se použije minimální nebo maximálně hodnota je určeno tím, který hráč je ve zkoumaném herním stavu na tahu. Strategie Minimax předpokládá, že každý soupeř vždy provede nejlepší možný tah.[5]

Tato podoba Minimaxu ale pro většinu případů nestačí. Hloubka stromu je často nedostačující, protože strom se příliš rychle rozrůstá do šířky. Proto existují různé modifikované metody jako třeba Alfa-beta ořezávání nebo Monte Carlo tree search.

Alfa-beta ořezávání není příliš odlišné od původního Minimaxu. Pouze průběžně zahazuje tahy, o kterých je jasné, že už nemají šanci být lepší než některý z prozkoumaných tahů. Název pochází z hodnot alfa a beta, které se při průchodu stavovým prostorem ukládají a slouží k určování vyřazení či ponechání uzlů.[8]

Oproti tomu algoritmus Monte Carlo tree search je daleko komplikovanější. Náhodně prohledává stavový prostor a na zjištěných informacích provádí statistické výpočty. Na základě těchto výpočtů vybere tah, který má největší šanci na výhru.[2]

# Kapitola 5

# Princip hry

Praktickou částí této práce bylo vytvořit vlastní hru s protivníkem s umělou inteligencí. Vytvořená hra nese jméno Queens Regicide. Tato kapitola obsahuje vysvětlení principu a základních pravidel vytvořené hry. Některé uvedené informace platí pouze pro standardní mód hry. Hra obsahuje možnost hrát scénáře, které se mohou od standardní hry v mnoha ohledech lišit.

Primární inspirací pro hru byly šachy a lze zvýraznit několik podobností s touto předlohou. Queens Regicide je tahová strategie pro dva hráče a hraje se na čtvercové síti, na které hráči střídavě hýbají se svými hracími figurami. Také podobně jako v šachách je i zde primárním cílem polapit soupeřovu klíčovou figuru (v šachách král) a u toho co nejlépe chránit svoji vlastní klíčovou figuru. Nakonec zde platí i základní strategická doporučení pro hraní šachů jako „rozvíjej své figury“ a „ovládni střed hrací plochy“. Obě tato doporučení budou hrát roli při tvorbě herní UI.

Hra svou komplexitou plně využívá toho, že je v elektronické podobě a není možné ji převést do podoby deskové. Následuje stručné vysvětlení základních principů hry.

Místo klasického střídání po jednom tahu se ve hře Queens Regicide střídají hráči po třech tazích. Tyto tři tahy mohou být rozděleny mezi libovolný počet jednotek (např. tři tahy jednou jednotkou nebo po jednom tahu třemi jednotkami). Hráč může kolo ukončit i předčasně, pokud si nepřeje nebo nemůže využít svoje zbývající tahy.

## 5.1 Hrací figury

Hra obsahuje 10 druhů hracích figur (dále nazýváno jednotky). Klíčová jednotka každého hráče se jmenuje Queen (královna). Podobně jako šachový král je tato jednotka téměř zcela bezbranná a spoléhá na ochranu ostatních jednotek. Zbývajících 9 druhů má na rozdíl od šachů relativně ekvivalentní hodnotu a přínos pro hráče. Jediný rozdíl v hodnotě jednotek by měl nastat vlivem osobní preference či vlivem výběru herní strategie, podobně jako tomu je například u figur střelce a jezdce v případě šachů.

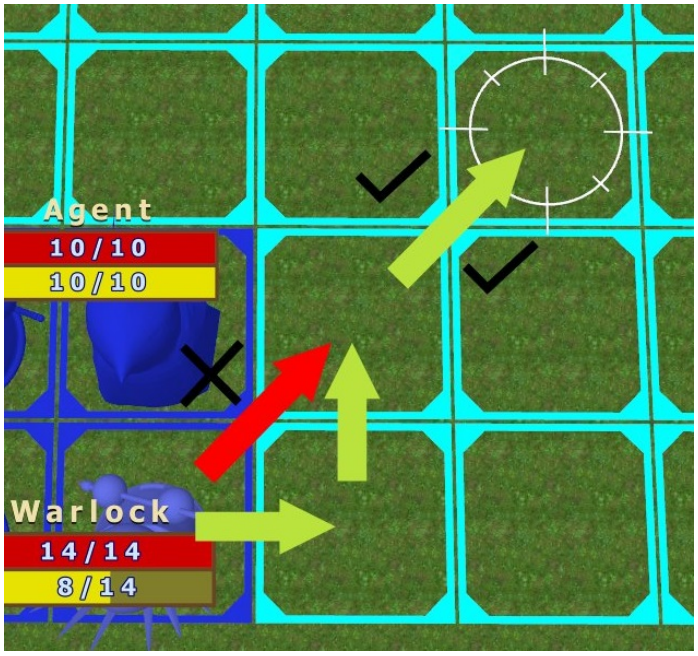
Každá jednotka má k dispozici několik zdrojů a schopností, kterými se odlišuje od ostatních. Primárními zdroji každé jednotky jsou životy a energie. Životy představují zdravotní stav jednotky. Určují maximální množství energie a v momentě, kdy životy jednotky klesnou na nulu, je tato jednotka nenávratně odstraněna ze hry. Cílem hry je tedy snížit životy soupeřovy královny na nulu. Životy jednotky se automaticky obnovují když jednotka neprovádí žádné akce nebo pomocí speciálních léčivých schopností.


Obrázek 5.1: Typy jednotek (figur) ve hře Queens Regicide

Energie slouží hlavně jako platidlo pro hýbání s jednotkou a k aktivování jejích speciálních schopností. Navíc slouží jako pasivní ochrana proti běžným útokům. Důsledkem toho je fakt, že používání jednotky ji činí více zranitelnou. Energie jednotky se automaticky obnovuje na začátku každého hráčova kola. Tato obnova je zvýšená, pokud jednotka neprovádí žádné akce.

Třetí zdroj, který mohou jednotky mít, je zbroj. Zbroj částečně chrání jednotku před útoky tak, že hodnota útoku je snížena o hodnotu zbroje. Je tedy obzvlášť účinná proti slabším útokům. Jednotky získávají zbroj buď ze své pasivní schopnosti nebo z aktivních schopností jiných jednotek.

Každá jednotka se umí pohybovat stejným způsobem. Může se v jednom tahu pohnout o libovolný počet polí, ale za každý krok mezi poli musí jednotka zaplatit energií. Kroky mohou být vždy prováděny mezi dvěma hracími poli sousedícími horizontálně či vertikálně, pokud jsou obě pole volná (nepočítaje pohybující se jednotku). Kroky mohou být prováděny i diagonálně pokud je volný celý $2 \times 2$ čtverec diagonálního pohybu (obrázek 5.2).

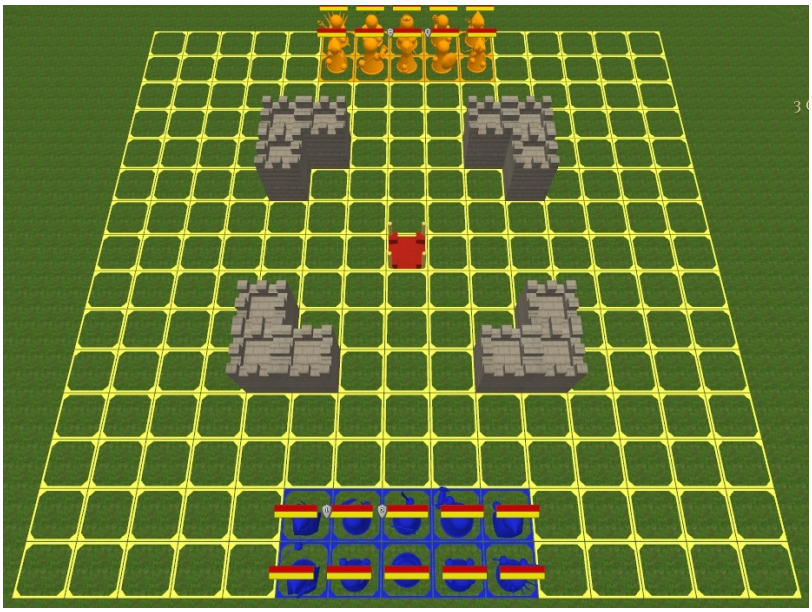



Obrázek 5.2: Povolené kroky mezi poli ve hře Queens Regicide

Kompletní seznam všech jednotek a jejich schopností lze nalézt v příloze B.

## 5.2 Hrací plocha

Místo klasické šachovnice $8 \times 8$ (64 polí celkem) se tato hra hraje na ploše $15 \times 15$ (225 polí celkem, obrázek 5.3). To v kombinaci s volnějším pohybem figur výrazně navyšuje množství možných tahů v běžné herní situaci, a tím zpomaluje jakékoliv prohledávání stavového prostoru možných rozvojů hry. Tento efekt je zcela záměrný.



Obrázek 5.3: Hrací plocha hry Queens Regicide

Původní návrh hry zahrnoval i herní mechaniku, kde by hráči na začátku hry libovolně rozmisťovali své figury po celé šířce hrací plochy. Z časových důvodů tato mechanika nebyla implementována a místo toho hráčových jednotky zahajují hru v předem daných formacích. Následek toho je, že rohy a boční okraje hrací plochy ztratily strategický význam. Herní UI je tedy docela uměle zpomalována tím, že musí vždy uvažovat i nad tahy do těchto nepoužívaných částí hrací plochy.

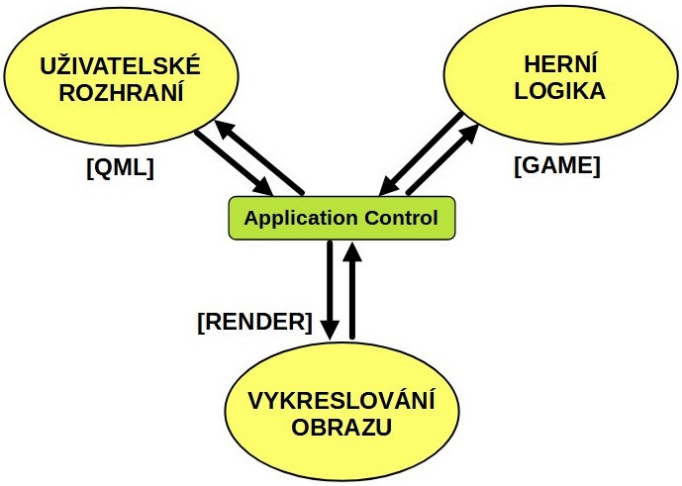
Jak už bylo zmíněno, v této hře je doporučeno bojovat o střed hrací plochy. Důvodem je políčko s trůnem, které poskytuje obrovskou výhodu tomu hráči, který má na trůnu svoji královnu.

# Kapitola 6

# Návrh hry

Tato kapitola jednak vysvětluje interní strukturu modulů a tříd ve zdrojovém kódu a jednak popisuje detailní návrhy uživatelského rozhraní, herní logiky a umělé inteligence. Součástí kapitoly jsou různé diagramy používající převážně anglické názvy tříd a atributů. Tyto anglické názvy nebo jejich blízké variace jsou používány ve zdrojovém kódu hry. Jejich použití (oproti českým překladům) má čtenáři umožnit lépe se orientovat v kódu. Zdrojový kód hry lze zpřístupnit například v repositáři na stránce GitHub.[1] Zmíněné diagramy jsou většinou zjednodušené oproti skutečnému obsahu kódu. Kompletní diagramy jednotlivých modulů hry by byly příliš složité a rozsáhlé.

Program je rozdělen do tří hlavních modulů – uživatelského rozhraní, herní logiky a vykreslování obrazu. Tyto moduly jsou od sebe navzájem izolované a komunikují spolu výhradně přes centrální řídící třídu ApplicationControl, jak je zobrazeno na obrázku 6.1. V repositáři se zdrojový kód uživatelského rozhraní nachází ve složce *queens-code/resources/qml/*. Kód ApplicationControl lze nalézt ve složce *queens-code/src/*. Kódy herní logiky a vykreslování obrazu se pak nachází na stejném místě ve složkách *game* a *render*.




