FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Znalostní systém pro podporu výuky BI-ZNS
Student:	Bc. Jan Horáček
Vedoucí:	doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Cílem práce je navrhnout a implementovat framework pro znalostní systém pro podporu výuky předmětu BI-ZNS.

1) Seznamte se s problematikou znalostních (expertních) systémů a proveďte rešerši stávajících softwarových nástrojů, které umožňují práci se znalostními (expertními) systémy, zejména diagnostickými.
2) Na základě získaných teoretických znalostí navrhněte framework pro tvorbu znalostních systémů, který bude sloužit jako podpora pro výuku předmětu Znalostní systémy na FIT ČVUT (BI-ZNS).
3) Navržený framework implementujte v jazyce Python a důkladně zdokumentujte.
4) Funkčnost navrženého a implementovaného frameworku demonstrujte na vhodném problému po konzultaci s vedoucím práce.
5) Vyhodnoťte dosažené výsledky a navrhněte možná další vylepšení.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

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

V Praze dne 11. února 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Znalostní systém pro podporu výuky BI-ZNS

Bc. Jan Horáček

Katedra Softwarového inženýrství
Vedoucí práce: doc. RNDr. Ing. Marcel Jiřina Ph.D.

6. května 2019

Poděkování

Rád bych poděkoval svému vedoucímu práce doc. RNDR. Ing. Marcelu Jiřinovi, Ph.D. za odborný dohled a poskytnuté cenné informace z oblasti znalostních systémů. Rád bych také poděkoval Matějovi Shánělovi za pomoc při návrhu hry a základních částí frameworku. Také bych rád poděkoval Ing. Kláře Hájkové za ochotu integrace nového systému do výuky předmětu. V poslední řadě bych rád poděkoval své rodině a přátelům za neustálou podporu a pomoc po dobu vytváření práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 6. května 2019 .....

Odkaz na tuto práci

Horáček, Jan. *Znalostní systém pro podporu výuky BI-ZNS*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Cílem této práce bylo vytvoření nového frameworku pro výuku znalostních systémů v předmětu Znalostní systémy (BI-ZNS) na Fakultě informačních technologií ČVUT.

V první fázi byla provedena analýza problematiky znalostních systémů. Analýza se převážně zabývala definicí požadavků a omezení na cílovou doménu problému, který budou uživatelé mít za úkol řešit pomocí znalostního systému. Na základě analýzy vznikl základní model počítačové hry žánru věžová obrana, která bude sloužit jako problémová doména pro znalostní systémy.

Následně byl proveden důkladní návrh veškerých herních principů, které se budou vyskytovat ve hře. Na základě principů a požadavků na celý systém byl vytvořen návrh modulárního frameworku a jeho komponent.

Na základě analýzy a návrhu byl naprogramován Python framework, který obsahuje samostatně hratelnou počítačovou hru. Zároveň framework obsahuje pomocné moduly a základní kostru pro vytváření znalostních systémů.

V závěru se práce zabývá možností rozšíření frameworku o další funkcionalitu nebo případné vytvoření nové hry s využitím modulů znalostního systému. Pro umožnění snadného rozšíření a bezproblémové využití frameworku při výuce se poslední část práce zabývá testováním veškeré funkcionality frameworku.

Klíčová slova Znalostní systém, Věžová obrana, BI-ZNS, Python, PyQt5

Abstract

The objective of this thesis was to create a new framework for teaching knowledge-based systems in the subject Knowledge-based Systems (BI-ZNS) at the Faculty of Information Technology.

In the first phase was done an analysis of knowledge-based systems. The analysis was focused on defining the requirements and constraints of the target problem domain that users will use for creating the knowledge-based system. Based on the analysis was developed a basic concept of the tower defense game. The game will serve as a problem domain for knowledge-based systems.

Subsequently, a detailed design of all game principles that will occur in the game was made. Based on the principles and requirements for the entire system, a modular framework and its components were designed.

Based on the analysis and design, the framework in Python was developed. The framework contains a standalone playable computer game. Also, the framework contains auxiliary modules and a basic skeleton for building knowledge-based systems.

In the end, the thesis deals with the possibility of extending the framework with additional functionality. Also, there is the possibility to create a new game with re-using knowledge-based system modules. To enable an easy extension and trouble-free use of the framework in teaching, the last part of the theses is focused on testing of all framework functionality.

Keywords Knowledge-based system, Tower defense, BIE-ZNS, Python, PyQt5

Obsah

Seznam obrázků

Seznam ukázek kódu

Seznam tabulek

Úvod

Znalostní systémy patří mezi první pokusy o vytvoření umělé inteligence. První programy postavené na tomto principu vznikly již v 70. letech 20. století. I přesto, že v dnešní době existují již velice sofistikované systémy umělé inteligence, v některých oborech se znalostních systémů stále využívá. Hlavní výhodou znalostních systémů je převážně práce s nečíselnými daty a práce s neurčitostí. Díky těmto vlastnostem je zde stále mnoho oblastí, ve které jsou znalostní systémy nedocenitelné.

V rámci bakalářského oboru Znalostní inženýrství na Fakultě informačních technologií se studenti také učí, jak vytvářet znalostní systémy. Největší problém při vytváření znalostních systémů je analýza domény, kterou by měl systém řešit. Častokrát se zde stává, že studenti nevymyslí vhodnou doménu, což vede k použití naprosto nesmyslné domény.

Jeden ze zajímavých problémů, který lze řešit pomocí znalostních systémů, je ovládání počítačových her. Jedná se převážně o strategické hry, protože lze jednotlivé kroky snadno popsat pomocí příkazů. Jako příklad může sloužit například hra Age of Empires 2, která pro ovládání umělé inteligence používá právě znalostní systém. Pokud se připraví dostatečně zábavná počítačová hra, kterou navíc bude možné snadno plánovat, vznikne tak ideální problémová doména pro tvorbu znalostních systémů. Navíc v dnešní době jsou počítačové hry mezi studenty ve veliké oblibě, tudíž by framework postavený na hře mohl velice zvednout oblíbenost předmětu.

Kapitola 1

Úvod do problematiky

Tato kapitola se zabývá převážně úvodem do problematiky znalostních systémů. Nachází se zde stručný popis fungování znalostních systémů, jejich dělení a popis rozdílů oproti ostatním metodám umělé inteligence. Dále se zde také nachází zhodnocení již existujících řešení, které by bylo možné využít při vytváření práce. V poslední části kapitoly je stručně popsán žánr počítačové hry věžová obrana.

1.1 Cíl práce

Cílem této práce je seznámit se s problematikou znalostních systémů a pochopit princip fungování jeho částí. Následně na základě teoretických znalostí vytvořit desktopovou aplikaci, která bude sloužit jako framework pro výuku znalostních systémů v předmětu „Znalostní systémy“ na Fakultě Informačních technologií. Práce se zaměřuje na vytvoření frameworku pro studenty, který jim velice usnadní proces vývoje znalostního systému.

Studenti budou vytvářet plánovací znalostní systém, který má za úkol automaticky ovládat počítačovou hru. Počítačová hra bude žánru věžová obrana, kde student (znalostní systém) bude hrát roli obránce.

Součástí práce je také vytvoření grafického rozhraní, ve kterém bude možné hru zcela ovládat. Toto rozhraní pak bude sloužit studentům ke snadnému pochopení herních principů a také pro ladění jimi vytvořených systémů. Zároveň grafické rozhraní bude umět procházet historii bitvy, a tím odhalit problematické části.

1.2 Znalostní systémy

Pojem znalostní systémy (ZS) se často v literatuře zaměňuje za pojem expertní systém. Podle přesnější definice lze chápat expertní systém jako zvláštní tip znalostních systémů, který využívá pro svoji funkci expertní znalosti. V dnešní době

se však rozdíl mezi těmito pojmy stále více stírá. Práce považuje oba výrazy za ekvivalentní a nepřiřazuje žádnému z nich žádné odlišnosti. Informace obsažené v této části práce jsou čerpány převážně z práce Jiřího Dvořáka[1], z knihy Expert Systems od autorů Josepha C. Giarratana a Garyho D. Rileyho[2] a přednášek předmětu Znalostní systémy [3].

1.2.1 Definice

Profesor Edward Feigenbaum definoval Expertní systémy takto:

> „Experní systémy jsou počítačové programy, simulující rozhodovací činnost experta při řešení složitých úloh a využívající vhodně zakódovaných, explicitně vyjádřených speciálních znalostí, převzatých od experta, s cílem dosáhnout ve zvolené problémové oblasti kvality rozhodování na úrovni experta.“

Definice mluví o expertních systémech, nicméně tuto definici lze použít i pro znalostní systémy. Znalostní systémy patří do oboru umělé inteligence a mají několik charakteristických rysů, kterými se odlišují od ostatních algoritmů umělé inteligence:

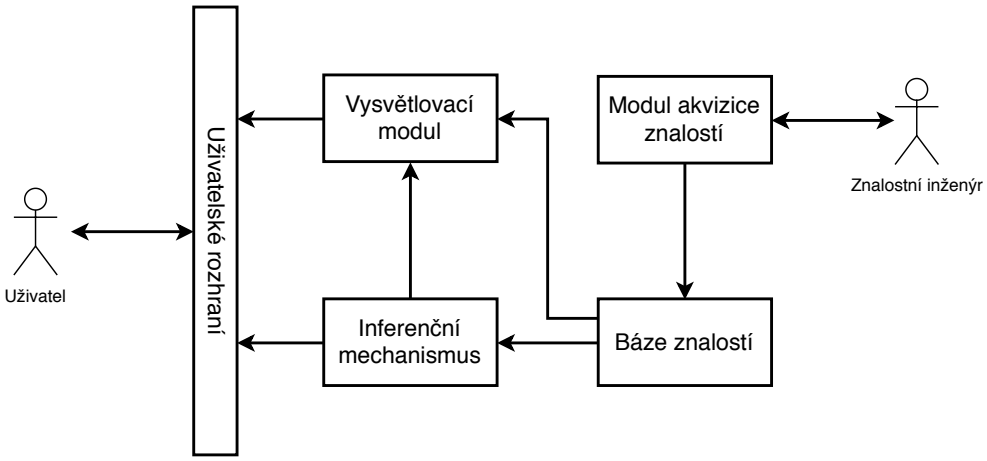
- Oddělují mechanismus zpracování znalostí a samotné znalosti.
- Dokáží rozhodovat o výsledku za přítomnosti neurčitosti (v datech nebo znalostech).
- Dokáží své rozhodnutí odůvodnit a vysvětlit zvolený postup důkazu.

1.2.2 Struktura znalostního systému

Základní struktura znalostního systému se skládá z 5 částí. Jejich vzájemná komunikace a vazby jsou znázorněny na obrázku 1.1. Struktura různých druhů a implementací ZS se může částečně lišit, nicméně je zde vždy zachován princip oddělení znalostí a jejich zpracování.

Báze znalostí

Báze znalostí obsahuje veškeré znalosti z cílové domény, pro kterou je znalostní systém určen. Veškeré informace obsažené v bázi znalostní jsou vytvořeny na základě znalostí experta, který definuje jednotlivá pravidla. Báze znalostí může obsahovat informace od velmi obecných a známých až po úzce odborné a specializované. Veškeré znalosti navíc nemusí být exaktně prokázány, ale může se také jednat o nejisté heuristiky či soukromé znalosti experta. Struktura a výsledná pravidla zcela závisí na expertovi, který bázi vytváří, a mají zásadní důsledek na funkcionalitu znalostního systému. Jediná limitace je zde ve vyjadřovací schopnosti jazyka, kterým se báze znalostí definuje.



Obrázek 1.1: Struktura znalostních systémů

Znalosti v bázi znalostí je možné reprezentovat mnoha způsoby. Mezi nejběžnější patří:

- matematická logika,
- pravidla,
- rozhodovací stromy,
- sémantické sítě,
- rámce a scénáře.

Inferenční mechanismus

Inferenční mechanismus má za úkol na základě báze znalostí a báze faktů (vysvětleno dále), rozhodnout o výsledku zadaného problému. Využívá obecné algoritmy, které jsou schopné řešit zadaný problém nezávisle na doméně problému. Nezávislost na doméně problému je velmi důležitá a umožňuje celkové oddělení báze znalostní od ostatních částí systému. Existuje několik různých metod inference, které mohou být použity a v některých případech je lze i kombinovat:

- **Dedukce** – logické usuzování, závěry plynou z předpokladů.
- **Indukce** – postup od specifických případů k obecným.
- **Abdukce** – usuzování na základě korektních závěrů a hledání příčiny.

- **Heuristiky** – usuzování na základě pravidel, které využívají odhady nebo zkušenosti.
- **Analogie** – Odvození závěru na základě jeho podobnosti s podobnými problémy.
- **Defaultní inference** – v případě absence specifických znalostí se usuzuje na základě obecných znalostí.
- **Nemonotónní** – metoda umožňuje upravovat či zneplatnit již odvozené závěry.
- **Intuice** – obtížně implementovatelný způsob rozhodování, který často využívají lidé. Velice podobného chování lze v informatice docílit pomocí neuronových sítí.

Další důležitou vlastností inferenčního mechanismu je zpracování neurčitosti. Práce s neurčitostí je specifickým znakem pro znalostní systémy, protože ostatní metody umělé inteligence mají se zpracování neurčitosti problém. Inferenční mechanismus musí zvládnout zpracovat neurčitost jednak z báze znalostní, ale také od uživatele. V případě znalostních systémů mohou být zdroji neurčitosti například vágní pojmy od uživatelů, nepřesnost či nekonzistence dat nebo také nejisté znalosti od experta. Neurčitost lze reprezentovat a následně zpracovávat pomocí mnoha postupů. Mezi nejznámější patří například:

- pravděpodobnostní přístupy (Bayesovský přístup),
- faktory jistoty,
- Dempster-Shaferova teorie,
- fuzzy neurčitost.

Inference pro svoji činnost využívá také **bázi faktů**. Do báze faktů se ukládají veškeré získané informace od uživatele a již známé či odvozené fakty. Báze faktů funguje jako jakási paměť inferenčního mechanismu. Díky ukládání závěrů splněných pravidel do báze faktů je možné následně odvozovat další pravidla, které bez uložených závěrů nebylo možné splnit. Báze faktů je úzce spjatá s inferenčním mechanismem, a proto není na obrázku 1.1 znázorněna jako samostatný modul.

Uživatelské rozhraní

Další důležitou částí znalostního systému je uživatelské rozhraní. Uživatelské rozhraní umožňuje komunikaci mezi ZS a uživatelem. Během procesu rozhodování se znalostní systém doptává uživatele na veškeré potřebné informace. Tato komunikace probíhá pomocí jednoduchých otázek, na které uživatel odpovídá nejčastěji ANO / NE nebo číselnou hodnotou v případě zpracování

neurčitosti. Uživatelské rozhraní se také používá ke zprostředkování informací z vysvětlovacího modulu.

Vysvětlovací modul

V případě, že znalostní systém komunikuje s uživatelem a pokládá mu otázky, je často žádoucí vědět, proč se na danou otázku systém ptá. Vysvětlovací modul má za úkol každé rozhodnutí znalostního systému odůvodnit. Základní funkcí modulu je zodpovězení dvou základních otázek:

- Proč se systém ptá na danou otázku?
- Jak systém dospěl k určitému faktu či závěru?

Pro odůvodnění faktu či závěru se jako výstup nejčastěji používá seznam faktů, které vedly k danému rozhodnutí. Vysvětlování proč se na danou otázku ptá pak závisí na konkrétním způsobu inference.

Modul akvizice znalostí

Hlavním cílem modulu akvizice znalostí je získání znalostí, ať již od experta či jiného externího systému. Tyto znalosti pak následně převede do báze znalostí, se kterou již pracuje znalostní systém. Modul může být pojat různým způsobem, buď se jedná o prostředí pro snadné zadávání znalostí od experta nebo o modul, který získává informace automaticky (například na základě umělé inteligence či databáze dat). Možná je také samozřejmě kombinace obou přístupů, kde expert upravuje a zdokonaluje informace získané automaticky. V práci se modul akvizice znalostí nepoužívá, protože pro framework není potřebný.

1.2.3 Typy znalostních systémů

Znalostní systémy je možné rozdělit na skupiny podle různých kritérii. První možností je klasifikace na základě obsahu báze znalostí. V takovém případě rozlišujeme ZN na:

- **Problémově orientované** – báze znalostní ZS obsahuje znalosti z určité domény
- **Prázdné** – báze znalostní ZS je prázdná

Dále je možné znalostní systémy rozdělit do kategorií na základě aplikace výsledného znalostního systému. Podle způsobu použití je pak vhodné zvolit správnou reprezentaci znalostí a inferenční mechanismus. Jako základní skupiny ZS lze považovat:

Diagnostické systémy mají za úkol rozhodnout, která z předem definovaných výsledků z konečné množinu nejlépe koresponduje s poskytnutými informacemi. Velice často je možné se setkat s touto kategorií v medicíně pro diagnózu chorob na základě symptomů a stavu pacienta. Nadstavbou nad diagnostickými systémy jsou opravné systémy, které navíc musí uživateli poskytnout informace, jak daný problém vyřešit.

Plánovací systémy na základě známého cíle a počátečního stavu se snaží nalézt posloupnost akcí, pomocí které lze dosáhnout cíle. Často se zde také klade podmínka na kvalitu nalezeného řešení. Diagnostické systémy se často využívají například v ekonomickém sektoru pro plánování složitých operací nebo například při hraní počítačových her.

Monitorovací systémy porovnávají referenční data s daty od uživatele nebo měřící techniky. Na základě porovnání hodnot vytváří komplexní zprávy o stavu celého systému a varují před případnými problémy. Monitorovací systémy mohou také pracovat neustále a přímo kontrolovat stav získaný od uživatele. Příkladem může být systém, který během psaní textu kontroluje pravopis.

Interpretační systémy slouží k vysvětlování určitého pozorovaného jevu pro uživatele. Systém může například analyzovat měřená data ze senzorů celého systému a následně tyto informace interpretovat uživateli v snadno pochopitelné formě. Do této kategorie se dají též zařadit například systémy pro překlad cizích jazyků.

Prediktivní systémy se snaží na základě historických dat předvídat vývoj v blízké budoucnosti. Velkou výhodou u prediktivních systémů je podpora neurčitosti, pomocí které lze lépe interpretovat a definovat některé jevy. Takové systémy je pak možné využít například pro předpověď počasí nebo vývoj ekonomiky, kde se dá očekávat určitá spojitost mezi některými jevy.

Školící systémy jsou speciální kategorie systémů, jejichž hlavním účelem je simulování konkrétních situací a problémů pro výukové účely. Systém celou simulací provádí, poskytuje dodatečné informace a informuje o důsledcích, které se staly na základě rozhodnutí studenta. Ukázkovým příkladem může být například systém `STEAMER` [4], který využívá americké námořnictvo pro výuku inženýrů lodních motorů.

1.3 Současný stav

Aktuální framework, který se nyní používá k výuce v předmětu „Znalostní systémy“, je z hlediska funkcionality nedostačující a těžko použitelný. Jeho funkcionalita je zaměřená na usnadnění vytváření pravidel pro znalostní systémy

a jejich následné zpracování. Program využívá již existující knihovnu pro práci s pravidly `eprove`. Jedná se o externí knihovnu, kterou nelze modifikovat ani nijak upravit pro potřeby systému. V některých ohledech je její vyjadřující schopnost nedostatečná, například pro práci s pokročilými pravidly a fuzzy logikou. Systém také definuje základní strukturu, která musí být uživatelem dodržena. Tato struktura je velice omezující a přináší výhodu pouze základního konzolového rozhraní. Studenti se ve většině případů rozhodli pro usnadnění práce tento framework nepoužít a raději si napsat celý znalostní systém od začátku.

Kvůli již zmiňovanému problému s knihovnou `eprove` není možné framework rozšířit bez zásadního zásahu do samotné logiky. Z tohoto důvodu a také z důvodu špatně navržené struktury celého programu nemá smysl projekt dále rozšiřovat a řešit případné problémy. Řešením tohoto problému je vyvinout nový framework nebo případně použít již stávající a upravit ho k potřebám předmětu.

1.4 Existující řešení

Jedním z možných řešení současného stavu frameworku je využít již existující prostředí pro vývoj znalostních systémů. Mezi nejznámější patří například prostředí CLIPS neboli *C Language Integrated Production System*[5]. CLIPS má vlastní vestavěný inferenční mechanismus, který rozhoduje o zpracování vstupních pravidel a vstupu od uživatele. Z tohoto důvodu je nepoužitelný pro výuku znalostních systémů, jehož hlavní náplní je naprogramování vlastního inferenčního systému. Ze stejného důvodu můžeme vyloučit i ostatní standardní vývojová prostředí pro znalostní systémy.

Dalším požadavkem na nový framework je systém ovládání počítačové hry pomocí znalostního systému. Mezi ohromným množstvím počítačových her lze nalézt hry, které pro své ovládání či alespoň některé části ovládání využívají znalostní sytém. Jako dobrý příklad lze použít *Age of Empires 2* (AoE)[6]. Znalostní systémy se ve hře AoE používají pro definici chování umělé inteligence nepřátel. Na základě velkého množství pravidel je zde možné nadefinovat poměrně přesné chování jednotek. Hráči si ve hře mohou zkusit vytvořit vlastní umělou inteligenci. Bohužel se opět jedná pouze o systém, který již má vestavěný inferenční mechanismus. Nicméně AoE lze použít jako zdroj informací a inspirace ohledně herních principů, které lze řadit pomocí znalostních systémů.

Poslední možností řešení problému je odstranění frameworku z předmětu a nechat studenty vytvářet celý znalostní systém. V takovém případě je zde ale problém, že student musí věnovat velké úsilí částem, které jsou velice vzdáleny samotnému návrhu a implementaci znalostního systému. Student by musel naprogramovat vlastní ovládací rozhraní, připravit vlastní jazyk pravidel, implementovat jeho analýzu a mnoho dalšího. Navíc každý student by si musel

vymyslet a analyzovat vlastní téma, pro které by vytvářel znalostní systém, což pro mnoho studentů je složité a velice časově náročné. Z těchto důvodů je existence dobrého a snadno použitelného frameworku velmi žádaná a velice usnadní výuku předmětu BI-ZNS.

1.5 Herní žánr věžová obrana

S herním žánrem věžová obrana (tower defense) je možné se setkat nejenom na stolních počítačích, ale také na mobilních zařízeních a ve webových prohlížečích. Jedná se převážné o hry, které využívají většinou jednodušší 2D grafiku často s fantasy nebo sci-fi žánrem. Hry většinou cílí na jednoduchou hratelnost, která zabaví na dlouhé chvíle na cestách. V historii vzniklo i několik propracovaných titulů tohoto žánru, ale jedná se o nepříliš úspěšné výjimky. Tento žánr se také velice často implementuje do jiných her v podobě bonusových map a módů. Převážně pak ve hrách, které umožňují vytváření nových map od komunity hráčů, kteří často vytváří právě věžové obrany z důvodu jednoduchého základního herního principu. Velké množství her tohoto žánru například vzniklo ve hře Warcraft 3[7], díky propracovanému editoru map.

Základní princip tohoto žánru je velmi jednoduchý. Hráč je v pozici obránce a jeho úkolem je ubránit důležitý bod (například krále nebo vesnici) před nájezdem hordy nepřátel. K obraně mu slouží věže, které během plánování svých tahů staví na strategická místa na mapě. U velkého množství her se pak setkáváme s principem, kdy obránce má předem definovanou množinu míst, kde může svoji obranu vystavět. V takovém případě i útočník má pevně definovanou trasu útoků.

Průběh bitvy bývá obvykle rozdělen na kola, kde jedno kolo se skládá ze stavění obrany na straně obránce a vyhodnocení útoku. Některé věžové obrany využívají striktní systém kol, kdy během útoku nemůže obránce provádět žádné akce. U ostatních se pak kolo útočníka a obránce slévá, a hráč může reagovat na tahy útočníka okamžitě.

Aby hráč nebyl omezen jenom volným prostorem pro stavbu mapy, stavění bývá také omezeno surovinami, které má hráč k dispozici. Jednotlivé věže stojí různé množství surovin podle jejich schopností a účinku. Suroviny pro výstavbu se získávají buď pravidelným příjmem každé kolo, nebo jako odměna za poražení nepřátelské armády.

Důležitým prvkem tohoto žánru je strategie plánování obrany. Velmi často se zde využívá tvaru a členitosti mapy, která umožňuje využít strategických míst pro vytvoření komplexní obrany. Druhý důležitým aspektem bývá různorodost útočících jednotek a obránců. Kombinují se zde různé vlastnosti a odolnosti jednotek, aby obránce musel svoji obranu mít připravenou na různé druhy útoku (vzdušný útok, silné brnění, magická ochrana atd.). Pokud hra nenabízí dostatečné množství možností a kombinací, velice rychle se začíná opakovat a stává se nudnou.

Mezi nejznámější zástupce tohoto žánru na mobilních zařízení lze zařadit titul `Plants vs. Zombies`[8]. Již trochu upravený, ale stále postavený na základu tower defense, se tento žánr dostal i do e-sportu díky hře `Clash Royale` [9]. Mezi nejúspěšnější 3D kooperativní hry v tomto žánru pak patří titul `Dungeon Defenders`[10].

1.5.1 Plants vs. Zombies

Hra `Plants vs. Zombies` patří mezi klasické zástupce herního žánru věžové obrany, a je tudíž ideální příklad pro ukázku základních herních mechanismů. Hráč se zde nachází v pozici obránce, který brání svůj domov před útokem zombií. Pro vybudování své obrany využívá rostliny, které svými vlastnostmi přispívají do celkové obranné strategie.

Na začátku každé bitvy se hráč rozhodne, které rostliny bude mít k dispozici pro svoji obranu. Vybírat si může z velkého množství rostlin, přičemž každá z nich má různé vlastnosti a svoji speciální schopnost. Je zapotřebí zvolit vhodnou kombinaci rostlin, jejichž schopnosti se doplňují. Na obrázku 1.3 je vidět, že seznam všech rostlin je opravdu rozsáhlý.