Obrázek 6.1: Diagram hlavního rozdělení kódu

Tato zprostředkovaná komunikace zahrnuje především zpracování událostí z uživatelského rozhraní, aktualizace uživatelského rozhraní a generování signálů k překreslování gra-

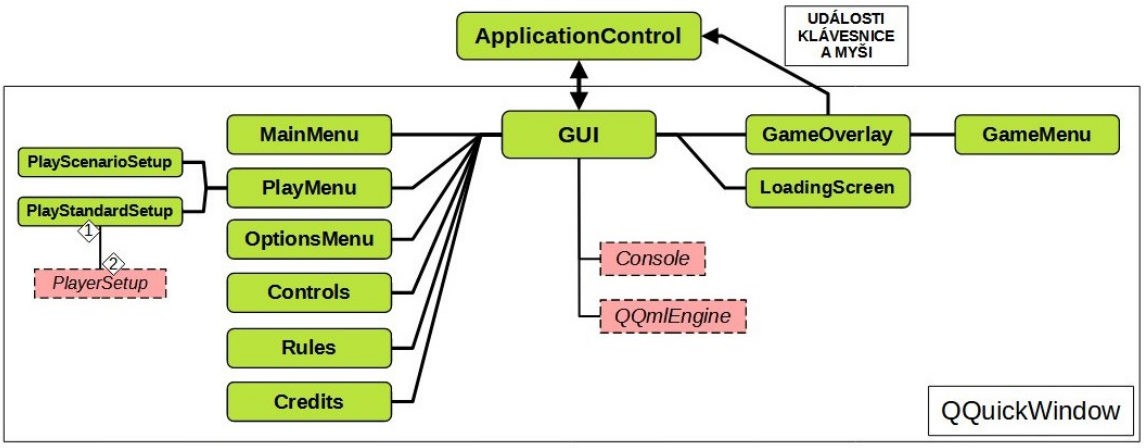
[1] https://github.com/xhrana02/queens-code

fické scény. Některé informace jsou po prvotním navázání spojení přes ApplicationControl posílány mezi moduly přímo. Například jednotky si drží ukazatel na svůj informační panel životů a energie a aktualizují jeho hodnoty bez použití ApplicationControl.

## 6.1 Uživatelské rozhraní

Uživatelské rozhraní je velmi důležitým prvkem pro každou strategickou hru. Uživatelské rozhraní zahrnuje jednak různé interaktivní a informační prvky zobrazované na obrazovce a jednak ovládání samotné hry interagováním s vykreslovanou scénou.

Uživatelské rozhraní je rozděleno do mnoha samostatných tříd (obrázek 6.2). Třída GUI (grafické uživatelské rozhraní) je kořenová třída okna aplikace a představuje hlavní komunikační spoj mezi uživatelským rozhraním a zbytkem aplikace. Každý další prvek představuje jednu z obrazovek, které může hráč ve hře vidět.



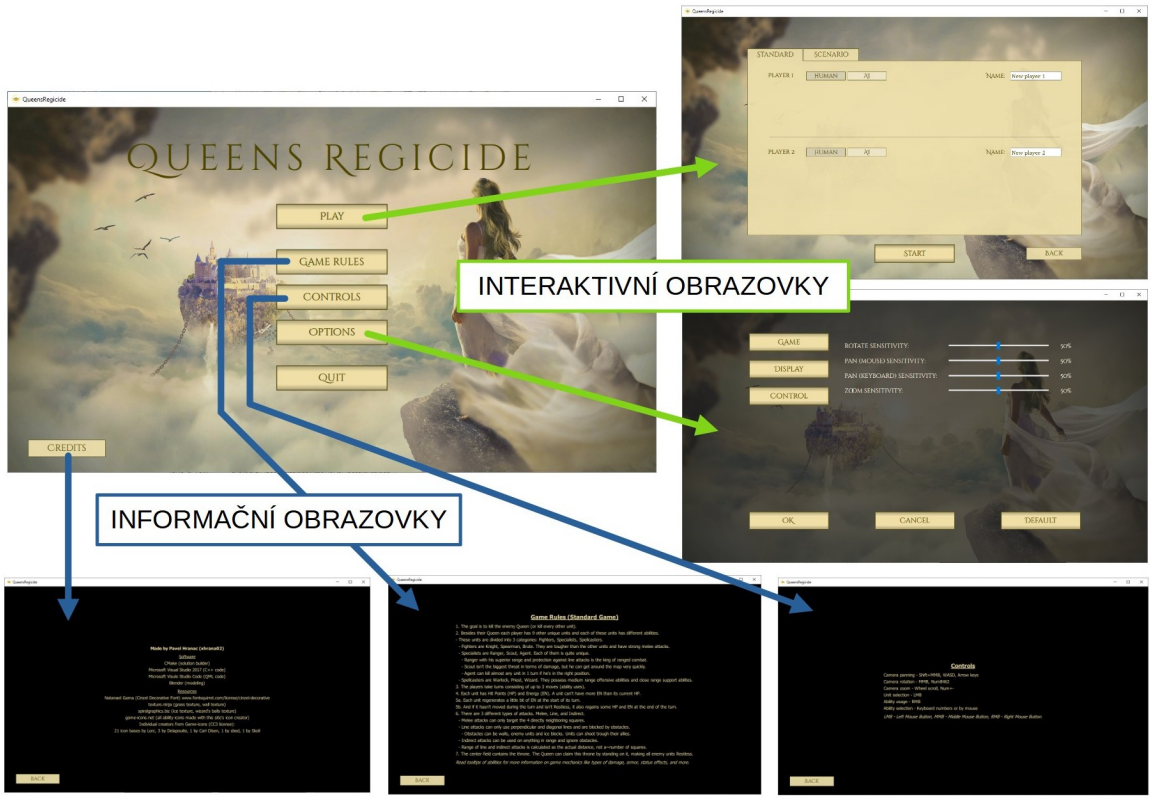

Obrázek 6.2: Diagram tříd uživatelského rozhraní

Třída MainMenu (hlavní menu, obrázek 6.3) je první obrazovka, kterou hráč uvidí po spuštění hry. Obsahuje několik tlačítek pro přepnutí na další obrazovky a také tlačítko pro vypnutí hry.

Třídy Controls, Rules a Credits jsou čistě informační obrazovky zobrazující pouhý text s vysvětlením ovládání hry, pravidel hry a s citováním zdrojů pro hru.

Třídy PlayMenu a OptionsMenu jsou interaktivní obrazovky. Obrazovka OptionsMenu slouží k nastavení různých možností v rámci celé aplikace jako například přepínání mezi zobrazením v okně a zobrazením na celou obrazovku nebo nastavení citlivosti herní kamery. Obrazovka PlayMenu slouží k nastavení parametrů hry a k jejímu spuštění. Nastavení parametrů hry je zanořeno do tříd PlayStandardSetup a PlayScenarioSetup mezi kterými může hráč na obrazovce PlayMenu přepínat.

Po spuštění hry hráč krátce uvidí průběh načítání na obrazovce LoadingScreen. Během hry pak hráč před sebou vidí obrazovku GameOverlay (obrázek 6.4), která zahrnuje vše potřebné k samotnému hraní. Pozadí obrazovky GameOverlay je průhledné/prázdné, díky čemuž lze vidět scénu vykreslovanou přes QQmlEngine. Stisknutím klávesy ESC může hráč vyvolat obrazovku GameMenu, která umožňuje měnit nastavení během hry nebo návrat do hlavního menu, odkud se může spustit nová hra.

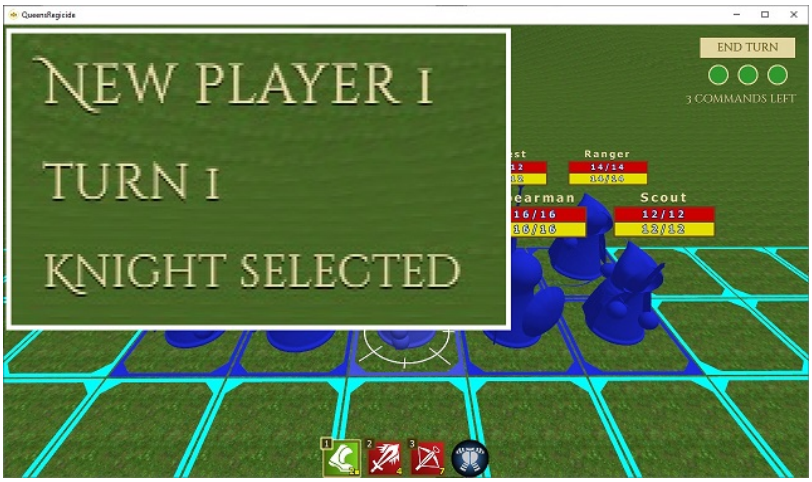



Obrázek 6.3: Hlavní menu hry


Obrázek 6.4: Herní overlay

Během hry má hráč k dispozici několik informačních prvků. Záměrem při navrhování těchto prvků byl minimalistický přístup s maximální vypovídací hodnotou.

Levý horní roh obrazovky (obrázek 6.5) zobrazuje který hráč je právě na tahu, kolikáté herní kolo se hraje a nakonec jméno vybrané jednotky. V případě, že hra skončí, první řádek zobrazí jméno vítěze.




Obrázek 6.5: Levý horní roh obrazovky GameOverlay

Pravý horní roh obrazovky (obrázek 6.6) obsahuje tlačítko pro předčasné ukončení kola a grafický i textový ukazatel zbývajících tahů/příkazů v současném kole. Během tahů umělé inteligence je celý tento roh skryt.


Obrázek 6.6: Pravý horní roh obrazovky GameOverlay

Uprostřed spodního okraje obrazovky se nachází panel schopností vybrané jednotky (obrázek 6.7). Aktivní schopnosti mají tvar čtverce s hnědým okrajem. Aktivní schopnosti lze vybírat myší nebo klávesovou zkratkou. V levém horním rohu každé ikony aktivní schopnosti se nachází číslo klávesové zkratky na alfanumerické klávesnici pro výběr schopnosti. Vybraná schopnost je zvýrazněna světle žlutým okrajem. V pravém dolním rohu ikon aktiv-

ních schopností se nachází cena za použití schopnosti. Žlutá čísla představují cenu v bodech energie a červená čísla představují cenu v životech. Čtverec vedle čísla u pohybových schopností značí, že cena se platí za každý provedený krok. Pasivní schopnosti mají kulatý tvar s černým okrajem. Pasivní schopnosti nelze nijak vybrat a jejich ikony neobsahují klávesové zkratky ani cenu za aktivaci schopnosti. Při najetí myší na jakoukoliv schopnost (aktivní i pasivní) se zobrazí detailní popis schopnosti.


Obrázek 6.7: Panel schopností vybrané jednotky

Každá jednotka má nad sebou informační panel (obrázek 6.8). Tento panel simuluje umístění v prostoru, ale jedná se o prvek uživatelského rozhraní. To lze ověřit pokusem o překrytí informačního panelu jiným modelem (např. hradbou). Panel se zobrazí před modelem hradby. Pokud se panel příliš vzdálí od kamery, některé detaily panelu se skryjí, protože by byly nečitelné. Pozice panelu a jeho hodnoty jsou aktualizovány přímo z individuálních objektů příslušných jednotek.