Na obrázku 1.2 je vidět probíhající bitva. Vlevo nahoře se nachází číslo udávající počet sluncí, které slouží ve hře jako měna pro stavění obrany. Postavení každé rostliny stojí určitý obnos sluncí, které je nutné zaplatit. Slunce hráč získává během probíhající bitvy, nebo pomocí svých rostlin. Hráč může své rostliny umístit na libovolné čtvercové místo na mapě. Na jedné pozici může být pouze jedna rostlina. Rostliny je nutné rozestavit takticky, aby odolné rostliny byly vpředu bitvy a slabší je naopak podporovaly zpovzdálí.

Po krátké době, kterou hráč má na přípravu obrany, začne útočník posílat své jednotky. Jednotky se smějí pohybovat pouze v jednom směru. Útočník pouze rozhodne, do které řady svoji jednotku nasadí. Za útočníka v případě `Plants vs. Zombies` hraje umělá inteligence. Jednotky, které útočník posílá, jsou různě silné a některé z nich mají odolnost proti různému druhu útoků. Úkolem útočníka je dostat se přes všechny rostliny až k bráněnému domu. Pokud se libovolná zombie dostane až na konec mapy, obránce prohrává. Útočník má pro každou bitvu omezený počet jednotek. Pokud útočníkovi již žádná jednotka nezbývá, bitvu vyhrává obránce.

Celá bitva v případě `Plants vs. Zombies` není rozdělena na jednotlivá kola, ale bitva probíhá kontinuálně. Obránce může kdykoliv stavět své rostliny a nepřítel neustále vysílá nové zombie do útoku. Kontinuální průběh jednak dovoluje hráči lépe reagovat na blížící se jednotky, naopak ale musí počítat s rychle měnící se situací na bitevním poli.

Úkolem hráče je postupně porážet zombie na jednotlivých bojištích, až nakonec porazí všechny nepřátele. Hra také nabízí velké množství rozšiřujících principů, díky kterým se hra nestane stereotypní. Mezi ně například patří odemykání nových druhů rostlin nebo vylepšování stávajících. Navíc s postupu-

Obrázek 1.2: Průběh bitvy Plants vs. Zombies

Obrázek 1.3: Ukázka výběru druhů rostlin

jícími bitvami se také mění nepřátelé, což nutí hráče vytvářet nové kombinace rostlin pro svoji obranu.

Kapitola 2

Analýza

Tato kapitola práce se věnuje analýze znalostních systémů a požadavků na cílový framework. V první části se kapitola věnuje detailnímu popisu požadavků na framework. Nalézá se zde detailní popis všech funkčních a nefunkčních požadavků. V druhé části se kapitola zaměřuje na definici jednotlivých scénářů užití, které detailně popisují problematické části fungování znalostních systémů.

2.1 Požadavky

Důležitým krokem analýzy každého programu je sběr funkčních a nefunkčních požadavků. Na základě požadavků lze navrhnout základní kostru celého programu. Přesná definice veškerých požadavků je velice důležitá, protože na jejich základě se určuje výsledný rozsah práce.

2.1.1 Funkční požadavky

Funkční požadavky vymezují veškerou funkcionalitu, kterou by aplikace měla umožňovat. Požadavky jsou seřazeny podle logických souvislostí.

Počítačová hra žánru věžová obrana

Základní částí celého frameworku bude počítačová hra žánru věžová obrana. Ovládání hry bude cílová doména, kterou bude muset uživatel řešit pomocí znalostního systému. Hra bude splňovat základní principy žánru a bude možné její snadné rozšíření o další herní prvky (nové rozhraní, nový druh jednotek, atd.).

Prostředí pro vytváření znalostního systému

Framework bude umožňovat snadné rozšíření zpracování znalostního systému, pomocí kterého bude možné hru ovládat. Znalostní systém bude primárně určen jako hlavní ovládací prvek. Celé rozšíření funkcionality bude možné

provést v oddělené části frameworku. Framework bude umožňovat snadnou kontrolu pokusu o podvádění, pří kterých student upraví nepovolené části implementace.

Samostatně hratelná hra

Hra, na které bude celý framework postaven, bude samostatně hratelná bez nutné implementace znalostního systému. Bude možné veškeré operace, které je možné udělat pomocí znalostního systému, realizovat i prostřednictvím pokynů z ovládacího rozhraní. Ovládací rozhraní bude sloužit převážně k pochopení všech herních principů a ladění znalostního systému. Jednotlivá kola ZS musí být možné ovládat z grafického rozhraní.

Poskytování informací ze hry

Framework bude v procesu zpracování znalostního systému nahrazovat uživatele. Na místo uživatele bude framework poskytovat veškeré potřebné informace, které bude možné použít k vyhodnocení závěrů. Znalostní systém bude moci díky této funkcionalitě fungovat zcela nezávisle na uživateli.

Zpracování pravidel báze znalostí

Framework bude za uživatele řešit práci s pravidly, které definují bázi znalostí. Zdrojový soubor s pravidly framework analyzuje, a pokud nenalezne žádné problémy, převede pravidla do stromové struktury, kterou poskytne uživateli. Stromová struktura pravidel bude sloužit převážně ke snadnému vyhodnocení pravdivosti podmínky pravidla. V případě problému s pravidly nahlásí uživateli chybu a její příčinu.

Práce s neurčitostí

Framework bude podporovat práci s neurčitostí jak ze strany báze znalostní, tak ze strany uživatele (informace ze hry). Hra musí poskytovat některé informace s mírou neurčitosti, aby bylo možné tyto informace využít ve znalostním systému.

Vyhodnocování kvality znalostního systému

Framework bude umožňovat testovat kvalitu vytvořeného znalostního systému. Kvalita znalostního systému bude reprezentována na základě počtu kol, které se zvládne ZS ubránit útokům nepřítele. Kvalita ZS musí být snadno porovnatelná s jiným řešením.

Konzolové rozhraní

Framework bude poskytovat konzolové rozhraní, pomocí kterého bude možné automaticky vyhodnocovat kvalitu znalostního systému. Jako vstup přijme konfigurační soubor, který definuje mapu a umělou inteligenci a na výstupu zobrazí míru kvality znalostního systému. Konzolové rozhraní bude sloužit převážně pro hromadné testování kvality a porovnávání.

Generování a konfigurace mapy

Framework bude umožňovat generovat náhodné mapy pro bitvy. Vygenerovaná mapa bude složená z předem definovaných terénů. Chování generátoru a výslednou mapu bude možné ovlivnit pomocí konfigurace. Generátor bude možné snadno rozšířit o další druhy terénu.

Pro případy testování a vyhodnocení kvality ZS bude framework umožňovat definici výsledné mapy. Mapu bude možné nadefinovat pomocí dvourozměrného pole jednotlivých pozic. Generování mapy bude také možné ovlivnit pomocí počátečního seedu náhodného generátoru.

Umělá inteligence útočníka

Součástí frameworku bude také implementace základní umělé inteligence útočníka. Chování umělé inteligence musí umožňovat deterministické chování za totožného průběhu hry z důvodu porovnatelnosti výsledků. Modul pro umělou inteligenci musí být snadno rozšířitelný pro budoucí vývoj.

Systém procházení historie akcí

Framework bude ukládat veškerou historii akcí všech jednotek a následně bude možné tuto historii procházet. V grafickém rozhraní bude možné zobrazit stav mapy a jednotek v libovolném časovém okamžiku.

Export historie bitvy

Veškerou historii bitvy bude možné z frameworku exportovat. Bude možné využít export do formátu HTML, který bude vhodný pro případné procházení uživatelem. Také bude možné exportovat historii do formátu XML, ze které bude možné celou bitvu zrekonstruovat.

2.1.2 Nefunkční požadavky

Nefunkční požadavky vymezují převážně nároky na hardware a softwarové řešení. Jedná se například o dostupnost aplikace, výkonost aplikace nebo podporované jazyky prostředí.

Podpora operačních systémů Windows a Linux

Framework bude možné nainstalovat a spustit na operačních systémech Windows a Linux. V případě Windows budou podporovány všechny verze novější než Windows 7. V případě Linuxových distribucí, bude podporována velká část nejpoužívanějších distribucí s doporučenými aktualizacemi. Z linuxových distribucí musí být podporovány: Ubuntu, OpenSUSE a Debian. Framework bude mít definované potřebné závislosti pro svůj běh.

Rychlost odezvy jednotlivých operací

V případě vyhodnocování velkého množství herních kol bude systém dostatečně rychle jednotlivá kola vyhodnocovat a v případě grafického prostředí také zobrazovat. Rychlost odezvy musí být natolik rychlá, aby práce s prostředím byla pro uživatele pohodlná. V případě standardní mapy s výškou a šířkou mapy 11 políček musí být jedno kolo vyhodnoceno pod jednu sekundu (na procesoru s taktem alespoň 2.4 GHz). V případě velkých map a velkého množství jednotek, může být pomalejší, protože tyto případy se nebudou běžně testovat.

Modularita frameworku

Celý framework bude logicky rozdělen do více modulů, které bude možné snadno nahradit nebo znovu použít pro jiné využití. Podstatné je oddělení implementace znalostního systému a samotné hry. V rámci samotné hry je také důležité oddělení ovládací části frameworku od logiky, aby bylo možné tyto části snadno nahradit.

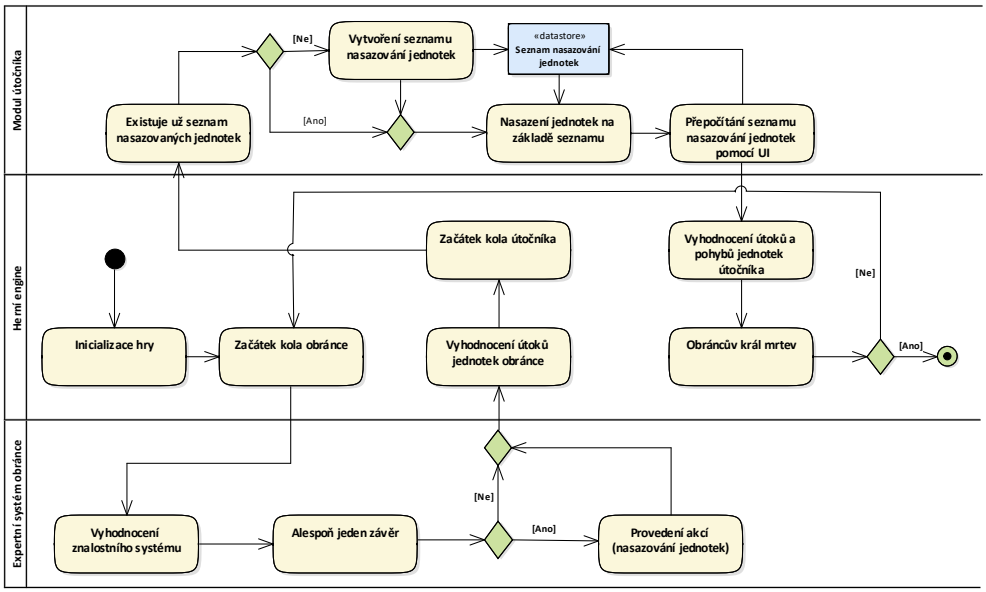
2.2 Business procesy

Business proces (někdy nazývaný podnikový nebo obchodní proces) má za úkol jednoznačně a srozumitelně popsat proces, který zákazník chce mít implementovaný ve svém frameworku. Business procesy se dají definovat textem, kde se celý proces slovně popíše. V případě složitého procesu s velkým množství větvením je mnohem přehlednější popsat proces pomocí diagramu aktivit. Diagram aktivit také rozděluje jednotlivé činnosti podle toho, kdo je za danou činnost zodpovědný (uživatel nebo systém).

Business procesy je dobré definovat hlavně pro náročnější části domény, které mohou být při návrhu či implementaci problematické. V práci je popsaný proces průběhu kola a detailní pohled na funkcionalitu znalostního sytému. Ostatní business procesy jsou k nalezení v příloze práce.