Obrázek 6.8: Informační panel jednotky

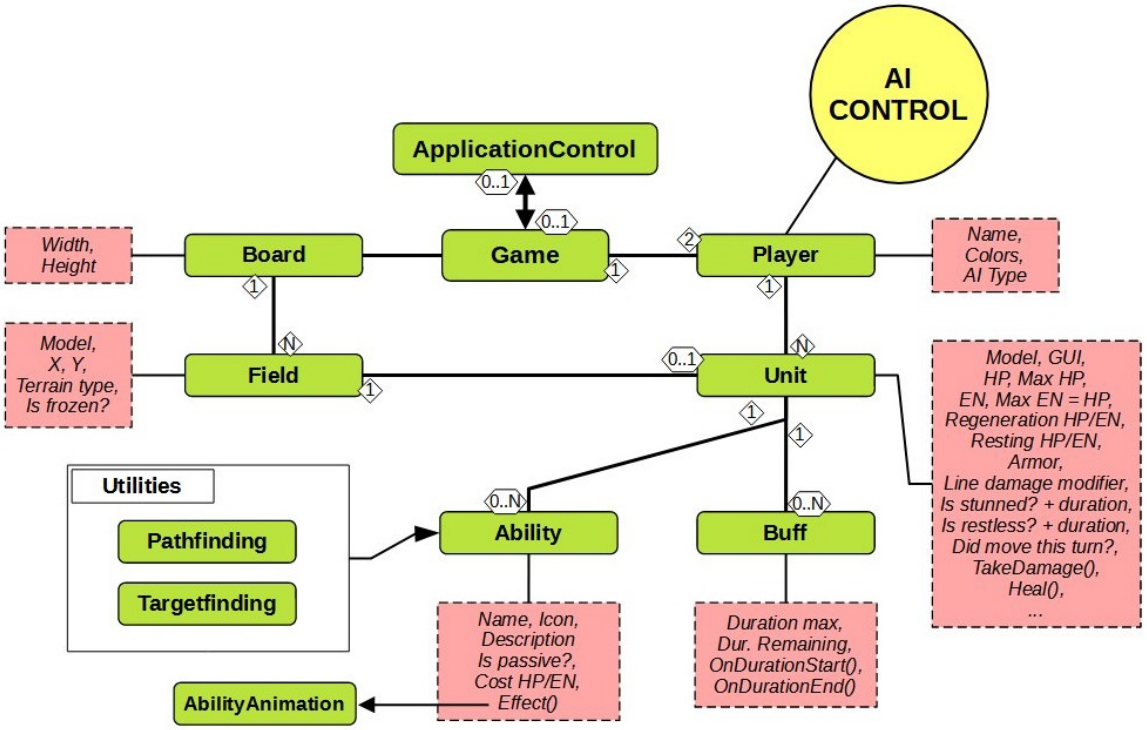
Centrální část panelu obsahuje ukazatele životů a energie jednotky. Pokud hráč najede myší na validní cíl pro vybranou schopnost, část hodnot životů a energie může začít pulzovat, což napovídá jaká by nastala změna, kdyby hráč vybranou akci provedl. Po provedení akce se okamžitě projeví změny v hodnotách životů a energie a z panelu vyletí vzhůru číslo odpovídající změně oproti předchozímu stavu.

Nad panelem se nachází jméno jednotky. V případě, že má jednotka nenulovou zbroj, vlevo od panelu se zobrazuje aktuální hodnota zbroje v ikoně štítu. Pokud je jednotka ovlivněna nějakým dočasným stavem, jsou všechny tyto stavy a jejich zbývající doby trvání vypsány pod informačním panelem.

Hra také obsahuje konzoli, kterou lze zobrazit stisknutím klávesy „‘“. Konzole ve svém současném stavu slouží pouze jako záznam událostí v aplikaci. Konzole přijímá i vstup od uživatele, ale žádné příkazy zatím nerozpoznává.

## 6.2 Návrh herní logiky

Modul herní logiky obsahuje především vše, co je potřeba k virtuálnímu běhu hry. Tento modul sám o sobě je schopen provádět simulované hry za pomocí umělé inteligence. Mohl by být lehce upraven tak, aby tyto hry spouštěl například přes příkazovou řádku, a vracel by výsledky her umělých inteligencí se zadanými parametry, což by mohlo být využito při trénování herní UI. Stejně tak by se dal snadno upravit tak, aby místo grafického výstupu zobrazoval průběh hry v textové podobě v konzolovém okně. Ve své stávající podobě ale podporuje pouze současný grafický výstup a použití.




Obrázek 6.9: Diagram tříd herní logiky

Diagram tříd herní logiky lze vidět na obrázku 6.9. Podobně jako u uživatelského rozhraní je i zde jedna třída, která zprostředkovává většinu komunikace s ApplicationControl.

Instance této třídy Game představuje jednu rozehranou hru. K ApplicationControl může náležet 0 až 1 rozehraných her (hráč může být například v menu). Stejně tak ke každé instanci hry může náležet 0 až 1 ApplicationControl. Hra nekomunikuje s ApplicationControl pokud je simulovaná – vytvořila ji umělá inteligence jako uzel stavového prostoru v rámci vymýšlení svého tahu.

Hra obsahuje jednu hrací desku a dva hráče. Uchovává si obecné informace jako současné kolo hry a odkaz na aktivního hráče, ale i mnoho prvků specifických pro vykreslování obrazu a uživatelské rozhraní (animovaný kurzor pod myší, modely prostředí, synchronizace pulzovacích efektů v GUI, blokování GUI při přehrávání animací, ...).

Třída Board představuje hrací desku. Je definovaná svou šířkou a výškou – je vždy obdélníkového tvaru. Dále obsahuje ukazetele na jednotlivá hrací pole, ukazetel na případné pole s trůnem a nakonec seznam zvýrazněných polí.

Třída Field představuje hrací pole. Pamatuje si, které hrací desce patří a jaké jsou jeho koordináty v rámci hrací desky. Dále si drží informace o tom, jestli je pole zmrzlé a hlavně jaká (jestli nějaká) jednotka se na poli nachází. Nakonec má uložené objekty s modely pro vykreslování obrazu – okraj pole, terén (pokud je, např. hradby) a ledový blok (vytvářený schopností Ice Block jednotky Wizard).

Třída Player představuje jednoho hráče náležícího do právě jedné hry. Uložené má svoje jméno, id, typ umělé inteligence (člověk/žádná, jednoduchá, střední, těžká), případný ukazatel na umělou inteligenci ovládající tohoto hráče, seznamy živých a mrtvých jednotek, které hráči patří, hráčovy barvy a počet zbývajících tahů v kole.

Třída Unit představuje jednotku. Tato třída obsahuje velké množství informací – model jednotky, životy, energie, zbroj, regenerační hodnoty, hodnoty odpočinku, modifikátory zranění, stavy, seznam schopností, seznam Buffů a další. Obsahuje také všechny možné funkce pro udělování poškození, léčení, aplikování Buffů, změny stavů jednotky, vybrání jednotky a další. Jednotky si také uchovávají ukazatele na svůj informační panel a panel schopností.

Třídy jednotlivých typů jednotek (Agent, Brute, Knight, Priest, Queen, Ranger, Scout, Spearman, Warlock, Wizard) dědí z třídy Unit. Odlišují se od sebe pouze obsahem konstruktoru, který v těchto 10 případech určuje jméno jednotky, maximální počet životů a seznam schopností. Změny ostatních hodnot vlivem pasivních schopností nastávají v konstruktorech samotných schopností, jak lze vidět na příkladu ve výpisu 6.1. Vytvoření nového typu jednotky za využití stávajících schopností je díky tomu otázka vteřin. IceBlock také dědí z třídy Unit a stává se tak zvláštním případem jednotky. Na rozdíl od normálních jednotek IceBlock přepisuje chování některých funkcí třídy Unit.

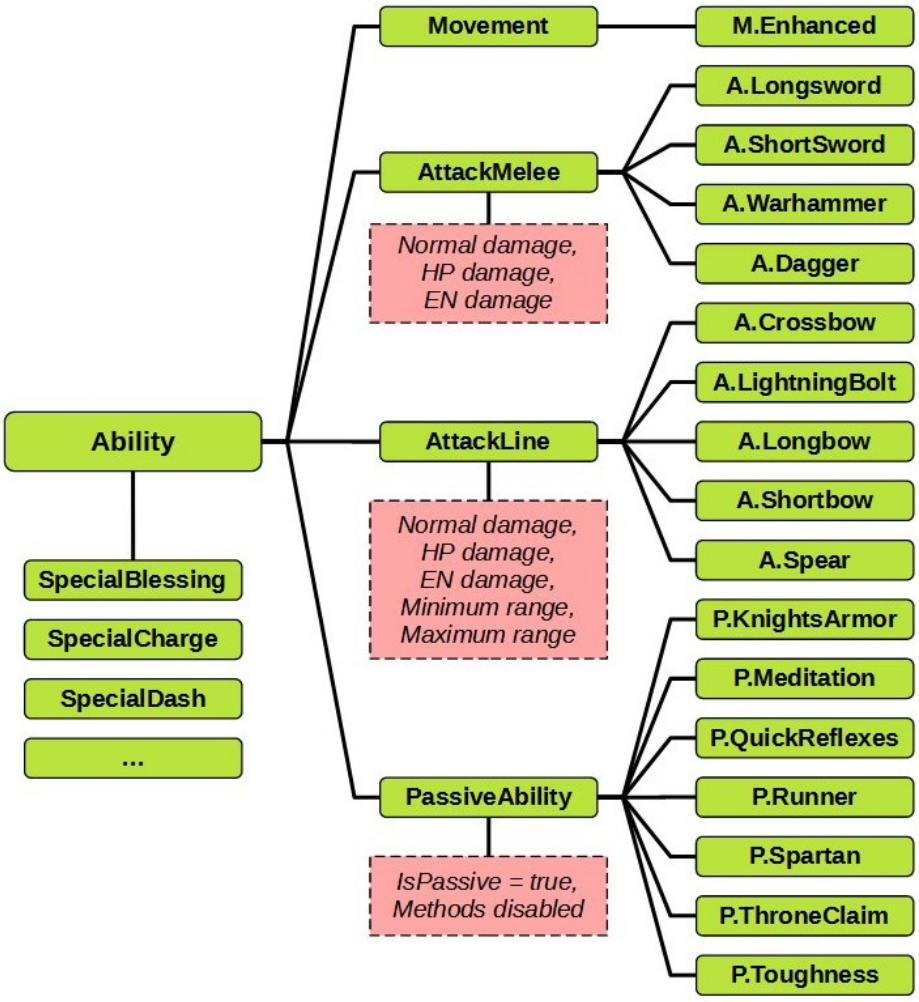
```
explicit PassiveSpartan(Unit* unit)
{
    name = "Spartan";
    iconPath = "icons/PassiveSpartan.png";
    description = "<b><u>Spartan</u> ( passive )</b><br><br>"
        "When an enemy unit moves out of a neigbouring tile, the spearman
           deals 3 HP damage to it.<br>"
        ACTIVEPASSIVE_TOOLTIP
        HP_DAMAGE_TOOLTIP
        "<br><br>The Spearman also has bonus 1 Armor.<br>"
        ARMOR_TOOLTIP;
    unit->HasOpportunityAttack = true;
    unit->DamageOpportunityAttack = 3;
    unit->IncreaseArmor(1);
}
```

Výpis 6.1: Příklad konstruktoru pasivní schopnosti

Model jednotky není stabilní součástí typu jednotky. Místo toho je jednotce přiřazen po jejím vytvoření v třídě GameFactory (návrhový vzor Factory). Jednotky tak mohou v rámci různých scénářů obdržet libovolné modely, což může být využito například při tvorbě příběhových postav v kampaních. Stejně tak všechny ostatní vlastnosti jednotky jsou veřejně přepisovatelné, takže v rámci tovární metody pro výrobu scénáře mohou být vytvořeny kompletně unikátní jednotky. Možným rozšířením hry by mohlo být načítání továrních metod scénářů ze souborů, díky čemuž by si mohli hráči vytvářet své vlastní scénáře a sdílet je s ostatními hráči.

Třída Ability představuje schopnost jednotky. Obecně si uchovává jméno schopnosti, cestu k ikoně, popis schopnosti, cenu v životech a energii a zdali je schopnost pasivní či aktivní. V případě aktivních schopností je pak velmi důležitá funkce Effect, která obsahuje efekt schopnosti, čili sekvenci příkazů, která se provede v případě, že se živý nebo simulovaný hráč pokusí schopnost použít. Schopnosti by měly také definovat chování metod OnSelected a SelectedAbilityOnFieldHovered, které graficky napomáhají hráči používat schopnost. Nakonec by mělo být definováno chování metody calculateViableTargets, kterou pak používá umělá inteligence. Implementace této metody zpravidla využívá pomocnou třídu Targetfinding.

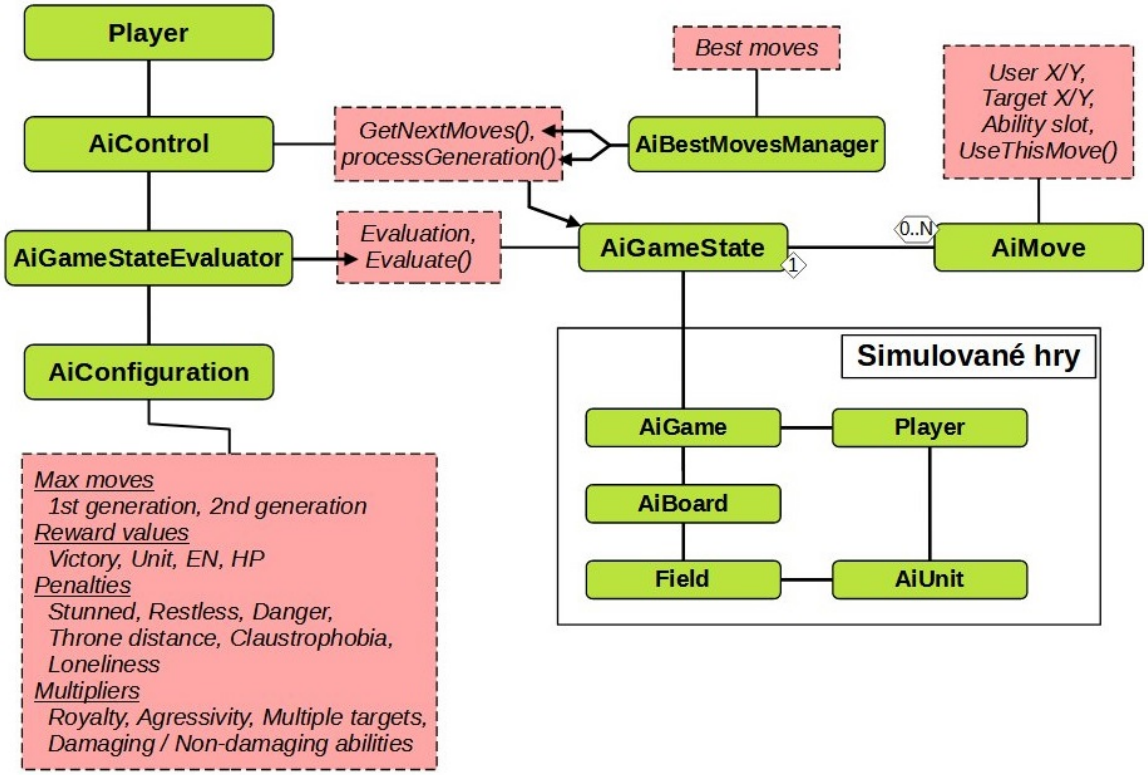
Schopnosti využívají víceúrovňové dědění pro agregaci podobných schopností, sdílejících implementaci funkcí a lišících se pouze v parametrech. Hieararchie dědění je ukázána na obrázku 6.10. Schopnosti s prefixem Special jsou unikátní a dědí proto přímo z třídy Ability.




Obrázek 6.10: Hierarchie dědění schopností

## 6.3 Návrh umělé inteligence

Část modulu herní logiky se zabývá umělou inteligencí simulovaných hráčů. Vnitřní strukturu této části lze vidět na obrázku 6.11. Třída AiControl jako jediná komunikuje se skutečnou hrou. Když se simulovaný hráč ovládaný umělou inteligencí dostane na řadu, požádá svou AiControl (svou umělou inteligenci) o vymyšlení tahů[2] pomocí metody GetNextMoves.




Obrázek 6.11: Diagram tříd umělé inteligence

AiControl si při svém vzniku vytváří instanci třídy AiGameStateEvaluator, která slouží k ohodnocování teoretických herních stavů pro účely použití strategie Minimax k nalezení vhodných tahů. Konkrétní implementace strategie Minimax a ohodnocování uzlů lze najít v podkapitole 7.5. Ohodnocování je ovlivněno různými parametry uloženými v instanci třídy AiConfiguration. Různé verze a varianty herní UI se od sebe liší právě hodnotami těchto parametrů. V současné verzi hry jsou tyto parametry předem dané podle zvolené obtížnosti UI. Stejně tak by se ale mohly parametry dynamicky měnit nebo by mohly být individuálně nastavitelné při spouštění nové hry.

Třídy AiGame, AiBoard a AiUnit dědí ze svých reálných verzí (Game, Board, Unit) a mírně upravují jejich chování, aby lépe fungovaly v prostředí simulované hry. Třídy Player, Field a další jsou použity ve své normální podobě.

Třída AiGameState slouží jako obal pro třídu AiGame. Rozšiřují ji o seznam tahů typu AiMove, které vedly ke vzniku daného herního stavu ze současného stavu skutečné hry (té, kterou vidí hráč na obrazovce). AiGameState dále umožňuje ohodnotit hru pomocí AiGameStateEvaluator a výsledek tohoto hodnocení si ukládá pro následné porovnání s ostatními herními stavy za pomocí třídy AiBestMovesManager.

[2]Pro připomenutí – kolo hráče se může skládat až ze tří tahů.

# Kapitola 7

# Implementace hry

Tato kapitola vysvětlí některé implementační detaily ze všech oblastí vytvořené aplikace.

Aplikace využívá celkem čtyři programovací jazyky. V jazyce CMake je napsán soubor CMakeLists, který umožňuje multiplatformní použití zdrojového kódu. V jazyce C++ je napsán modul herní logiky, modul vykreslování obrazu a centrální třída ApplicationControl. V jazyce QML je popsáno celé uživatelské rozhraní. Posledním jazykem je jazyk GLSL, ve kterém jsou napsány grafické shadery. Při programování byly použity vývojové prostředí Visual Studio (C++) a Visual Studio Code (QML).

Hra využívá knihovnu Qt verze 5.7. Samotná aplikace je instance třídy QApplication. Okno aplikace je instancí třídy QQuickWindow. Grafická scéna se vykresluje pomocí QQmlEngine. Qt modul QtQuick je zodpovědný za převedení QML kódu do funkčního uživatelského rozhraní.

## 7.1 Grafika

Grafická stránka hry je z technického hlediska primitivní. Hra využívá grafickou knihovnu GPUEngine,[1] která usnadňuje práci s OpenGL. Pro práci s vykreslovanými objekty byla vytvořena třída *RenderingObject*. Ta obsahuje instanci třídy *ge::sg::Scene*, jenž představuje soubor modelů a animací daného objektu. Modelové animace v práci použity nebyly, takže scéna slouží pouze jako úložiště modelů. RenderingObject dále obsahuje doplňující vlastnosti objektu - barvy, pozice, otočení, opakování textury a flagy pro zvýraznění objektu. Z pozice a otočení se před vykreslením scény vytváří translační a rotační matice, které po vynásobení vytváří modelovou matici. Barva a opakování textur jsou pak použity ve fragment shaderu. RenderingObject obsahuje také funkce (PrepareObject, updateGLScene, semantic2Attribute) převzaté ze vzorového příkladu Simple_QtgeSG knihovny GPUEngine. Tyto funkce umožňují samotné vykreslení objektu.

Barev v RenderingObject je celkem pět. Podle aktuálního nastavení flagů je vykreslovací funkci předávána jedna z nich. Nejvyšší prioritu má barva záblesku (flashingColor) při nastavení odpovídajícího flagu (flashing). Tato barva je definována animací, jenž si záblesk vyžádala. Například jednotka červeně zableskne, když ji zraní nějaká schopnost. Druhou nejvyšší prioritu má barva pulzující (fluctuatedColor). Používá se, pokud hráč na objekt ukazuje myší a její hodnota se mění podle sinusové křivky s průběhem času. Třetí barvou v pořadí je polozvýrazňovací barva (halflightColor), která se používá k označení validních cílů vybrané schopnosti. Čtvrtá barva je zvýrazňovací barva (highlightColor), která zvý-

[1] https://github.com/Rendering-FIT/GPUEngine

razňuje právě vybranou jednotku. Nejnižší prioritu má normální barva, která se použije v případě, že není nastavený žádný z flagů.

V aplikaci je pro zlepšení grafického výkonu a kvality použito několik jednoduchých metod. První metodou je takzvaný *back-face culling*, který zařizuje, aby se nevykreslovaly plochy otočené svou normálou pryč od kamery. Zapnutí této funkce je vidět ve výpisu 7.1.

```
1 glEnable(GL_CULL_FACE);
2 glCullFace(GL_BACK);
```

Výpis 7.1: Povolení OpenGL funkce back-face culling

Druhá metoda je *Multisample anti-aliasing*, který se zapíná opět jednoduše a to trojicí příkazů ukázaných ve výpisu 7.2.

```
1 auto format = QSurfaceFormat();
2 format.setSamples(4);
3 mainWindow->setFormat(format);
```

Výpis 7.2: Zapnutí multisample anti-aliasingu pro QQuickWindow

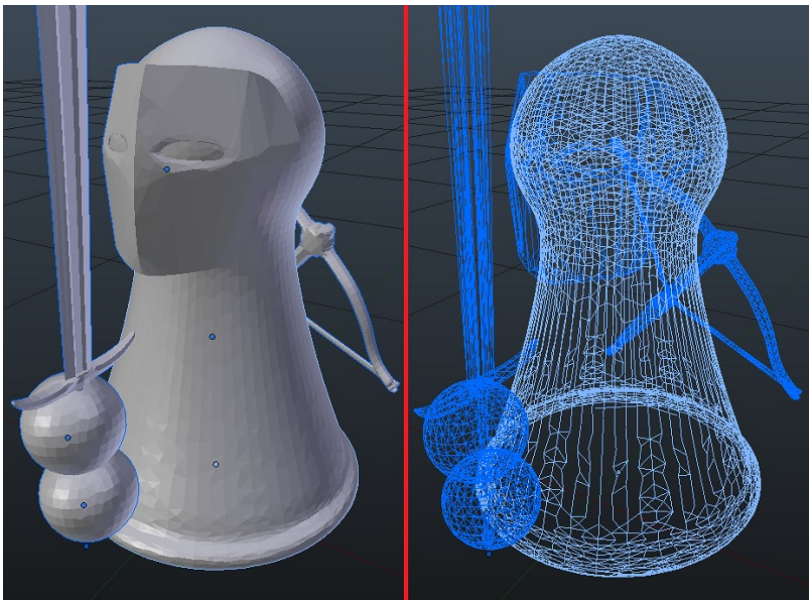
Třetí metodou je Phongův osvětlovací model implementovaný pomocí shaderů. Z technických důvodů byly implementovány pouze ambientní a difúzní složky. Odlesková složka chybí, což vytváří dojem matných materiálů a absence přímého světla.