2.2.1 Průběh herního kola

Diagram na obrázku 2.1 popisuje průběh jednoho herního kola ve věžové obraně, která má striktní posloupnost kol a využívá znalostní systém. Diagram



Obrázek 2.1: Průběh herního kola

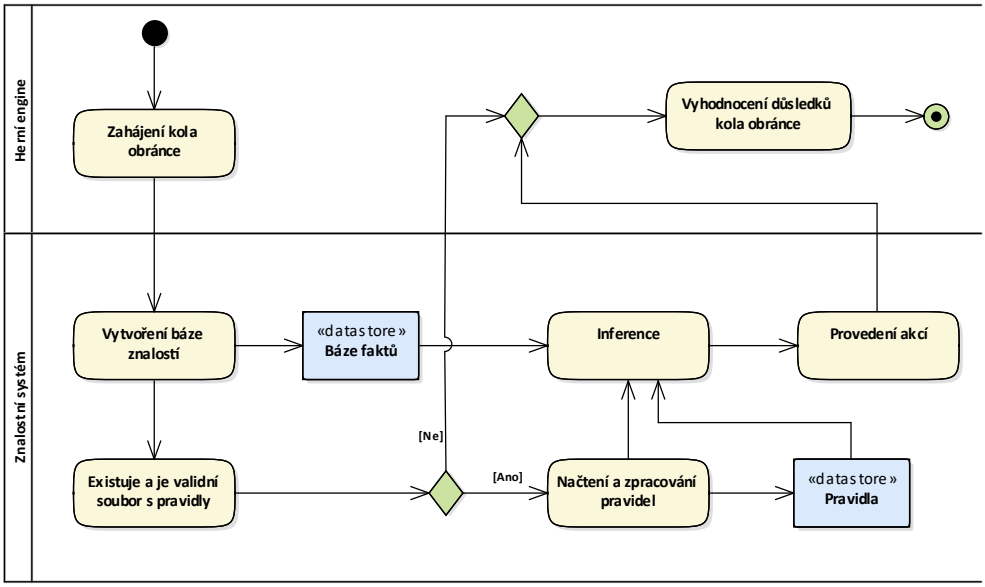
je rozdělen na tři části podle toho, který modul je za dané procesy zodpovědný:

- **Modul útočníka** – umělá inteligence řídící útočníka
- **Herní engine** – hlavní logika hry
- **Modul obránce** – znalostní systém řídící obránce

Po počáteční inicializaci hry (připravení mapy, nastavení příjmů hráčů, atd.) začíná první kolo. Každé kolo vždy začíná obránce. Kolo obránce je celé řízeno pomocí znalostního systému, který na základě poskytnutých informací rozhodne o případných akcích. Po případném provedení akcí herní engine (hlavní logika hry) vyhodnotí důsledky obráncova kola. Následuje část kola útočníka, která je velice podobná. Opět na základě umělé inteligence a dostupných informací nasadí nové jednotky a provede potřebné akce. Kolo zakončí herní engine, který přepočítá veškeré důsledky celého kola. Následně zkontroluje, že obráncův král je stále naživu (podmínka, při jejímž naplnění obránce prohrává), a pokud ano, následuje další kolo. V opačném případě hra končí. Přesnější průběh herního kola je závislý na konkrétních mechanismech hry jako systém pohybu jednotek a je detailně popsán v následují kapitole návrh.

2.2.2 Průběh vyhodnocení pomocí ZS

Diagram na obrázku 2.2 popisuje detailnější pohled na kolo obránce, konkrétně průběh vyhodnocení, ke kterému dospěje znalostní systém. Zpracování začíná



Obrázek 2.2: Průběh vyhodnocení pomocí ZS

na začátku kola obránce. V první řadě znalostní systém zpracuje informace z rozhraní hry čímž doplní informace v bázi faktů. Znalostní systém už zde nijak nekomunikuje s uživatelem, veškeré dostupné informace získá pouze zmíněným způsobem. Následně systém vyhodnotí, zda má dostupný a validní seznam pravidel, se kterými může pracovat. Pokud jsou pravidla syntakticky nevalidní nebo soubor s pravidly neobsahuje žádná pravidla, kolo obránce končí. V opačném případě se pravidla načtou a převedou do formátu stromové struktury. Následuje hlavní část celého ZS, totiž inferenční mechanismus. Inferenční mechanismus na základě báze faktů a báze znalostí vyhodnotí závěry. Na základě vyhodnocených závěrů se následně provedou veškeré akce. Tímto kolo obránce končí a herní engine vyhodnotí důsledky kola obránce.

Kapitola 3

Návrh

Tato kapitola se věnuje návrhu celého frameworku. Velká část se zabývá návrhem samotné hry a popisu návrhu herních principů. Hra využívá různých netradičních mechanismů, které přizpůsobují celý systém pro ovládání znalostními systémy. Následně je zde popsána modularita celého systému a popis jednotlivých částí. V poslední části jsou zde detailněji rozpracované složitější části návrhu pomocí modelů komunikace a modelu nasazení.

3.1 Návrh hry a herních principů

Herní žánr věžová obrana velice úzce definuje velké množství herních mechanismů. Nicméně je zde stále prostor pro definici některých vlastní a originálních principů. Většina je navržena tak, aby usnadnila ovládání hry pomocí znalostního systému. Po několika pracovních názvech je hra pojmenována Orodael Turrim, což má svůj původ v latinském jazyce a znamená to horské věže.

3.1.1 Základní princip hry

Orodael Turrim se z velké části drží standardních mechanismů žánru věžové obrany. Základním principem je probíhající bitva mezi obráncem a útočníkem. Obránce má za úkol na začátku bitvy umístit někam na mapu svého krále, kterého musí bránit co nejdéle dokáže. Obránce má krále pouze jednoho a po jeho smrti hra končí. Úkolem útočníka je co nejrychleji krále zabít.

Celá bitva je rozdělena na jednotlivá kola. V každém kole obránce i útočník můžou nasadit libovolný počet jednotek, na které mají peníze. Na začátku hry útočník a obránce dostanou určitý obnos, který můžou využít. Další peníze získávají na začátku každého kola. V každém kole nejprve nasadí libovolný počet jednotek obránce a následně nasazuje útočník.

Každé kolo kromě nasazování jednotek se také provádí akce jednotek. Mezi akce jednotek patří pohyb po mapě a útok na nepřátelskou jednotku. Každá jednotka se může nejprve posunout po mapě a následně zaútočit na jednotku

v dosahu, čímž vyvolá konflikt. Výsledek konfliktu záleží na atributech útočící a bránící jednotky. Každá jednotka má počet životů, který když klesne na nulu, jednotka umírá. Při konfliktu ztrácí životy pouze bránící se jednotka. Výsledek konfliktu mezi dvěma jednotkami vypočítá herní engine, uživatel konflikt již nijak neovlivňuje.

Jednotky se pohybují po mapě, která má rozličný terén a hraje důležitou roli. Terén může přidávat jednotkám bonusy nebo je naopak postihovat. Terény je možné nadefinovat libovolně, například ve hře může být sopečná krajina, která jednotku každé kolo zraňuje.

Aby obránce získal informace o příchozích jednotkách, vysílá každé kolo zvědy. Při každé výpravě má zvěd náhodnou úspěšnost, ze které se odvíjí, jak přesné informace o příchozích jednotkách dokázal zjistit. Díky zvědovi má obránce před začátkem každého kola svého nasazování informace o přibližných pozicích, kde se v příštích kolech objeví jednotky nepřítele.

3.1.2 Návrh průběhu bitvy

První důležitým herním principem je časové rozdělení jednotlivých kol. Orodael Turrim během bitvy nepoužívá spojitý čas, ale hra je rozdělena na jednotlivé kola. Diskrétní průběh hry je zvolen převážně z důvodu řízení pomocí znalostního systému. Toto rozhodnutí velice zjednodušuje potřebný ZS a také snižuje nároky na jeho rychlost řešení problémů. Zároveň diskrétní čas umožňuje mnohem snazší práci s historií celé bitvy.

3.1.3 Herní mapa

Dalším důležitým rozhodnutím při návrhu této hry je zvolení tvaru a fungování herní mapy. Hlavní dvě varianty pohybu ve 2D, které se v tomto žánru využívají jsou volný a jedno-pozicový pohyb.

Volný pohyb umožňuje jednotkám se po mapě pohybovat libovolným směrem a na libovolnou pozici. Volný pohyb se simuluje pomocí jemné sítě, utvořené z n-úhelníků (nejčastěji se využívají trojúhelníky nebo šestiúhelníky). Přes tuto síť je následně vykreslena mapa, která není nijak dělena na jednotlivá políčka. Pozice jednotek a věží v takovém případě není definován jednou pozicí, ale seznamem políček, které daný objekt zabírá. Tento princip přidává do hry velké množství další principů, jako jsou různé velikosti věží nebo nemožnost stavět na některých typech terénu.

Jedno-pozicový pohyb definuje pravidlo, že každý objekt se nachází na právě jedné pozici na mapě. Pozice v takovémto případu může být například jeden čtverec na čtverečkové síti. Jednotlivé pozice jsou na mapě jasně rozlišitelné a každá pozice reprezentuje jeden druh terénu. Pozice objektů v takovém případě je velice jednoduchá a lze ji snadno

reprezentovat. V případě diskrétního průběhu hry je tento druh pohybu nejlogičtější variantou.

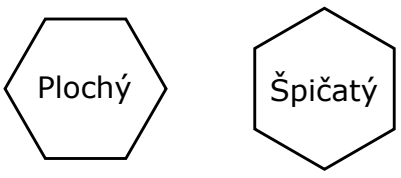
Orodael Turrim využívá jedno-pozicový pohyb převážně kvůli usnadnění plánování. Díky tomu znalostní systém nemusí řešit velké množství kombinací, které vznikají při použití volného pohybu. Zároveň jodno-pozicový systém se více hodí při použití diskrétního času, protože lze mnohem snadněji definovat pohyb jednotek.

Dále je zapotřebí rozhodnout tvar jednotlivých políček. Aby se částečně vykompenzovala příliš velká jednoduchost jedno-pozicového pohybu, Orodael Turrim používá mapu složenou ze šestiúhelníků. Šestiúhelníky umožňují daleko komplexnější pohyb po mapě a zároveň lze mnohem lépe využívat okolního terénu. Celkové plánování obrany tak umožňuje daleko více možností než v případě mapy složené ze čtverců.

Nicméně šestiúhelníková mapa sebou přináší některé komplikace, které obyčejná čtvercová mapa nemá. Příkladem může být poziční systém a jednotlivé algoritmy pro výpočet vzdálenosti či dohledu. Velkou inspirací a pomocí při řešení problému s šestiúhelníkovou mapu je stránka Amita Patela[11], na které lze nalézt podrobně rozebranou problematiku šestiúhelníkových map včetně různých algoritmů a ukázek.

Souřadnicový systém

První problém, který nastává při práci se šestiúhelníky, je souřadnicový systém. Při práci se šestiúhelníky už nelze snadno využít dvě souřadné osy, které by určovaly jednotlivé pozice. Pro určování pozice existuje více než jeden přístup, přičemž každý z nich má nějaké výhody i nevýhody. Protože jedním z hlavních požadavků na framework je snadná práce ze strany uživatele a zároveň rychlá odezva a počítání výsledků, jsou v práci využity celkem tři souřadné systémy. Uživatel si může vybrat, se kterým souřadným systémem se mu bude nejlépe pracovat a bude pro něj nejvíce srozumitelný.



Obrázek 3.1: Ukázka natočení šestiúhelníků

První důležité rozhodnutí, které je potřeba vyřešit před implementací souřadného systému, je tvar mapy a otočení jednotlivých šestiúhelníků. I když je samozřejmě možné natočit šestiúhelník libovolně, nejčastěji se používají dvě varianty znázorněné na obrázku 3.1:

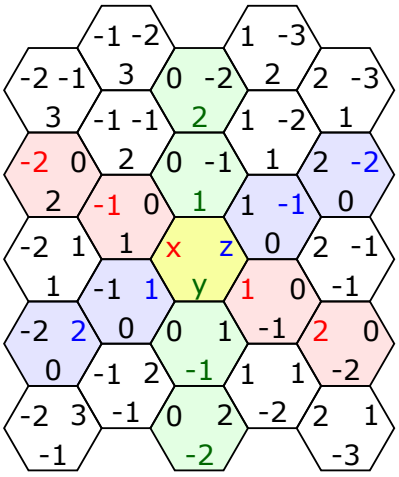
- **plochý** – šestiúhelníky jsou natočeny stranou směrem nahoru
- **špičatý** – šestiúhelníky jsou natočeny vrcholem směrem nahoru

Zvolený směr natočení má následně velký vliv na jednotlivé algoritmy počítání se souřadnicemi, nicméně složitost algoritmů se nijak nemění. Orodael Turrim využívá plochou variantu, hlavně z důvodu použité grafiky podkladu. Co se týče tvaru mapy, Orodael Turrim využívá obdélníkovou mapu, která má lichý počet řádků i sloupců a první šestiúhelník je posunut směrem dolů. Obdélníkový tvar je zvolen z důvodu co nejlepšího využití prostoru v grafickém prostředí. Díky lichému počtu polí v rozměrech mapy lze snadno nalézt střed mapy, což velice zjednodušuje některé algoritmy pohybu. Posunutí má opět význam pouze v případě některých algoritmů.

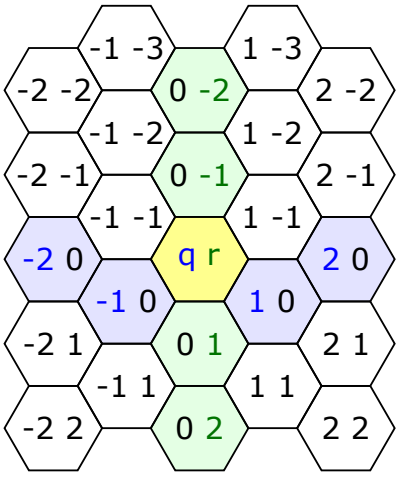
První z využívaných systémů je **kubický** souřadný systém, který pro definování pozice využívá tři souřadnice (x, y, z). Tyto souřadnice určují odchylku ve všech šesti směrech (díky kladným a záporným číslům) od počátku systému (pozice [0 0 0]). Pro stanovení jasného číslování se zde používá podmínka, že součet všech souřadnic musí vždy být 0. Jednotlivé osy je možné libovolně natočit. Na obrázku 3.2 lze vidět zvolený směr os a ukázka souřadného systému. Tento systém je velice optimalizovaný pro výpočet veškerých operací na mapě jako jsou dohled jednotky, vzdálenost dvou pozic a další. Z tohoto důvodu se veškeré použité souřadnice vždy převedou do kubického souřadného systému, který se pak následně používá pro vnitřní reprezentaci. Bohužel ale pro uživatele může být systém tří os velice matoucí a nepřirozený.

Lépe čitelný pro uživatele může být systém **posuvný** (offsetový). Tento systém využívá pro reprezentaci dvě souřadnice (q, r), které se snaží pozice připodobnit klasické čtvercové mapě. Souřadnice definují posunutí v řádcích a posunutí ve sloupcích. Při použití ploché varianty šestiúhelníků, je určení sloupce pozice jednoduché, protože se šestiúhelníky nacházejí přímo pod sebou. V případě řádků je již situace složitější, protože zde máme více varianta výběru šestiúhelníků, které tvoří řádek. Index řádku se určuje pomocí čísla pozice při průchodu sloupečku. Celý systém je navíc posunut tak, aby souřadnice [0 0] byla ve středu mapy a lépe korespondovala s kubickým souřadným systémem. Na obrázku 3.3 lze vidět ukázku souřadného systému okolo počátku.

Poslední z implementovaných systémů je **axiální**. Jedná se o upravenou verzi kubického systému, při kterém se využívají pouze dvě souřadnice. V případě kubického systému je jedna ze souřadnic redundantní díky definované podmínce o nulovém součtu souřadnic. Díky této podmínce je možné vždy dopočítat třetí souřadnici na základě dvou ostatních. Tento souřadný systém se používá převážně z důvodu možnosti uložení pozic do dvourozměrného pole. Axiální souřadný systém je vidět na obrázku 3.2 pokud se ze souřadnic odebere složka z.



Obrázek 3.2: Kubický souřadný systém



Obrázek 3.3: Posuvný souřadný systém

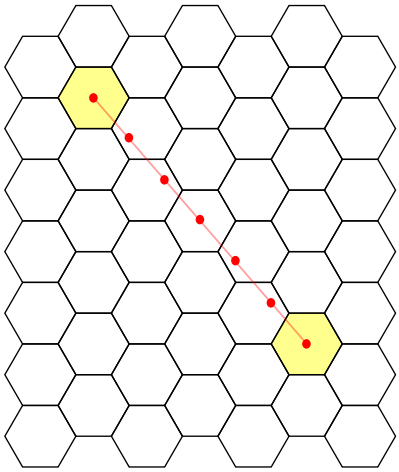
Pohyb po mapě a viditelnost

Co se týče pohybu po šestiúhelníkové mapě, nejedná se o žádný složitý systém. Jednotka se může pokaždé pohnout na jedno ze svých šesti sousedních polí, žádný jiný pohyb není dovolen. V případě kubického souřadného systému je zjištění sousedních pozic velice jednoduché pomocí přičtení čísel k počáteční pozici:

- +0 +1 -1 – horní soused
- +1 +0 -1 – pravý horní soused
- +1 -1 +0 – pravý dolní soused
- +0 -1 +1 – dolní soused
- -1 +0 +1 – levý dolní soused
- -1 +1 +0 – levý horní soused

Veškerý pohyb všech jednotek je počítán pouze na základě sousedů. Stejný princip se také využívá pro zjištěný možných cílů pro útok jednotky. Díky šestiúhelníkovým polím je zapotřebí dobře plánovat cestu pochodu, protože je zde mnohem snazší obejít těžko schůdný terén.

V případě počítání viditelnosti jednotek na mapě je již situace složitější. Aby bylo docíleno co nejvíce realistického dohledu jednotky, je nutné správně vybrat políčka, která se mají při výpočtu zohlednit. Při zjišťování, zda je cílové pole viditelné, se používá princip lineární interpolace.



Obrázek 3.4: Ukázka interpolace při výpočtu viditelnosti

Při lineární interpolaci se nejprve spočítá vzdálenost mezi počátečním bodem A a cílovým bodem B. V případě kubického souřadného systému se vzdálenost vypočítá jako:

$$vzdálenost = (|A.x - B.x| + |A.y - B.y| + |A.z - B.z|)/2$$

Následně se pomocí interpolace spočítají rovnoměrně rozložené body, které leží na úsečce spojující bod A a B

$$C.x = A.x + (B.x - A.x) * \frac{1}{vzdálenost} * i$$

$$C.y = A.y + (B.y - A.y) * \frac{1}{vzdálenost} * i$$

$$C.z = A.z + (B.z - A.z) * \frac{1}{vzdálenost} * i$$

Kde C je hledaný bod a i jde od 0 do $vzdálenost + 1$. Na základě vypočítaných bodů je nutné nalézt šestiúhelníky, na kterých se dané body nacházejí. Souřadnice šestiúhelníků je možné zjistit pomocí speciálního způsobu zaokrouhlení souřadnic, které je popsané na stránce Amita Patela[11]. Tyto šestiúhelníky se budou podílet na výpočtu viditelnosti. Na obrázku 3.4 je vidět ukázka interpolace. Orodael Turrim z této posloupnosti navíc ještě odebere první a poslední šestiúhelník na úsečce, aby bylo vždy zajištěno, že jednotka vidí alespoň své sousední políčka a políčko na kterém stojí. V poslední řadě se vezmou v potaz vlastnosti terénu jednotlivých políček, které ovlivňují dohled jednotek (viz dále). Tento výpočet je nutné provést na všechny políčka ve vzdálenosti viditelnosti jednotky. Z důvodu častého výpočtu dohledu jednotek je vhodné napočítané výsledky ukládat, protože se s časem nijak nemění.

Terén

Terén hraje v případě Orodael Turrim velký význam. Terén může ovlivňovat pohyb, viditelnost a také veškeré atributy jednotek. Terén je navržen tak, aby bylo možné vhodnou kombinací vlastností jednotlivých terénů tvořit rozmanité mapy. Pokud vlastnosti jednotlivých terénů jsou dostatečně rozmanité, je pak nutností ze strany hráče s vlastnostmi terénů počítat ve svém plánování obrany.

První důležitou vlastností terénu je ovlivnění pohybu jednotek po něm. Cena pohybu mezi pozicemi je závislá jak na cílové pozici, tak na počáteční pozici. Pro každý druh terénu se definuje cena přechodu na všechny druhy terénu.

Dále také terén ovlivňuje viditelnost jednotek. Každý druh terénu může mít nadefinované, jak složité je přes daný terén vidět. Tuto hodnotu lze definovat procentuálně nebo konstantní hodnotou.

Poslední vlastností terénu je ovlivňování veškerých atributů jednotek. V Orodael Turrim lze navrhnout terény, které budou například výhodné pro obranu, a jednotce stojící na této pozici budou obranu zvyšovat. Také je možné nastavit ovlivňování životů jednotek, ať už se jedná o ztrátu z důvodu sopečného podlaží nebo léčení v klášteře.

3.1.4 Jednotky

Jednotky v Orodael Turrim jsou rozdělené do dvou frakcí – útočníci a obránci. Každý hráč může nasazovat jednotky pouze ze své frakce. Každá jednotka je definovaná svým názvem, cenou, šesti atributy a schopnostmi.

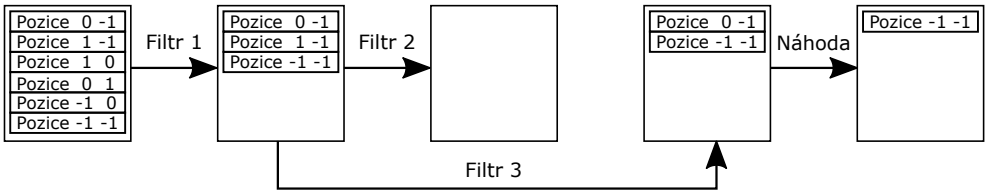
Jméno slouží pouze jako jednoznačný identifikátor jednotky.

Cena definuje, kolik je nutné za jednotku zaplatit při jejím nasazení.

Atributy udávají sílu dané jednotky a jsou rozděleny na:

- **Počet akcí** – o kolik se jednotka může pohnout
- **Útok** – po odečtení obrany jednotky udává zranění při útoku
- **Obrana** – snižuje sílu útoku útočníka
- **Dosah** – jak daleko jednotka může útočit (počítáno stejně jako viditelnost)
- **Dohled** – jak daleko jednotka vidí
- **Počet životů** - maximální počet životů jednotky

Schopnosti jednotek jsou rozděleny na aktivní schopnosti a odolnosti. Aktivní schopnost se aplikuje na cílovou jednotku při každém útoku, pokud cílová jednotka nemá odolnost. Odolnosti definují, které schopnosti jednotku nemohou ovlivnit.



Obrázek 3.5: Ukázka fungování filtrů

3.1.5 Systém pohybových a útočných filtrů

V případě ovládání hry pomocí znalostního systému je zapotřebí cílovou doménu v některých ohledech zjednodušit, aby potřebná komplexnost výsledného ZS nebyla příliš vysoká. Zjednodušení je zde vhodné převážně kvůli rozumným nárokům na komplexnost výsledného systému od studentů. Z tohoto důvodu Orodael Turrim využívá speciální systém pohybu a útoků.

Aby znalostní systém nemusel každé kolo řešit pohyby a útok veškerých jednotek obránce, využívá Orodael Turrim systém filtrů. Filtry definují chování jednotky již při jejím nasazení a toto chování se během životnosti jednotky nemění.

Každé jednotce současně s nasazením musí být definována posloupnost filtrů pro pohyb a útok. Každý filtr na svém vstupu příjme seznam možných pozic a následně vrátit vybranou podmnožinu těchto pozic. Pokud filtr vrátí prázdnou podmnožinu, tento filtr je přeskočen a pokračuje se následujícím. Pokud po využití všech filtrů zbývá stále více než jedna pozice, výsledná pozice se vybere náhodně. Na obrázku 3.5 je vidět příklad průběhu výběru pozic pomocí filtrů.

Samotné vyhodnocení pohybů a útoků pak provádí herní engine sám a znalostní systém se nemusí tímto problémem zabývat. Herní engine na konci kola každého hráče vezme všechny jeho jednotky a pro každou jednotku provede nejprve pohyb a následně útok. Po vyhodnocení všech jednotek začíná kolo dalšího hráče. Po nasazení jednotky hráči již pohyb ani útok jednotky nijak nemohou ovlivnit ani měnit již nadefinované filtry.

Orodael Turrim definuje několik základních filtrů, které jsou dostačující pro základní pohyb po mapě a útoky. V případě potřeby si uživatel může definovat vlastní specifický filtr. Filtry mohou pozice vybírat na základě různých kritérii v případě pohybu například:

- Terén na dané pozici
- Vzdálenost od pozice
- Vzdálenost nejbližší jednotky

V případě útoku to pak mohou být například:

- Terén na dané pozici
- Atributy jednotky
- Nebezpečnost jednotky
- Vzdálenost jednotky

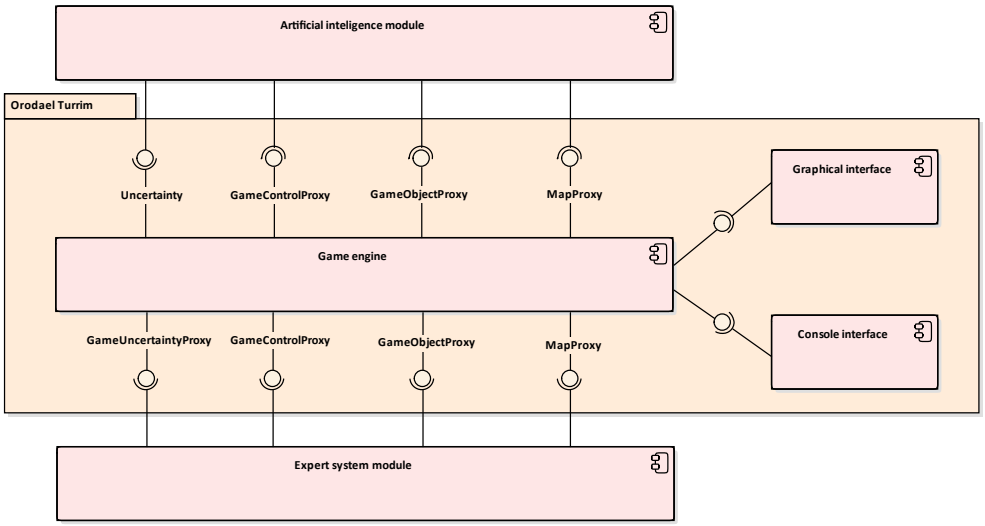
Tento systém značně zjednodušuje samotný průběh hry, ale přidává velké množství možností při plánování obrany. Hráč se stará pouze o nasazení jednotek s vhodnými filtry a na vhodná místa. O samotný pohyb jednotek už se stará herní engine. Pro úspěšný systém je zapotřebí zvolit vhodnou kombinaci jednotek a jejich definici filtrů.