## 7.2 Modely

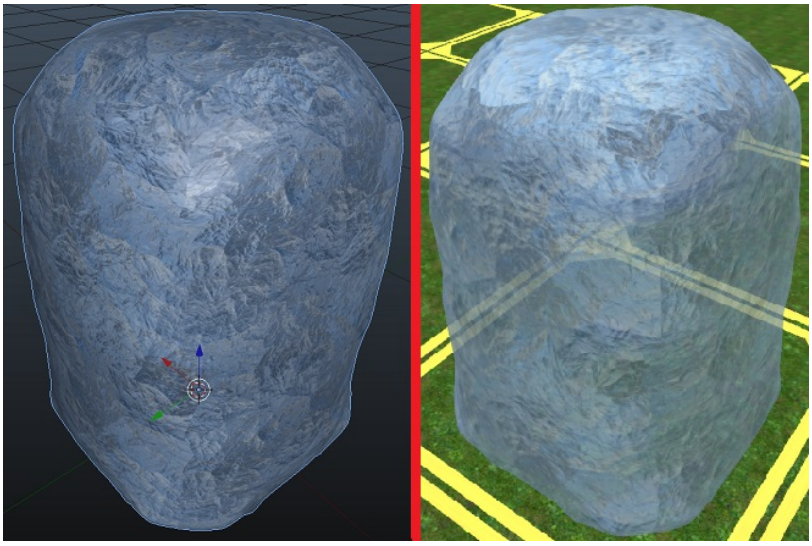
Všechny modely ve hře byly vytvořeny v programu Blender. Exportovány jsou jako soubory formátu *.obj*. Celkem hra obsahuje 16 nových modelů:

- Model kopcovitého terénu s plošinkou uprostřed pro hrací plochu. Terén používá upravenou texturu travnaté plochy. Originál textury byl získán z webu Texture Ninja.[2]
- Jednoduchý model hradby, který používá upravenou texturu zdi. Originál byl opět získán z webu *Texture Ninja*.
- Model červeno-zlatého trůnu.
- Jednoduchý model pro zobrazení hracích polí.
- Model „kurzoru“, který ve hře sleduje pozici myši promítnuté do roviny XZ.
- Deset detailních modelů jednotek (např. rytíř z obrázku 7.1). Jedna z jednotek používá texturu získanou z webu Spiral Graphics.[3]
- Model ledového kvádru (obrázek 7.2), který taktéž používá texturu z webu Spiral Graphics.

[2]Dostupný z: https://texture.ninja
[3]Dostupný z: http://spiralgraphics.biz/packs



Obrázek 7.1: Model rytíře v programu Blender před použitím modifikátoru Triangulate.



Obrázek 7.2: Vlevo model ledového kvádru v programu Blender. Vpravo ten samý model použitý ve hře s alfa kanálem jeho barvy nastaveným na hodnotu menší než jedna, díky čemuž je model průhledný.

## 7.3 Orbitální kamera

V aplikaci je implementován objekt kamery a orbitální ovládání kamery. Hráč může kamerou posouvat, otáčet a přibližovat či oddalovat. Ke zprovoznění takové kamery je potřeba správně převést dvourozměrný vstup uživatele (pohyb myší, stisknutí klávesy) na změnu trojrozměrných souřadnic kamery s ohledem na současnou perspektivu uživatele k vykreslované scéně.

Zaprvé je potřeba umět vypočítat pozici kamery z pozice středu scény, vzdálenosti kamery a její rotace. Kritický úsek tohoto výpočtu ukazuje výpis 7.3. S tímto výpočtem se stává změna rotace a vzdálenosti kamery triviální. Stačí pouze upravit hodnoty distance, rotationX nebo rotationY a přepočítat pozici kamery.

```
1  camera->LookAtPosition.x = positionX;
2  camera->LookAtPosition.y = positionY;
3  camera->LookAtPosition.z = positionZ;
4
5  auto correction = cosf(radians(rotationY));
6
7  camera->EyePosition.x = positionX +
8         distance * correction * cosf(radians(rotationX));
9  camera->EyePosition.y = positionY +
10        distance * sinf(radians(rotationY));
11 camera->EyePosition.z = positionZ +
12        distance * correction * sinf(radians(rotationX));
```

Výpis 7.3: Úryvek kódu pro vypočítání pozice pozorovatele z pozice středu scény

Implementovaný výpočet pro pohyb kamery (anglicky panning) ukazuje výpis 7.4.

```
1  // Faster panning when zoomed out
2  deltaX *= sqrt(distance);
3  deltaZ *= sqrt(distance);
4
5  positionX +=
6      - sinf(radians(rotationX)) * deltaX
7      - cosf(radians(rotationX)) * deltaZ;
8
9  positionZ +=
10       cosf(radians(rotationX)) * deltaX
11     - sinf(radians(rotationX)) * deltaZ;
```

Výpis 7.4: Úryvek kódu pro pohyb kamery

Součástí třídy pro ovládání kamery je také metoda, která konvertuje dvourozměrné souřadnice pozice kurzoru v okně aplikace na dvourozměrné souřadnice roviny XZ. Tohoto je dosaženo zjednodušenou verzí metody Raycast (výpis 7.5). Nejdříve je za využití funkce *unProject* vypočítán směrový vektor, jenž definuje přímku (nebo také paprsek – ang. ray), která prochází pomyslným umístěním kurzoru myši v prostoru a předpokládaným umístěním oka uživatele. Poté je nalezen průsečík této přímky s rovinou XZ, ve které jsou umístěna

hrací pole. Tato funkce umožňuje základní herní úkony jako vybírání jednotek nebo cílení schopností.

```
1  vec2 CameraControl::CalculateMousePosition(float x, float y,
       Simple_geSGRenderer* renderer) const
2  {
3      auto winWidth = renderer->GetWindowWidth();
4      auto winHeight = renderer->GetWindowHeight();
5      auto view = renderer->GetViewMatrix();
6      auto projection = renderer->GetPerspectiveMatrix();
7
8      y = winHeight - y - 1;
9
10     auto wc1 = unProject(vec3(x, y, 0.f), view, projection, vec4(0, 0,
           winWidth, winHeight));
11     auto wc2 = unProject(vec3(x, y, 1.f), view, projection, vec4(0, 0,
           winWidth, winHeight));
12     auto dwcy = wc2.y - wc1.y;
13     if (abs(dwcy) < 0.0001f)
14     {
15         dwcy = 0.0001f;
16     }
17     double f = wc1.y / dwcy;
18     auto x2d = wc1.x - f * (wc2.x - wc1.x );
19     auto z2d = wc1.z - f * (wc2.z - wc1.z );
20
21     return vec2(x2d, z2d);
22 }
```

Výpis 7.5: Úryvek kódu pro získání pozice myši

## 7.4 Hledání cest

Hledání cest na hrací ploše je v aplikaci implementováno ve třídě Pathfinding ve funkci FindPath (najdi cestu) pomocí algoritmu A*. Implementace je obdobná jako u obecné verze algoritmu popsané v podkapitole 4.1. Její konkrétní podoba vypadá následovně:

1. Inicializace všech proměnných a počátečních hodnot:
   (a) Uzavřený seznam (closedSet) je inicializován jako prázdný seznam ukazatelů na hrací pole (unordered_set<Field*>).
   (b) Otevřený seznam (openSet) je inicializován jako seznam ukazatelů na hrací pole, obsahující startovní hrací pole (origin).
   (c) Je vytvořena mapovací datová struktura cameFrom, která mapuje vždy jeden ukazatel hracího pole na jiný (unordered_map<Field*, Field*>). Tato struktura bude sloužit k určení cesty k danému hracímu poli. Nahrazuje tak potřebu mít v hracích polí odkazy na rodičovské uzly. Důvodem pro toto řešení je dodržení modularity. Hrací pole by nemělo obsahovat atributy specifické pouze pro hledání cest.

(d) Z podobných důvodů jsou vytvořeny mapy hodnot g (gScore) a hodnot f (fScore). Pro připomenutí – hodnota g určuje cenu cesty do daného pole z pole startovního a hodnota f určuje součet hodnoty g a výsledku heuristické funkce. Pro všechna neobsazená hrací pole jsou obě hodnoty nastaveny na 9999 (může být jakékoliv vysoké číslo). Pro startovní pole je hodnota g nastavena na 0 a hodnota f je vypočítána pomocí heuristické funkce. Hodnota f je u startovního uzlu irelevantní a počítá se zde zbytečně.

2. Následující kroky jsou opakovány dokud není otevřený seznam prázdný:

   (a) Z otevřeného seznamu je vybráno pole s nejnižší hodnotou f. Toto pole je označeno za současné pole (current).

   (b) Pokud je současné pole zároveň polem cílovým, běh algoritmu je ukončen úspěšně a výsledná cesta je vytvořena použitím mapy cameFrom a navrácena.

   (c) Současné pole nebylo polem cílovým – algoritmus pokračuje. Současné pole je odstraněno z otevřeného seznamu a přidáno do uzavřeného seznamu.

   (d) Je vytvořen dočasný seznam všech validních sousedních polí současného pole. Validní sousední pole musí být neobsazené a v případě diagonálního pohybu musí být volný celý čtverec $2 \times 2$, jak bylo popsáno v podkapitole 5.1.

   (e) Pro každé z těchto sousedních polí je provedeno:

      i. Pokud je sousední pole v uzavřeném seznamu, je ignorováno a pokračuje se dalším sousedním polem.

      ii. Pokud sousední pole ještě není v otevřeném seznamu, je tam přidáno.

      iii. Kandidát na novou hodnotu g (new_gScore) je vypočítán jako hodnota g současného pole +1 (všechny kroky jsou cenově ekvivalentní).

      iv. Pokud je kandidát na novou hodnotu g vyšší než současná hodnota g zkoumaného sousedního pole, je toto pole ignorováno a pokračuje se dalším sousedním polem.

      v. Pokud je hodnota f současného pole vyšší než hodnota f pole z mapy cameFrom pro sousední pole, je toto pole ignorováno a pokračuje se dalším sousedním polem.

      vi. Do sousedního pole je nejkratší cesta z pole současného. Tato skutečnost je zaznačena do mapy cameFrom, sousednímu poli je přiřazena hodnota kandidáta na novou hodnotu g a hodnota f tohoto pole je vypočítána jako součet nové hodnoty g a výsledku heuristické funkce.

3. Cesta neexistuje – algoritmus vrací prázdný seznam místo cesty k cíli.

Heuristická funkce počítá svou hodnotu jako 80 % ze vzdušné čáry mezi zkoumaným hracím polem a polem cílovým (viz výpis 7.6). Hodnota 80 % byla určena experimentálně.

```
1 float Pathfinding::HeuristicCostEstimate(Field* origin, Field* target)
2 {
3     auto deltaX = abs(origin->GetX() - target->GetX());
4     auto deltaY = abs(origin->GetY() - target->GetY());
5         return 0.8f * sqrt(float(pow(deltaX, 2) + pow(deltaY, 2)));
6 }
```

Výpis 7.6: Heuristická funkce algoritmu A*

Tento výpočet teoreticky nesplňuje podmínku nutnou pro optimálnost algoritmu A*. V případě dlouhých diagonálních pohybů na otevřených prostranstvích může tato heuristická funkce vracet vyšší hodnoty než je skutečně nejkratší cesta. Důvodem je to, že diagonální krok má skutečnou cenu 1, ale tato heuristická funkce odhadne cenu stejného úseku jako $0,8 \times \sqrt{1^2 + 1^2} \doteq 1,13$. Korektní heuristická funkce by tedy měla vracet pouze větší z hodnot deltaX a deltaY. U té by ale nastal jiný problém. Jednotky by mohly chodit do cíle oklikou, takzvaně „do L“, což by sice nemělo dopad na hru, ale po vizuální stránce by trasa jednotky působila nepřirozeně. Alternativním řešením by bylo použití hodnoty 70 % místo 80, protože $0,7 \times \sqrt{1^2 + 1^2} \doteq 0,99$, což už je hodnota menší než 1. V praxi se zdá, že minimálně pro hlavní herní mapu nepředstavuje použitý nekorektní výpočet problém v optimálnosti algoritmu.