3.1.6 Systém plánování útočníka

Aby bylo možné ve znalostním systému pracovat také s neurčitostí, je zapotřebí zanést neurčitost také do hry samotné. V případě pouze samotné mapy a jejich pozic lze pouze velmi obtížně poskytnout nějakou informaci s nějakou mírou neurčitosti. Z tohoto důvodu je neurčitost poskytována od umělé inteligence řídící útočníka. Orodael Turrim definuje, že umělá inteligence řídící útočníka musí zprostředkovávat informace o nasazení jednotek alespoň jedno kolo dopředu (může i více). Tento požadavek nijak drasticky neovlivňuje chování umělé inteligence, pouze veškeré principy zde fungují s jednokolovým zpožděním. Umělá inteligence může samozřejmě s touto podmínkou počítat a implementovat pokročilé metody, které řeší tento problém. Umělá inteligence musí poskytnout seznam jednotek a jejich pozic, na kterých se objeví pro následující kola. Nemusí však poskytovat informace o filtrech jednotek, což částečně vyrovnává zpoždění v plánování útoku. Informace poskytnuté od umělé inteligence jsou přesné, neurčitost do informace zanáší až herní engine.

Neurčitost zde funguje na bázi zvědů. Každé kolo obránce vypraví své zvědy, kteří mají každé kolo náhodnou úspěšnost. Podle této úspěšnosti se následně znepřesní informace o příchozích jednotkách. Na základě hodnoty úspěšnosti zvědů se podle přepočtové tabulky spočítá rozptyl pozic pro každou jednotku nepřítele. Jelikož nepřítel smí nasazovat jednotky pouze na okraji mapy, rozptyl polí se vybírá po obvodu mapy a tvoří vždy spojitou posloupnost. Těmto polím je následně přiřazena pravděpodobnost pomocí rovnoměrného rozdělení.

Toto chování simuluje ve znalostním systému neurčitost od uživatele. Znalostní systém může tuto informaci zpracovat a na základě těchto informací lépe plánovat svoji obranu. V případě úspěchu zvědů se může jednat o velice cenné informace.



Obrázek 3.6: Model architektury

3.2 Model architektury

Jelikož součástí zadání je požadavek na modularitu výsledného systému, je zapotřebí vhodně navrhnout jednotlivé moduly a jejich rozhraní. Hlavní modul Orodael Turrim je navržen jako dvouvrstvý. Veškerá herní logika se provádí v herním enginu a datová vrstva zde není vůbec vyžita. Pro komunikaci mezi herním enginem a prezentační vrstvou se využívá rozhraní samotného herního enginu. Pro komunikaci s ostatními moduly (umělé inteligence a znalostního systému) se využívá návrhového vzoru proxy. Na obrázku 3.6 je vidět UML diagram popisující jednotlivá rozhraní mezi moduly a vrstvami.

3.2.1 Proxy rozhraní herního enginu

Návrhový vzor proxy vytváří zástupce, který odstiňuje objekt od jeho uživatelů a sám řídí přístup uživatelům k danému objektu[12]. V případě potřeby může proxy k funkcionalitě zdrojového objektu přidat vlastní funkce.

Hlavní důvod pro využití proxy rozhraní je zapouzdření herního enginu. Herní engine má přístup k veškeré herní logice a pro hráče musí být zpřístupněna pouze část této logiky. V opačném případě by uživatel mohl snadno podvádět a provádět nepovolené akce. Zároveň proxy velice zpřehledňuje práci s enginem, protože k některým funkcím je přidaná dodatečná funkcionalita pro snazší použití (např. změna formátu informace).

Přístup k jednotlivým funkcím je rozdělen do čtyř rozhraní z důvodu oddělení různých informací a akcí od sebe. Toto rozdělení pak umožňuje jednotlivé proxy zpřístupnit pouze na místech, kde jsou potřeba. Detailní popis

všech funkcí rozhraní je k dispozici v dokumentaci[13]. Všechna rozhraní kromě rozhraní neurčitosti jsou společné pro modul umělé inteligence a znalostního systému. To nám zajišťuje skoro stejné podmínky pro obránce i útočníka. Jediný rozdíl je v rozhraní neurčitosti, kde modul umělé inteligence musí naopak poskytovat dodatečné informace pro herní engine a následně pro znalostní systém.

MapProxy rozhraní zprostředkovává veškeré informace ohledně herní mapy. Jsou zde informace o velikosti a terénu mapy, která je všem hráčům známá. Podrobnější informace poskytuje pouze o viditelných pozicích, kde lze například zjistit, zda se na pozici nachází nějaké jednotka. Dále zprostředkovává informace pro usnadnění práce s mapou jako například seznam viditelných polí nebo seznam polí na okraji mapy. Toto rozhraní je jednak zpřístupněno pro získávání informací pro řídící moduly, ale také se využívá k funkcionalitě pohybových a útočných filtrů.

GameObjectProxy rozhraní poskytuje veškeré informace o viditelných jednotkách. Lze zjistit například jejich atributy, aktivní efekty, odolnosti jednotek a další. Toto rozhraní je opět přístupné v modulech a také ve pohybových a útočných filtrech.

GameControlProxy rozhraní je určeno pro zpřístupnění veškerých akcí, které může obránce a útočník dělat. V aktuální verzi lze pouze nasadit jednotku na cílové pole. Funkce nasazení jednotky má navíc svá omezení. Obránce může jednotku nasadit pouze na viditelné pole (kromě krále) a útočník pouze na okraj mapy (pole nemusí být viditelná). Rozhraní `GameObjectProxy` není přístupné z filtrů, aby filtry nemohly obsahovat žádnou dodatečnou herní logiku.

GameUncertaintyProxy rozhraní zprostředkovává informace, které jsou zatíženy neurčitostí. Toto rozhraní je určeno pouze pro obránce, který díky němu získává přibližné informace o blížících se jednotkách útočníka. Modul umělé inteligence naopak poskytuje rozhraní, které poskytuje informace ohledně nasazení jednotek pro příští kola. Herní engine zde slouží jako prostředník, který nejprve přesné informace zatíží neurčitostí a následně poskytne znalostnímu systému v upraveném formátu včetně neurčitosti.

3.3 Návrh modulu znalostního systému

Další důležitou částí návrhu je samotný znalostní systém. Do návrhu je třeba zahrnout všechny požadavky na systém definované v analýze. Musí být zachována modularita řešení, aby bylo možné případné moduly snadno nahradit. Zároveň také musí být oddělena implementace uživatele od celého systému, aby bylo možné výsledné znalostní systémy snadno opravovat s jistotou, že

nebyl změněn kód ostatních modulů. Celý návrh vznikl na základě fungování standardních znalostních systémů a herního mechanismu.

3.3.1 Implementace od uživatele

Hlavní funkcionalita frameworku má umožňovat uživateli vytvořit implementaci znalostního systému. Veškeré změny, které bude uživatel implementovat, musí být ve samostatném modulu, aby byla práce s frameworkem pro uživatele přímočará. Uživatelé budou všechny změny dělat v uživatelském modulu znalostního systému.

Uživatelský modul znalostního systému je rozdělen na 4 části:

- **Báze faktů** – kde se získané informace od herního enginu převádějí do báze faktů.
- **Inference** – kde se na základě báze znalostí a báze faktů vyhodnocují závěry.
- **Báze akcí** – kde jsou nadefinované závěry inference a jejich akce, které se provedou ve hře.
- **Soubor s pravidly** – kde se definuje báze znalostní pomocí pravidel.

Uživatel bude modul postupně vylepšovat a přidávat nové funkcionality. Uživatel při práci bude upravovat společně všechny čtyři části. Znalostní systém je možné rozšiřovat postupně a iterativně vytvořit modul, který zvládne zpracovat například neurčitost v podobě fuzzy pravidel.

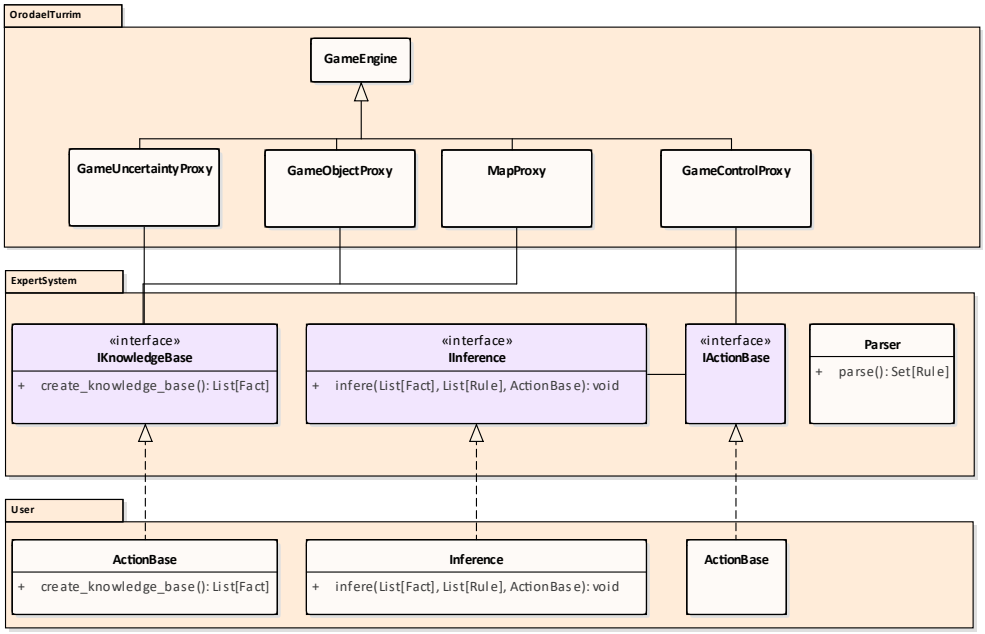
Hlavní účel uživatelského modulu znalostního systému je řízení nasazování jednotek ve hře Orodael Turrim. Uživatel musí vytvořit znalostní systém, který dokáže na základě terénu mapy a chování nepřítele plánovat svoji obranu. K testování veškerého chování je k dispozici grafické rozhraní, pomocí kterého je možné veškeré akce provést manuálně.

Výsledný systém bude následně hodnocen podle toho, kolik kol se dokáže ubránit útokům nepřítele. Tyto výsledky je následně možné porovnat s ostatními uživateli a vytvořit žebříček nejlepších řešení.

3.3.2 Návrh modulů a tříd

Na obrázku 3.7 je vidět zjednodušený diagram tříd, které se týkají fungování znalostního systému. Třídy nemají vyplněné veškeré metody z důvodu přehlednosti diagramu. Kompletní diagram se nachází v dokumentaci[13]. Znalostní systém je rozdělen na dva moduly:

- **ExpertSystem** modul implementuje podpůrné funkcionality znalostního systému a definuje rozhraní pro uživatelský modul.
- **User** modul pak slouží k zaobalení veškeré funkcionality, kterou musí uživatel implementovat.



Obrázek 3.7: Diagram tříd modulu znalostní systém

Modul ExpertSystem

V modulu `ExpertSystem` se nalézá veškerá podpůrná funkcionalita pro znalostní systém. Nejdůležitější částí je třída `Parser`, která se stará o kompletní zpracování definice pravidel. Pravidla se definují pomocí speciálního jazyka, který byl nadefinován pro potřeby Orodael Turrim (popsán níže). Ostatní podpůrné funkce jsou zabudovány v rozhraních samotných tříd.

Rozhraní `IKnowledgeBase` definuje třídu, která se stará o prvotní naplnění báze faktů. Tato třída simuluje dotazování se uživatele, které se využívá ve standardních ZS. Místo dotazování uživatele se modul dotazuje přímo herního enginu za použití předem definovaných otázek. Pro dotazování se využívají proxy zmiňované výše. Rozhraní definuje pouze jednu funkci `create_knowledge_base`, která se volá jako první pří zpracování ZS a její návratovou hodnotou je seznam faktů, které tvoří bázi faktů.

Rozhraní `IInference` je hlavní logikou celého znalostního systému. Rozhraní definuje metodu `infere`, jejíž implementace provádí inferenci. Metoda `infere` dostane jako parametry seznam faktů, které byly vytvořeny v implementaci rozhraní `IKnowledgeBase`, a seznam zpracovaných pravidel. Dále má také k dispozici odkaz na implementaci rozhraní `IActionBase`, která slouží k provedení závěrů inference.

Rozhraní `IActionBase` slouží pro definici závěrů znalostního systému. Uživatel si zde implementuje vlastní metody, přičemž jejich názvy definují seznam

možných závěrů inferenčního mechanismu. Samotná implementace těchto metod pak určuje, co daný závěr znamená a jaká akce se provede. Pro provádění úkonů ve hře má rozhraní přístup k `GameControlProxy`, přes kterou musí být prováděny veškeré akce. Dále rozhraní implementuje několik podpůrných funkcí pro snadnější volání závěrů z implementace rozhraní `IInference`.

Díky rozdělení funkcionality na jednotlivé rozhraní a přiřazení jednotlivých proxy pouze tam, kde jsou zapotřebí, je zajištěno, že uživatel musí využít standardní struktury znalostního systému. V opačném případě by uživatel mohl veškerou funkcionalitu dát pouze do jedné z tříd, a degradovat modulární strukturu na jeden dlouhý nepřehledný kód.

Modul User

Modul `User` slouží převážně jako modul, který zaobaluje veškerou funkcionalitu znalostního systému od uživatele. Celá část pro implementaci je vysunuta do jednoho modulu, aby práce s frameworkem byla přehledná a dala se snadno opravovat. Při případném opravení se vezmou referenční zdrojové soubory ostatních modulů a přidá se k nim implementovaný modul od uživatele. Tím se snadno zabrání možnosti, že uživatel změní jakoukoliv část kódu, která nesmí být změněna.

Modul v základu obsahuje ukázkovou implementaci inference. Inference umí pracovat pouze se základními pravidly a neumí přidávat závěry do báze faktů. Inference neumí nijak pracovat ani s neurčitostí. Nicméně celý modul je v základu funkční a implementuje nejzákladnější znalostní systém. Jedná se o minimální funkční kostru znalostního systému, která slouží jako ukázka fungování modulu. Pokročilé funkce modulu, jako je odvozování pravidel nebo práce s neurčitostí, musí uživatel naprogramovat sám. Nicméně tato minimální implementace zaručuje, že celý framework je bez zásahu uživatele spustitelný a obránce přežije několik kol.

3.3.3 Jazyk pravidel

Pro potřeby zpracování pravidel v Orodael Turrim je navrhnut speciální jazyk, který podporuje veškeré potřebné konstrukty. Hlavní důvod pro vytvoření nového jazyka je snaha co nejméně omezovat uživatele při tvorbě pravidel. Jazyk je nadefinován tak, aby bylo možné tvořit i složité konstrukce. Zároveň jazyk podporuje vyjádření neurčitosti a také zápis fuzzy logiky. Samotný parser nijak zapsané pravidla nevyužívá, pouze kontroluje syntaxi a zprostředkovává pravidla ve snadno použitelném tvaru. Pravidla jsou převedena do stromové struktury, kterou je následně možné rekurzivně procházet a vyhodnocovat.

Samotný jazyk je popsaný pomocí gramatiky. Kompletní definice gramatiky se nachází v příloze B. Každé pravidlo má základní tvar:

$$IF\ podmínka\ THEN\ závěr\ WITH\ pravděpodobnost;$$

kde IF a $THEN$ jsou terminální slova, která dělí pravidlo na podmínku a závěr. Konec pravidla, obsahující terminální slovo $WITH$, není povinný a definuje neurčitost celého pravidla.

Podmínka se pak následně přepisuje na jednotlivé funkční výrazy, které mohou být spojeny logickými spojkami konjunkce AND a disjunkce OR. Zápis také umožňuje definici vnoření pomocí uzavření výrazů do kulatých závorek. Funkční výraz má tvar:

$$identifikátor\ parametry\ operátor\ hodnota\ [neurčitost]$$

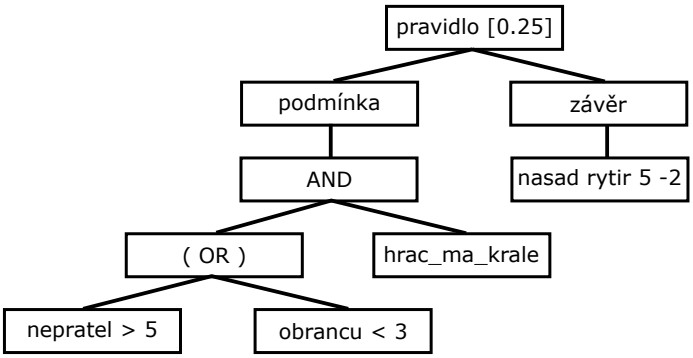
kde

- *identifikátor* je textový řetězec určující název funkce,
- *parametry* jsou seznam čísel a textových řetězců, které funkci parametrizují a využívají se při vyhodnocování funkce,
- *operátor* lze nahradit standardními matematickým operátory pro porovnání čísel $>, <, ==$ a slouží k převedení číselné hodnoty funkce na pravdivostní hodnotu,
- *hodnota* je číselná nebo textová a slouží jako porovnávací hodnota pro operátor,
- *neurčitost* v hranatých závorkách je číslo, které určuje neurčitost jedné části pravidla.

Závěr má velice podobný formát zápisu, ale nepodporuje logickou spojku disjunkce OR, protože v závěru nemá smysl. Také je zde operátor porovnání nahrazen operátorem přiřazení, který umožňuje definovat závěru hodnotu (číslo nebo text). Tento zápis se převážně hodí, pokud se závěr přidá do báze faktů a dále se s ním pracuje. Výsledné pravidlo může pak například vypadat takto:

$$\begin{aligned}
&IF\\
&\quad hrac_ma_krale\ AND\ (\ nepratel\ >\ 5\ OR\ obrancu\ <\ 3\ )\\
&THEN\\
&\quad nasad_jednotku\ rytir\ 5\ -2\\
&WITH\\
&\quad WITH\ 0.25;
\end{aligned}$$

Toto pravidlo analyzátor převede do stromové struktury, která je znázorněna na obrázku 3.8.



Obrázek 3.8: Stromová struktura pravidla

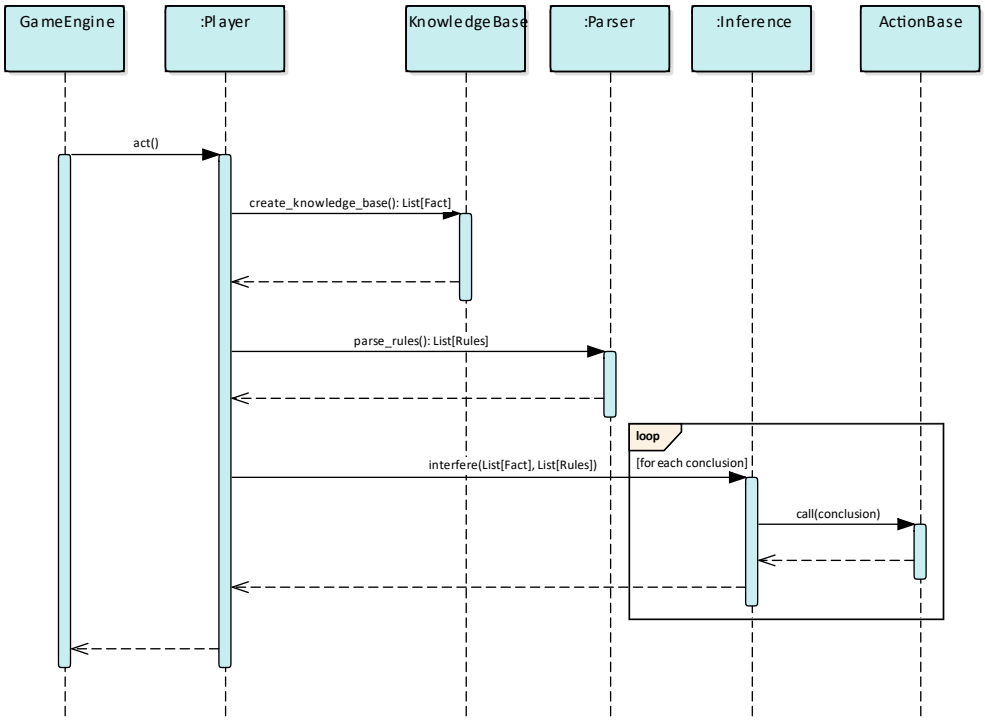
3.3.4 Model komunikace znalostního systému

Samotné volání jednotlivých metod při zpracování znalostního systému je znázorněno na diagramu komunikace na obrázku 3.9. Herní engine zavolá na instanci obránce funkci `act`, která se stará o průběh zpracování jednoho kola. Nejprve získá ze třídy `KnowledgeBase` bázi faktů ve formě seznamu jednotlivých faktů. Následně získá od třídy `Parser` pravidla ve formátu stromové struktury. Tato získaná data pak vloží jako argumenty třídy Inference, kde probíhá samotná inference. Inference následně zpracuje všechny pravidla a fakty. Získané závěry zavolá jako metody nad třídou `ActionBase`. Třída `ActionBase` se postará o to, aby cílové akce byly provedeny.

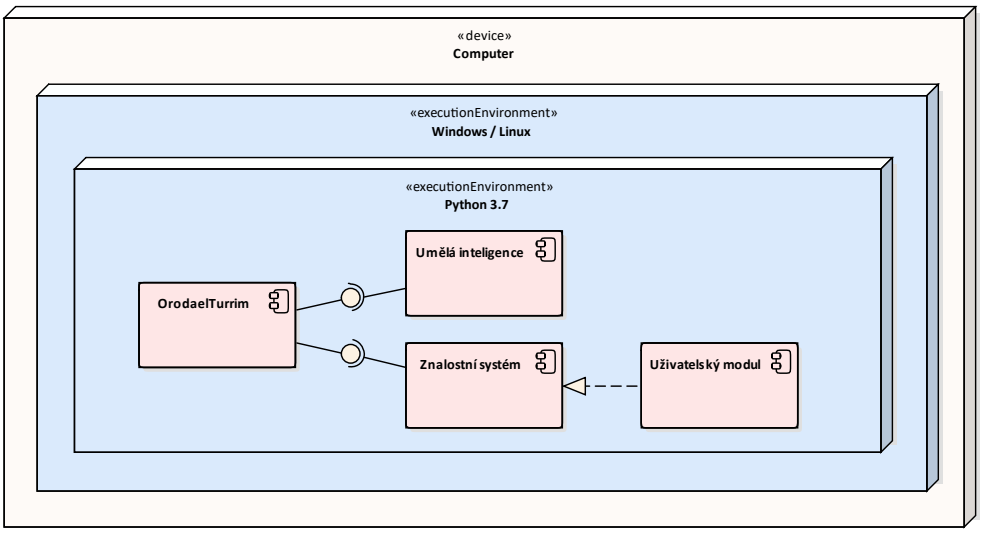
3.4 Model nasazení

Model nasazení zobrazuje veškeré závislosti a jednotlivé komponenty, čímž definuje architekturu celého systému. Model nasazení se nejčastěji znázorňuje pomocí UML diagramu s dodatečným textovým popisem.

Na obrázku 3.10 je vidět, že Orodael Turrim je určen pro počítače s platformou Linux nebo Windows. Celý systém navíc vyžaduje pro svůj běh Python verze 3.7. V případě frameworku nedává smysl, aby byla aplikace zabalena do spustitelného balíčku včetně Python prostředí, protože uživatelé budou upravovat kód modulu znalostního systému. Samotný Framework je pak rozdělen do 4 oddělených modulů, které mezi sebou navzájem komunikují pomocí rozhraní.



Obrázek 3.9: Diagram komunikace modulu znalostní systém



Obrázek 3.10: Diagram nasazení

Kapitola 4

Implementace

Tato kapitola se zabývá implementací samotného frameworku. Popisuje a zdůvodňuje výběr programovacího jazyka a dalších podpůrných knihoven. Dále je zde popsán průběh definování herních objektů, jejich parametrů a grafiky. Poslední část se věnuje rozhraní celé aplikace a zajímavým částem implementace.

4.1 Programovací jazyk a knihovny

Jedním z nejdůležitějších rozhodnutí implementace je volba programovacího jazyka a použitých knihoven. Programovací jazyk je vhodné zvolit až po provedení důkladné analýzy, kdy jsou již známé veškeré požadavky na systém. Na základě výsledných požadavků jako jsou rychlost, jednoduchost rozšíření a další požadavky se vybere programovací jazyk.

Na základě požadavků získané v analýze a návrhu frameworku byl zvolen jazyk Python. Python má mnoho vlastností, které se v případě tohoto frameworku velice hodí. Jelikož hlavním účelem frameworku je umožnit studentům naprogramovat si vlastní znalostní systém, je dobré, že se jedná o vysokoúrovňový jazyk. Programování v Pythonu je velice rychlé a intuitivní a studenti se nemusí zabývat nízkoúrovňovými záležitostmi, jako je například správa paměti. Další velkou výhodou je, že se jedná o interpretovaný skriptovací jazyk. Díky tomu je výsledný framework velice jednoduše a rychle testovatelný. Uživatel nemusí při každé změně celý framework znovu kompilovat, pouze ho spustí a okamžitě vidí výsledek provedených změn. Této vlastnosti se například využívá při změně pravidel báze znalostí. Pokud se změní soubor s pravidly, není nutné framework restartovat, ale stačí pouze znovu spustit inferenci. Protože framework má sloužit jako prostředí pro ladění znalostního systému, je tato vlastnost velice užitečná. V neposlední řadě je také velice užitečné, že se jedná o netypový jazyk. Tato vlastnost umožňuje studentům upravit si modul uživatele k vlastním potřebám, jako například změnit datové typy informací. Celý uživatelský modul má datové tipy pouze doporučené, uživatel si je může změnit

na vlastní pokročilejší. Tyto změny můžou být provedeny bez nutnosti zásahu do hlavní části frameworku.

Jednou z nevýhod Pythonu je jeho rychlost. V porovnáním převážně s kompilovanými jazyky, je rozdíl velice znatelný. Nicméně Python umožňuje různé nízkoúrovňové optimalizace, jako je například práce se sloty, která částečně řeší problém s výkonem. Z tohoto důvodu byla také zvolená verze Python 3.7, která má oproti předchozím verzím značné vylepšení rychlosti například při volání metod[14]. Mimo to tato verze podporuje nové syntaktické prvky, které značně zpřehledňují výsledný kód. Z důvodu dosažení co nejlepšího výkonu framework obsahuje některé části kódu, které nejsou zpětně kompatibilní se starší verzí Pythonu. Nicméně Python 3.7 je již plně podporovaná a doporučená verze, takže vynucení této verze není problém.