Třída Pathfinding obsahuje také funkci GetAllPossibleTargets (získej všechny možné cíle). Tato funkce slouží k získání validních cílů pro pohyb jednotky a uložení ceny cesty do daného pole. Tyto pole jsou pak ve hře zvýrazněna, aby hráč viděl na která pole se může s jednotkou přesunout. Ještě důležitějším využitím je optimalizace výkonu herní UI, kde tato funkce kompletně nahrazuje použití funkce FindPath.

Implementace funkce GetAllPossibleTargets je velice podobná implementaci hledání cesty. Hlavním rozdílem je absence heuristické funkce a s tím související absence hodnot f. Hodnota g je ukládána přímo do objektu hracího pole pod názvem CostToGetHere. Hodnota je ukládána, aby nemusela být opakovaně počítána dokud se nezmění výběr jednotky nebo situace na hracím plánu. Také se vůbec neukládá cesta do zkoumaných polí, není důležitá. Funkce slouží pouze k získání validních cílů pohybu a uložení délky cesty do daného cíle. Místo při nalezení cíle je běh funkce ukončen při překročení maximální povolené vzdálenosti (zpravidla určené množstvím energie jednotky).

## 7.5 Implementace herní UI

Pro ovládání simulovaného hráče v této hře byla použita strategie Minimax s vlastní modifikací. Hra QueensRegicide se vyznačuje obrovským množstvím možných tahů v běžné herní situaci. Řádově se počet možných tahů pohybuje od vyšších stovek a může přesáhnout i hranici tisíce tahů. To už samo o sobě daleko převyšuje možnosti her šachy a Go. Navíc ale herní UI musí vymyslet sekvenci 3 tahů za sebou. Možných trojkombinací tahů může být řádově miliardy. Při testování bylo zjištěno, že ohodnocení jednoho stavu při kompletním využití právě jednoho jádra procesoru na běžném stolním počítači trvá řádově stovky nanosekund. Ohodnocení miliardy herních stavů by tedy trvalo desítky až stovky hodin. Je tedy potřeba zredukovat množství prohledávaných stavů na řádově desítky tisíc, aby prohledávání trvalo jen několik vteřin.

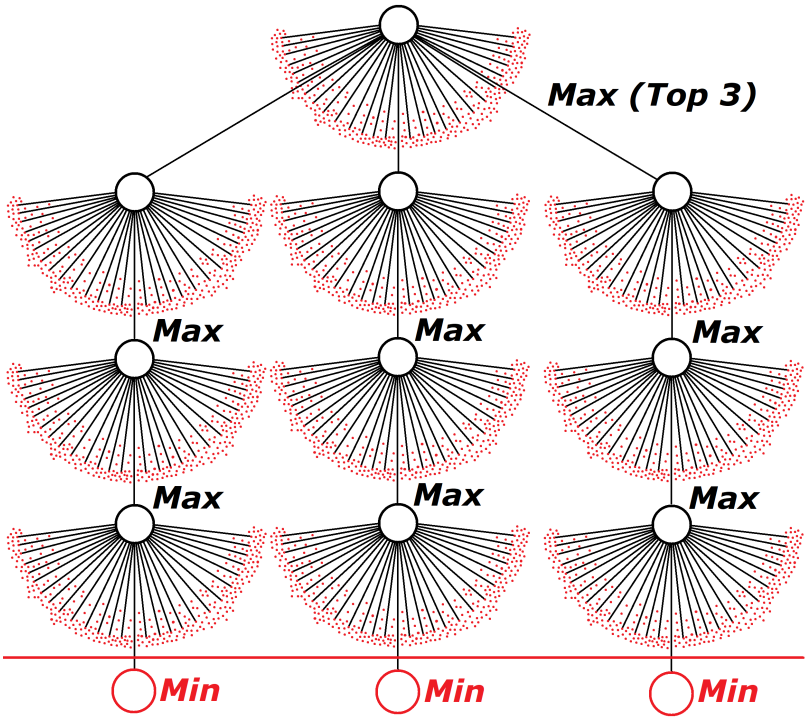
Za tímto účelem byla vytvořena modifikace strategie Minimax. Tato modifikace v každé generaci/hloubce uzlů vybírá jen několik nejlépe ohodnocených uzlů a pouze ty dále rozvíjí. Počet rozvinutých uzlů se mezi generacemi liší a je stanoven parametry z třídy AiConfiguration. V současné verzi hry jsou tyto parametry nastaveny podle tabulky 7.1. Průchod stromem je znázorněn na obrázku 7.3.

S omezeným počtem zkoumaných stavů je obzvlášť důležité, aby byly zkoumány jen ty skutečně nejlepší stavy. K tomuto účelu slouží evaluační funkce EvaluateGameState ve třídě AiGameStateEvaluator. Tato funkce zvažuje, mimo jiné, následující aspekty herního stavu:

- Výhra jednoho z hráčů
- Počet jednotek naživu

| Obtížnost UI | Jednoduchá | Střední | Těžká |
|---|---|---|---|
| Počet podstromů 1. generace | 3 | 4 | 5 |
| Počet podstromů 2. generace | 1 | 2 | 3 |
| Počet podstromů 3. generace | 1 | 1 | 1 |

Tabulka 7.1: Míra větvení stavového prostoru u různých obtížností herní UI




Obrázek 7.3: Rozvíjení uzlů u jednoduché obtížnosti UI

- Životy a energie jednotek
- Negativní i pozitivní stavy jednotek
- Nebezpečí hrozící jednotkám
- Aktivita jednotek
- Vzdálenost jednotek od středové čáry hrací plochy
- Soudržnost formace hráčových jednotek
- Klaustrofobie jednotek (aby se mohly pohybovat)

Jednotlivé obtížnosti UI se kromě počtu rozváděných uzlů ve stavovém prostoru liší i v některých parametrech evaluační funkce. Těžší verze UI jsou méně agresivní, více si chrání své figury, především královnu a udržují své jednotky blíže u sebe.

# Kapitola 8

# Testování

Pro účely testování byl vytvořen dotazník, který obsahoval instrukce ke stažení a spuštění hry, seznámení se s pravidly a ovládáním, spuštění hry proti jednoduché UI a následném zhodnocení hry v několika kategoriích. Kategorie byly: srozumitelnost pravidel, ovládání hry, uživatelské rozhraní, vizuální stránka, rychlost UI, kvalita UI (její obtížnost) a kvalita hry (jak zábavná hra byla).

Doba potřebná k provedení testu byla odhadnuta na 30 minut. Cizí člověk pravděpodobně nebude ochotný tolik času obětovat, a proto bylo testování provedeno pouze mezi přáteli. V instrukcích dotazníku bylo apelováno na testery, aby vyplňovali dotazník podle pravdy a nebáli se dávat negativní hodnocení. Celkem hru vyzkoušelo a vyplnilo dotazník šest lidí.

Testeři hodnotili kategorie od jedné do pěti, kde skóre jedna znamenalo nejhorší hodnocení a skóre pět nejlepší. Součástí jejich hodnocení byly i slovní komentáře. Průměrná hodnocení dopadla následovně:

| | |
|---|---|
| Srozumitelnost pravidel | 4,17 |
| Ovládání hry | 3,75 |
| Uživatelské rozhraní | 4 |
| Vizuální stránka | 3,5 |
| Rychlost AI | 4,5 |
| Kvalita AI | 4,67 |
| Kvalita hry | 4,17 |

## 8.1 Srozumitelnost pravidel

Průměrné skóre: 4,17
Vzhledem ke komplexitě pravidel se dá tento výsledek považovat za výborný. Ve slovních komentářích měli testeři občas výtky ke konkrétním pravidlům, ale kromě jedné se žádná výtka nevyskytla více než jednou. Onou opakovanou výtkou bylo, že někteří testeři nevěděli, jaký význam má centrální pole s trůnem. K tomuto pravděpodobně došlo kvůli tomu, že jednotka královny nemá žádné aktivní schopnosti a testery nenapadlo přečíst si popis její pasivní schopnosti *Claim Throne*, kde je funkce trůnu vysvětlena.

## 8.2 Ovládání hry

Průměrné skóre: 3,75
Testeři měli častý problém s rolí levého a pravého tlačítka myši. Tento fakt by se dal přisoudit tomu, že jsou testeři z podobných her zvyklí na odlišné ovládání. Jeden tester očekával, že bude moct posouvat kamerou při najetí na okraj obrazovky. Také se objevil jeden požadavek na možnost vypnout automatické sledování pohybujících se jednotek.

## 8.3 Uživatelské rozhraní

Průměrné skóre: 4
Testeři chtěli označení nepoužitelných schopností v panelu schopností, vysvětlení některých prvků rozhraní, větší uživatelské rozhraní, vracení kamery na poslední známou pozici po skončení tahu AI a pár dalších drobností.

## 8.4 Vizuální stránka

Průměrné skóre: 3,5
Kategorie, která dopadla nejhůř a zároveň kategorie, ve které se názory testerů nejvíce rozcházely. Dvakrát obdržela skóre 2 a naopak dvakrát obdržela skóre 5. Testeři měli velice častý problém s barevným odlišením týmů, odlišením typů jednotek a odlišením běžných žlutých políček od zeleně zvýrazněných políček.

## 8.5 Rychlost AI

Průměrné skóre: 4,5
Rychlost AI byla hodnocena velice pozitivně. Jeden tester zmínil, že by byl rád za možnost nastavit si, jak dlouho hra čeká po každém tahu AI na vstřebání tahu hráčem. Tato hodnota byla v době testování nastavena na 2 sekundy.

## 8.6 Kvalita AI

Průměrné skóre: 4,67
Nejlépe hodnocená kategorie. Paradoxně to ale zrovna u této kategorie není čistě pozitivní věc. Většina testerů se shodla na tom, že jednoduchá obtížnost AI je příliš těžká.

## 8.7 Kvalita hry

Průměrné skóre: 4,17
Zde se testeři rozdělili do dvou táborů. První skupina sice částečné uznání hře dala, ale jelikož jim hra nesedla žánrově, tak hodnotili skórem 3. Zbytek testerů hodnotil hru velice dobře a někteří pokračovali v hraní i mimo rámec testování. Mezi slovními komentáři se objevil požadavek na přidání možnosti online hry více hráčů.

# Kapitola 9

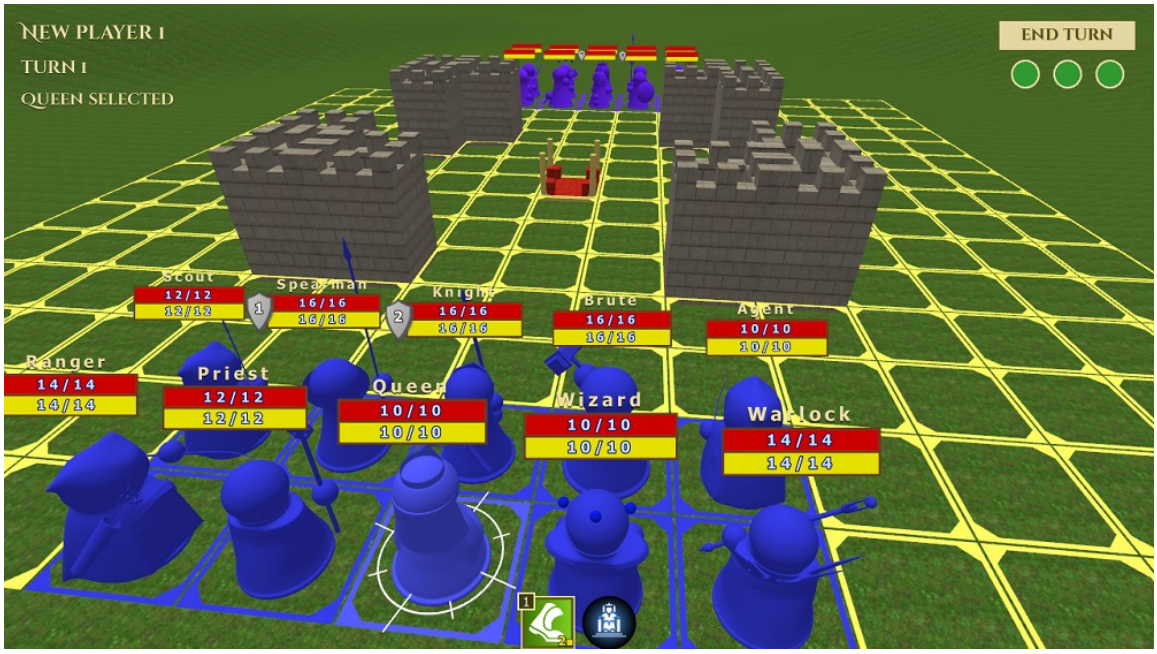
# Závěr

Celkově považuji práci za docela zdařenou. Všechny stanovené cíle byly splněny a některé části hry dokonce překročily rámec původního plánu. Testeři hodnotili výslednou hru převážně kladně. Největším úspěchem práce je herní AI, která je schopná rychlých nerušivých výpočtů a dokáže začátečníky bez problému porážet.

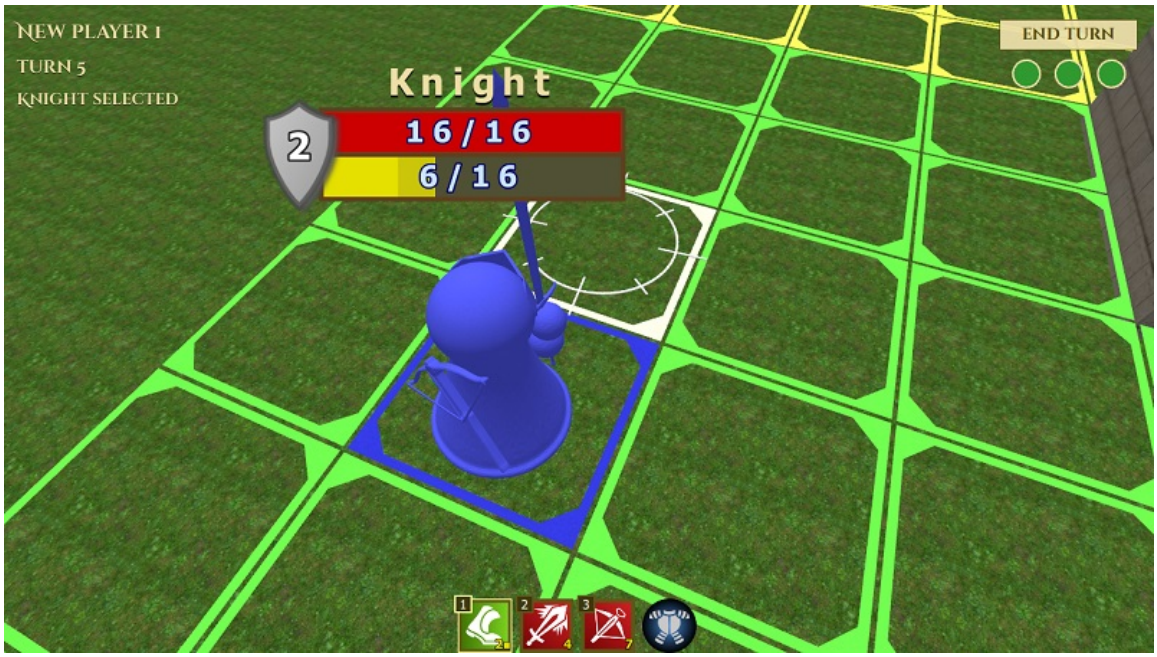
Funkčnost hry byla otestována na operačním systému Windows verzí 7, 8 a 10. Aplikaci by teoreticky mělo být možné zkompilovat i na linuxových systémech, avšak tato funkcionalita nebyla nikdy otestována. Hra nefunguje s integrovanými grafickými kartami od Intelu. U strojů s vícero grafickými kartami lze v grafickém ovladači karty nastavit, kterou kartu má hra používat.

Na základě zpětné vazby testerů bylo provedeno několik úprav a vznikla verze hry 1.01. Mezi provedené úpravy patří hlavně zlepšení palety funkčních barev hry (barvy hráčů a hracích polí) s důrazem na vyšší kontrast. Rozdíl v barvách lze vidět na obrázcích 9.1, 9.2, 9.3 a 9.4. Další úpravou bylo zvýšení rozdílů mezi obtížnostmi umělé inteligence a celkové snížení obtížnosti všech úrovní UI.

Hra má oblastí, ve kterých by se dala dále rozvíjet. Jako inspirace může sloužit zpětná vazba testerů popsaná v kapitole 8. Konkrétně by mohla být zajímavá implementace online módu pro hru více hráčů. Osobně bych navrhl přidání modelových animací, zlepšení kvality modelů s důrazem na vyšší kontrast mezi jednotkami, přidání zajímavých scénářů (např. trénovací koncovky), vytvoření interaktivního výukového scénáře, přidání kampaně s příběhem nebo implementaci nové herní UI s využitím například evolučních algoritmů.




Obrázek 9.1: Vzhled hry verze 1.00 (před závěrečnými úpravami)




Obrázek 9.2: Vzhled hry verze 1.00 (před závěrečnými úpravami)


Obrázek 9.3: Vzhled hry verze 1.01 (po závěrečných úpravách)


Obrázek 9.4: Vzhled hry verze 1.01 (po závěrečných úpravách)

# Literatura


[1] Arulkumaran, K., Cully, A. a Togelius, J. AlphaStar: An Evolutionary Computation Perspective. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. New York, NY, USA: Association for Computing Machinery, 2019, s. 314–315. GECCO '19. DOI: 10.1145/3319619.3321894. ISBN 9781450367486. Dostupné z: https://doi.org/10.1145/3319619.3321894.

[2] Chaslot, G. *Monte-Carlo Tree Search*. 2010. ISBN 978-90-8559-099-6.

[3] Colledanchise, M. a Ogren, P. *Behavior Trees in Robotics and AI: An Introduction*. 2018. ISBN 9780429489105.

[4] Cui, X. a Shi, H. A*-based Pathfinding in Modern Computer Games. *IJCSNS International Journal of Computer Science and Network Security*. 2011, sv. 11, č. 1.

[5] Dem'yanov, V. a Malozemov, V. *Introduction to Minimax*. Dover Publications, 1990. Dover Books on Mathematics. ISBN 9780486664231. Dostupné z: https://books.google.cz/books?id=xRT-AgAAQBAJ.

[6] Ian Millington, J. F. *Artificial Intelligence for Games*. 2. vyd. Taylor & Francis Group, 2009. ISBN 978-0-08-088503-2.

[7] Kaplan, A. a Haenlein, M. Siri, Siri, in my hand: Who's the fairest in the land? On the interpretations, illustrations, and implications of artificial intelligence. *Business Horizons*. 2019, sv. 62, č. 1, s. 15–25. DOI: https://doi.org/10.1016/j.bushor.2018.08.004. ISSN 0007-6813. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0007681318301393.

[8] Knuth, D. E. a Moore, R. W. An analysis of alpha-beta pruning. *Artificial Intelligence*. 1975, sv. 6, č. 4, s. 293–326. DOI: https://doi.org/10.1016/0004-3702(75)90019-3. ISSN 0004-3702. Dostupné z: https://www.sciencedirect.com/science/article/pii/0004370275900193.

[9] OpenAI, :, Berner, C., Brockman, G., Chan, B. et al. *Dota 2 with Large Scale Deep Reinforcement Learning*. 2019. Dostupné z: https://openai.com/projects/five/.

[10] Osborne, M. J. *An introduction to game theory*. New York: Oxford University Press, 2004. ISBN 978-0-19-512895-6.

[11] Pavliska, J. *Umělá inteligence v moderních počítačových hrách*. Brno, CZ, 2009. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/9031/.

[12] Poole, D. L. *Computational Intelligence: A Logical Approach.* New York: Oxford University Press, 1998. ISBN 978-0-19-510270-3.

[13] Russell, S. J. a Norvig, P. *Artificial intelligence : a modern approach.* Upper Saddle River, N.J. : Prentice Hall/Pearson Education, 2003. ISBN 0137903952.

[14] Wlosok, J. *Umělá inteligence v počítačových hrách.* Ostrava, CZ, 2017. Diplomová práce. Vysoké škola báňská - Technická univerzita Ostrava. Dostupné z: http://hdl.handle.net/10084/119037.

# Příloha A

# Obsah přiloženého paměťového média

- `/` ..... kořenový adresář přiloženého CD
  - `QueensRegicide`
    - `readme.txt`
    - `blender_materials.zip` ..... soubory modelů ve formátu .blend
    - `QueensRegicide_v101.zip` ..... spustitelná verze hry pro 64x Windows[1]
    - `win64_libs.zip` ..... dll soubory pro 64x Windows
    - `source_code.zip` ..... zdrojové soubory[2]
  - `2021-xhrana02-hra-v-opengl.flv`
  - `BP_xhrana02.pdf` ..... text bakalářské práce
  - `BP_LaTeX_source.zip` ..... zdrojový tvar textu

[1] Plně funkční, dll soubory již obsaženy.
[2] Aktuální verzi těchto souborů lze najít na URL https://github.com/xhrana02/queens-code

# Příloha B

# Manuál

Tato příloha detailně popisuje pravidla a ovládání hry. Po jejím přečtení by měl být čtenář schopný hru ovládat a orientovat se v ní.

## B.1 Základní pravidla hry

V hlavní nabídce hry se nachází tlačítko *Rules* se stručným přehledem pravidel. Zde budou pravidla vysvětlená do detailu a to včetně těch, které se v *Rules* nenachází. Všechny zde uvedené informace se primárně vztahují ke standardní formě hry. Ve scénářích se mohou některé detaily lišit.

Hra se hraje ve dvou hráčích na čtvercové síti $15 \times 15$ polí. Každé pole může být obsazeno maximálně jednou jednotkou. Na poli se také může nacházet překážka ve formě hradby nebo trůn, jehož funkce bude vysvětlena v pokročilých pravidlech.

Každý z hráčů má k dispozici 10 unikátních jednotek. Jednotky se liší vizuální podobou, maximálním počtem životů a schopnostmi. Nejdůležitější z těchto jednotek je *Queen* (královna). Cílem hry je zabít protihráčovu královnu nebo všechny jeho ostatní jednotky, což by v drtivé většině případů vedlo i ke smrti královny. Královna, jenž je ekvivalent šachového krále, se totiž na rozdíl od šachů nedokáže bránit téměř vůbec.

Jednotky mají dva základní zdroje. Životy a energii. Energií se platí za používání schopností a hodnota energie nemůže převýšit aktuální hodnotu životů. Jakmile životy jednotky klesnou na nulu, umírá a je okamžitě odstraněna z herního plánu.

Hráči se střídají v tazích. Jeden tah se skládá ze tří příkazů, avšak hráč může svůj tah ukončit předčasně, nepřejeli si již v tomto kole nic dalšího dělat. Příkazy mohou být rozděleny mezi libovolný počet jednotek.

Jeden příkaz představuje jedno použití schopnosti vybrané jednotky. Mezi tyto schopnosti se řadí i schopnost *Movement* (pohyb), kterou všechny jednotky sdílí a díky které se mohou po hrací ploše pohybovat. Schopnost pohybu je ve hře vizuálně označena zeleným pozadím ikonky. Každá jednotka má kromě pohybu i tři další unikátní schopnosti. Schopnosti mohou být aktivní nebo pasivní.

Aktivní schopnosti používají čtvercové ikonky a dělí se na čistě útočné, víceúčelové a neškodné. Čistě útočné (např. Longsword Attack) slouží k prostému poškozování soupeřových jednotek a jsou označeny červeným pozadím ikonky. Víceúčelové (např. Shield Slam) kombinují poškození s dalšími formami ovlivnění soupeřových jednotek a označeny jsou žlutým pozadím ikonky. Neškodné (např. Healing Circle) nijak nepůsobí na nepřítele, ale místo

toho napomáhají vlastním jednotkám, a označeny jsou modrým pozadím. Všechny aktivní schopnosti mají cenu, která musí být pro jejich použití zaplacena.

Pasivní schopnosti nelze použít příkazem, mají buď trvalý pasivní efekt nebo se samy aktivují při splnění jejich podmínek a označeny jsou kruhovou ikonkou s černým okrajem.

## B.2 Herní pojmy

V této sekci jsou vysvětleny některé pojmy, jenž se v tomto textu používají nebo na ně narazíte přímo ve hře. Pojmy jsou nejdříve uvedeny anglicky, tak jak jsou používány ve hře a po té v českém překladu, tak jak jsou používány zde v textu.

- Unit, Jednotka – hrací figurka
- Ability, Schopnost – akce, kterou může jednotka použít
- Move, Command, Tah, Příkaz – použití schopnosti jednotky
- Turn, Kolo – kolo hráče skládající se až ze tří tahů
- Field, Tile, Pole – hrací pole, na kterém může jednotka stát
- Hit Points, HP, Životy – zdroj držící jednotku při životě
- Energy, EN, Energie – zdroj, kterým se platí za použití schopností
- Regeneration, Regenerace – proces, který obnovuje zdroje všem jednotkám na začátku jejich kola
- Rest, Odpočinek – proces, který na konci kola obnovuje zdroje jednotkám, které v daném kole nepoužily žádnou aktivní schopnost
- Damage, Poškození – efekt, který snižuje zdroje cílené jednotky
- Armor, Zbroj – vlastnost jednotky, která snižuje příchozí poškození
- Stun, Omráčení – stav jednotky, který ji znemožňuje pohyb
- Restless, Neklidný – stav jednotky, který jí znemožňuje odpočívat
- Melee, Na blízko – druh cílení schopností, který umožňuje zacílit čtyři kolmě sousedící pole
- Line, Přímý – druh cílení schopností, který umožňuje zacílit pole v kolmých a diagonálních směrech
- Indirect, Nepřímý – druh cílení schopností, který umožňuje zacílit jakékoliv pole v dosahu

## B.3 Pokročilá pravidla hry

V této sekci vysvětlím v bližších detailech ty části hry, které nebyly součástí základních pravidel.

### B.3.1 Speciální typy hracích polí

*Hradba* znemožňuje průchod jakékoliv jednotky a chová se jako překážka pro přímé útoky. Zároveň nemůže být cílem jakékoliv schopnosti, ať už přímé či nepřímé. Ve standardní hře je rozmístěno 12 polí hradby ve čtyřech skupinách po třech. Tyto hradby představují rohy trůnního sálu, v jehož středu se nachází políčko s trůnem.

*Trůn* se nachází uprostřed hracího plánu. Pokud na něm královna skončí své kolo a není omráčená, tak se všechny soupeřovy jednotky stávají na jedno kolo neklidnými. Tento efekt se pak může opakovat každé následující kolo, dokud platí původní podmínky. Pole s trůnem není bráno jako překážka a jednotky se po něm mohou volně pohybovat.

### B.3.2 Druhy poškození a zbroj

*Normal damage* (normální poškození) je nejběžnější druh poškození a cílené jednotce snižuje nejdříve energii a pokud jednotce již žádná energie nezbývá, tak zredukuje i její životy. Jeden zdroj poškození může naráz snížit energii i životy, pokud je hodnota energie jednotky dostatečně nízká. Např. jednotka, která má šest životů a dvě energie a je cílem šesti bodů normálního poškození, ztratí své dvě energie a k tomu ještě čtyři životy.

*EN damage* (poškození energie) je schopné snižovat pouze energii jednotky a pokud cílená jednotka žádnou další energii nemá, tak všechno přebytečné poškození propadává bez efektu. Na první pohled se jedna o nejslabší formu poškození, ale jeho poměr ceny ku množství poškození ho v některých případech dělá efektivnější volbou než použití normálního poškození.

*HP damage* (poškození životů) je nejsilnější druh poškození. Cílené jednotce snižuje přímo její životy a pokud má tato jednotka množství energie na maximu, tak přichází i o energii (k tomuto efektu dochází kvůli základnímu pravidlu o maximálním množství energie).

Pokud schopnost používá více než jeden druh poškození, říká se tomu *kombinované poškození*. V tomto případě mají různé druhy poškození pevně dané pořadí ve kterém se provádí. První proběhne poškození životů, druhé poškození energie a poslední se provede normální poškození. Toto pořadí nabízí útočníkovi maximální možnou efektivitu.

*Zbroj* je celočíselná hodnota, která snižuje příchozí poškození v poměru jedna ku jedné. Snížení poškození probíhá v přesně opačném pořadí než se poškození uděluje. Toto pořadí nenadržuje ani jedné straně, ale řídí se logickou funkcí zbroje. Jako první sníží normální poškození, které většinou pochází z běžných zbraní (jako třeba meče a luky). Druhé je sníženo poškození energie, které je způsobeno úderem štítu a zásahem blesku. Jako poslední je sníženo poškození životů, jenž představuje zásah zbraní, která má zbroj prorážet (jako válečné kladivo nebo kuše).

### B.3.3 Pravidla pohybu

Pohyb je možný uskutečnit v kolmých směrech a v případě, že k diagonálnímu směru jsou oba sousedící kolmé směry volné (tzn. není na nich žádná jednotka či překážka), je možný i diagonální pohyb. Cena pohybu se platí za každé prošlé pole, ale počítá se vždy jen jako jeden tah. Hra vždy najde nejkratší cestu k cíli.

Jednotka *Scout* (průzkumník) vlastní upravenou verzi schopnosti pohybu, která má poloviční cenu energie za prošlé políčko. Kromě pohybu existují dvě další schopnosti, které dokáží s jednotkou pohnout. První je *Dash* jednotky *Agent*, která ho dokáže přemístit na

krátkou vzdálenost, a druhou je *Charge* jednotky *Brute* (surovec), která ho přemístí k cílené jednotce a tuto cílenou jednotku může také o jedno políčko posunout.

### B.3.4 Stavy

Ve hře existují dva negativní stavy. První je *Restless* (neklidný), jenž znemožňuje ovlivněné jednotce odpočívat, a druhý je *Stunned* (omráčený), který jednak znemožňuje jednotce provádět jakékoliv akce (včetně pasivně aktivních) a zároveň se postižená jednotka stává neklidnou. Stavy mohou mít dobu trvání delší než jedno kolo. Tato doba trvání se snižuje vždy na konci kola ovlivněné jednotky.

Ve hře lze nalézt také dva *Buffy*, což jsou pozitivní stavy. Jedním je *Blessing* (požehnání), které jednotce na poměrně dlouhou dobu zvyšuje zbroj a regeneraci energie. Druhým je *Demon Shield* (ďábelský štít) a ten výrazně zvyšuje zbroj jednotky, ale pouze na krátkou dobu.

## B.4 Jednotky

Jednotky se dělí do čtyř kategorií: *Royalty* (královská rodina), *Fighters* (bojovníci), *Specialists* (specialisté) a *Spellcasters* (kouzelníci).

Do královské rodiny patří pouze *Queen* (královna). Jedná se o bezbrannou jednotku, která musí být ostatními jednotkami chráněna za každou cenu.
Královna:

- *Claim Throne, passive* – Pokud stojí na poli s trůnem, všechny soupeřovy jednotky se stávají neklidnými.

Skupina bojovníků se skládá z *Knight* (rytíře), *Spearman* (kopiníka) a *Brute* (surovce). Mají největší množství maximálních životů a k tomu navíc pasivní schopnosti, které jim pomáhají přežít. Jejich další výhodou jsou velice silné útoky na blízko.
Rytíř:

- *Longsword Attack, melee* – Způsobí obrovské množství normálního poškození.
- *Crossbow Attack, line 2-8* – Způsobí poškození životů a normální poškození.
- *Knight's Armor, passive* – Rytíř má bonus ke zbroji.

Kopiník:

- *Spear Attack, line 1-2* – Způsobí poškození životů a normální poškození.
- *Shield Slam, line* – Způsobí velké množství poškození energie a omráčení.
- *Spartan, passive* – Kopiník má bonus ke zbroji a způsobí poškození životů každé nepřátelské jednotce, která se pokusí odejít ze sousedícího pole.

Surovec:

- *Warhammer Attack, melee* – Způsobí poškození životů a velké množství normálního poškození.

- *Charge, line 2-4* – Surovec se rozběhne k cílené jednotce způsobí ji poškození životů. Pokud je pole za ní volné, posune jednotku něj a sám zaujme její předchozí pozici. Pokud volné není, zastaví se surovec na poli před jednotkou, ale způsobí ji dvojnásobné poškození a omráčení.
- *Toughness, passive* – Surovec si během regenerace doplňuje i životy.

Mezi specialisty se řadí *Ranger* (hraničář), *Scout* (průzkumník) a *Agent* (agent). Hraničář je velice efektivní pro boj na dlouhé vzdálenosti. Průzkumník se dokáže velice rychle pohybovat po hrací ploše a agent dokáže velice efektivně likvidovat soupeřovy jednotky.
Hraničář:

- *Longbow Attack, line 2-9* – Způsobí velké množství normálního poškození.
- *Short Sword Attack, melee* – Způsobí normální poškození.
- *Quick Reflexes, passive* – Ignoruje část veškerého poškození z přímých (*line*) útoků.

Průzkumník:

- *Short Sword Attack, melee* – Způsobí normální poškození.
- *Dirty Trick, melee* – Způsobí omráčení.
- *Runner, passive* – Průzkumník platí pouze polovinu ceny při pohybu.

Agent:

- *Dagger Attack, melee* – Způsobí poškození životů.
- *Shortbow Attack, line 2-7* – Způsobí normální poškození.
- *Dash, indirect 1-4* – Agent skočí na prázdné pole.

Do poslední skupiny patří *Warlock* (černokněžník), *Priest* (kněz) a *Wizard* (čaroděj). Tito kouzelníci mají ofenzivní schopnosti na střední až velkou vzdálenost a podpůrné schopnosti na krátkou vzdálenost.
Černokněžník:

- *Fireball, line 1-6* – Způsobí normálního poškození cíli a všem okolo cíle.
- *Demon Shield, indirect 0-3* – Dočasně výrazně zvýší zbroj cílené jednotky.
- *Soul Steal, melee* – Způsobí poškození životů. Způsobené poškození černokněžníkovi doplní jeho vlastní životy.

Kněz:

- *Smite, indirect 1-4* – Způsobí normální poškození a omráčení.
- *Blessing* – Všechny přátelské jednotky v okolí mají dočasně zvýšenou zbroj a regeneraci životů.
- *Healing Circle* – Obnoví životy a energie všem přátelským jednotkám okolo.

Čaroděj:

- *Lightning Bolt, line 2-8* – Způsobí poškození energie a normální poškození.
- *Ice Block, indirect 0-4* – Způsobí normální poškození.
- *Meditation, passive* – Čaroděj má zvýšenou obnovu energie při odpočinku.

## B.5 Ovládání hry

Hra obsahuje ovládání pomocí klávesnice a myši. Myš je ke hře nutnou podmínkou, ale klávesnice pouze vhodným doplňkem. Nastavení jednotlivých ovládacích kláves či tlačítek myši nelze změnit, ale lze přenastavit citlivost ovládání kamery.

- Pohyb kamery – klávesy WASD, šipky, Shift+MMB[1]
- Otáčení kamery – MMB, Num8462
- Přiblížení kamery – kolečko myši, Num+-
- Vybírání jednotek – LMB[2]
- Použití schopnosti – RMB[3]
- Vybírání schopností – LMB, 1234
- Pauza – Esc

[1] MMB – Middle Mouse Button (prostřední tlačítko myši)
[2] LMB – Left Mouse Button (levé tlačítko myši)
[3] RMB – Right Mouse Button (pravé tlačítko myši)