4.1.1 Grafická knihovna

Další důležitou volbou v Orodael Turrim je grafická knihovna. Jelikož framework je na pomezí počítačové hry a standardní grafické aplikace, není volba ideálního grafického prostředí jednoduchá. Pro vývoj 2D her má Python velké množství knihoven jako například `PyGame`, `Pyglet` nebo `Arcade`. Nicméně Orodael Turrim je spíše framework pro testování hry a zkoušení herních mechanismů než hra samotná. Z tohoto důvodu, je mnohem větší důraz kladen na ovládací prvky samotného testování než intuitivní hraní hry. Z toho důvodu Orodael Turrim nepoužívá žádnou knihovnu pro vytváření her, ale využívá knihovnu pro tvorbu klasického uživatelského rozhraní.

Pro tvorbu uživatelského rozhraní Python nabízí také širokou škálu knihoven jako například `Tkinter`, `PyQt` nebo `KIVY`. Zásadní podmínkou pro vybranou knihovnu je dobrá podpora grafické scény, pomocí které bude možné zaintegrovat herní prvky. Hlavní požadavky na grafickou scénu jsou rychlé více-úrovňové vykreslování a snadná práce s grafickými prvky.

Tkinter je vestavěná knihovna, která se velmi často používá pro jednoduché grafické rozhraní. Bohužel ale podpora grafické scény je v knihovně naprosto nedostačující, a proto je pro Orodael Turrim nevhodná. Zásadní problém je, že v grafické scéně nelze pracovat s vrstvami.

Alternativou by mohla být knihovna `KIVI`[15], která je postavena na standardu `OpenGL`. Díky podpoře webového prostředí a také mobilních zařízení, je výsledný design knihovny spíše určen pro dotykové ovládání. Jelikož Orodael Turrim má za cíl běžet převážně na nedotykových zařízení, mohl by být výsledný design pro uživatele příliš komplikovaný či neintuitivní.

Knihovna `PyQt`[16] je Python verze velice populární knihovny `Qt` pro C++. Je zaměřena na vytváření standardních grafických rozhraní, které jsou pro uživatele dobře známé. Zároveň podporuje i prvky grafické scény, která zvládá veškeré potřeby pro vykreslování herních prvků aplikace. Díky grafickému návrháři je vývoj aplikací velice rychlý a intuitivní.

Pro Orodael Turrim byla zvolena knihovna `PyQt`, protože podporuje všechny potřebné funkce a vývoj je zde velice rychlý díky grafickému návrháři. Konkrétně Orodael Turrim využívá `PyQt` verze 5, která je doporučená pro Python 3.7.

4.1.2 Knihovna zpracování jazyku pravidel

Další důležitá část aplikace, která usnadňuje uživatelům práci se znalostním systémem, je jazyk pravidel. Orodael Turrim používá vlastní navržený jazyk (viz sekce Návrh) pro zápis jednotlivých pravidel, který dokáže analyzovat a případně zprostředkovat uživateli v snadno použitelné podobě. Jelikož navržený jazyk má značnou vyjadřovací schopnost, jeho syntaxe může být v některých případech velice složitá. Z tohoto důvodu není možné použít pro analýzu jednoduché metody dělení textu podle znaků nebo regulární výrazy.

Pro složitější jazyky je zapotřebí využít složitější analyzátory, které analyzují text na základě gramatiky. Knihoven pro vytvoření vlastních analyzátorů existuje mnoho, nicméně pro základní použití jsou zbytečně složité. Jednoduší způsob je využít knihovny, které na základě zadané gramatiky vygenerují již hotový analyzátor. Práce využívá generátor analyzátoru `ANTLR`[17], který jako jeden z mála podporuje generování analyzátoru v jazyce Python verze 3.

ANTLR (Another Tool for Language Recognition) je jednoduchý ale mocný generátor LL syntaktického analyzátoru. Pomocí jednoduchého jazyka lze specifikovat gramatiku cílového jazyka a pomocí jednoho příkazu vygenerovat výsledný analyzátor. ANTLR podporuje definici jazyka pomocí bezkontextové gramatiky. Analyzátor podporuje mimo jiné stromové procházení analyzovaného textu, tudíž je velice snadné převést cílový jazyk do stromové struktury. Navíc podporuje kontrolu syntaxe s chybovými výstupy, které uživateli velice usnadní hledání chyb v nadefinovaných pravidlech. Práce využívá nejnovější verzi knihovny ANTLR 4. Syntaxe gramatiky je vidět na ukázce kódu 1, kde je pomocí gramatiky popsán jednoduchý zápis matematického výrazu.

```
grammar Expr;
prog:       (expr NEWLINE)* ;
expr:       expr ('*'|'/') expr
    |       expr ('+'|'-') expr
    |       INT
    |       '(' expr ')'
    ;
NEWLINE : [\r\n]+ ;
INT     : [0-9]+ ;
```

Ukázka kódu 1: Ukázka definování gramatiky v ANTLR

Obrázek 4.1: Hornatý terén

Obrázek 4.2: Vesnice

4.2 Definice herních objektů

Důležitou roli ve výsledné kvalitě hry také hrají herní prvky, jako jsou jednotky, terén a další. Aby nedocházelo k redukci použitelných objektů, je nutné mít veškeré herní prvky vyvážené. Pokud by tomu tak nebylo, uživatelé velice rychle přestanou využívat nepoužitelné části, čímž se sníží rozmanitost celé hry. Aby uživatele hra bavila, je také velice důležitá vizuální stránka herních objektů. Detailní popis všech herních objektů a jejich vlastností lze najít v dokumentaci[13].

4.2.1 Terén

První z trojice herních objektů je terén mapy. Terén zde nemá pouze vizuální funkci, ale také značně zasahuje do herních mechanismů. Druhy terénů například ovlivňují, jak rychle se jednotky můžou po něm pohybovat nebo jaká políčka na mapě vidí. Mimo to mohou také zlepšovat či zhoršovat atributy jednotek.

Jednotlivé vlastnosti terénů se se během práce na diplomové práci měnily z důvodu balancování vlastností terénu. Cílem balancování v tomto případě není mít všechny druhy stejně silné, ale docílit rozmanitosti terénu. Důležité je, aby terén mapy hrál důležitou roli ve vybírání počáteční pozice bránících hráčů. Hráči pak mohou například naučit svůj znalostní systém, aby upřednostňoval těžko přístupné pozice pro ukrytí svého krále. Sílu jednotlivých terénů je také dobré zohlednit v nastavení generátoru mapy. V generátoru je možné nastavit, jak častý daný terén bude. Pomocí této kombinace může být například vytvořen velmi silný terén, který se však bude na mapě nacházet zřídka.

Orodael Turrim ve verzi přiložené k práci má nadefinovaných šest druhů terénů: pláně, les, kopec, hory, řeka a vesnice. Každá z nich má unikátní vlastnosti více či méně ovlivňující jednotky. Pro ukázku dva vybrané terény:

Hornatý terén je velmi těžko dostupný, ze všech terénů kromě hornatého stojí pohyb tři akce. Jednotce stojící na hoře poskytuje značné bonusy k obraně (50 %) a výrazně zvyšuje dohled (+3). Aby terén nebyl příliš

Obrázek 4.3: Démon

Obrázek 4.4: Kouzelník

Obrázek 4.5: Král

výhodný, snižuje jednotce útok o 20 %. Navíc jednotka stojící na hoře ztrácí každé kolo 5 % svých životů. Velice důležitou vlastní hory je zvýšení dohledu, díky kterému jednotka stojící na osamocené hoře může vidět velkou část mapy, a tím přinášet velkou výhodu obránci.

Vesnice je naopak dobře přístupná. Lze využít okolních cest, takže zvyšuje množství akcí o 1. Také zvyšuje bránícím se jednotkám obranu o 30 %. Vesnice je velice výhodný terén a je nutné ji brát v potaz při plánování obrany. Převážně je důležité nenechat okolní vesnice padnout do rukou útočníků, protože zvýšená obrana útočníka může způsobit nemalé problémy.

4.2.2 Jednotky

Druhý druh herních objektů jsou jednotky. Pro všechny jednotky je použita fantasy tématika. Obránci se skládají z elfů a lidí a útočníci jsou složeni z různých nepřátelských ras jako orkové, nemrtví, démoni a další.

Balancování jednotek

Poměrně složitý problém je balancování jednotek. Pod balancováním se zde myslí vyrovnání atributů všech jednotek v poměru k ceně jednotky. Aby hráči mohli využívat veškeré jednotky, je zapotřebí docílit co nejlepší vyrovnanosti sil mezi nimi. U jednotek se také v žádném případě nesmí stát, aby některá z jednotek byla neporazitelná (měla příliš velkou obranu). V takovém případě by se z hra stala nesmyslnou. S balancováním jednotek se potýká velká spousta her.

Balancování jednotek v případě Orodael Turrim probíhalo během vývoje iterativně, přičemž důležitou roli zde hrálo uživatelské testování.

V první iteraci byl nadefinován seznam jednotek, jejich grafická podoba a první odhad atributů na základě cílené síly jednotky. Tím vzniklo základní dělení jednotek a první verze jejich atributů. První verze nebyla nijak balancovaná, šlo převážně o definici, které jednotky mají být silné a které nikoliv. Toto rozdělení vzniklo pouze jako zajímavý prvek hry.

V druhé iteraci bylo využito způsobu balancování pomocí bitev. Byla vytvořena tabulka všech možných soubojů a jejich výsledky. Výsledkem se myslí, kolik jednotek útočníků je zapotřebí na poražení obránce. Útočníci útočí postupně a začíná útočník. Pokud se tato informace dá dohromady s cenou útočící a bránící jednotky, lze toto číslo využít jako hodnotu pro balancování. Samozřejmě by bylo dobré vzít také v potaz schopnosti jednotek a ostatní atributy jednotek, které neovlivňují boj přímo (pohyblivost, dohled atd.). Balancování všech atributů se věnuje následující iterace, proto se tento způsob omezil pouze na výsledky bitvy jeden na jednoho. Tato iterace našla a odstranila zásadní problémy příliš silných jednotek v bitvě. V úvahu by se také měla vzít skutečná bitva, kde jednotky nebojují jedna proti jedné. Bohužel komplexnost této metody je natolik vysoká, že v práci nebyla použita.

Ve třetí iteraci se využilo numerického balancování. Na základě všech atributů a ceny jednotky byly vytvořeny poměry mezi jednotkami. Nejprve se všechny atributy přeškálovaly na hodnoty 0 až 1, aby bylo docíleno porovnatelnosti mezi nimi. Následně na základě poměru přeškálovaných atributů a ceny vznikla hodnota definující každou jednotku. Na základě této hodnoty je provedeno balancování.

Čtvrtá iterace byla založena převážně na uživatelském testování. Na základě vytvořených znalostních systémů od uživatelů je možné vyvodit, které jednotky jsou příliš silné nebo příliš slabé. Pokud je k dispozici velké množství testerů, lze považovat tuto metodu balancování za nejpřesnější. Každopádně předchozí iterace jsou také velice důležité, převážně kvůli urychlení a zefektivnění uživatelského testování.

Výsledné jednotky

Na základě zmíněných čtyř iterací vznikl seznam jednotek, které mají útočník a obránce k dispozici. Orodael Turrim ve verzi přiložené k práci má nadefinováno 5 obránců a 8 útočníků. Jako příklad je zde uveden jeden zástupce útočníků a jeden zástupce obránců.

Démon je jeden z nejsilnějších útočníků. Díky vysokému útoku a velkému rozsahu útoku se z něj stává velice nebezpečný nepřítel. Navíc útočí pomocí ohnivé koule, která jeho nepřátele popálí, a způsobuje dodatečné poškození v následujících kolech. Na druhou stranu má nízkou hodnotu obrany, což z něj dělá snadný cíl.

Kouzelník je obránce, který dokáže udělit největší poškození svým protivníkům díky vysokému útoku. Také ovládá ohnivá kouzla, která dokáží jeho nepřátele popálit. Nicméně má velice nízké životy a nízkou obranu, což z něj dělá snadnou kořist, pokud není bráněn ostatními jednotkami.

Speciální jednotkou obránce je král. Krále může mít obránce pouze jednoho a jeho úkolem je ho chránit. Jednotka musí být nasazena hned v prvním kole.

Král má velké množství životů oproti ostatním jednotkám, aby případné rychlé výpady útočících jednotek předčasně neukončily bitvy. Nicméně pokud král zůstane delší dobu nechráněn, může velice rychle padnout. Král nemá žádné speciální schopnosti a nestojí žádné peníze při nasazení.

4.2.3 Schopnosti jednotek

Posledním herním objektem jsou schopnosti jednotek. Všechny schopnosti jednotek se projevují při jejich útoku a mohou mít různou dobu trvání. Veškeré schopnosti poškozují nepřítele, a to buď pomalým ztrácením životů nebo dočasným snížením jeho atributů. Některé jednotky mohou mít také odolnost proti těmto schopnostem. V takovém případě schopnost útočníka se nijak neprojeví. Mezi schopnosti patří například popáleniny, při kterých jednotka ztrácí 5 % z maximálního počtu životů každé kolo. Zástupcem schopností, které ovlivňují atributy jednotky, je například zmrazení, které snižuje počet akcí zasažené jednotky na polovinu.

4.3 Rozhraní aplikace

Hra Orodael Turrim poskytuje uživatelům dvě uživatelská rozhraní:

- Grafické – pro testování mechanismů hry a snadné testování znalostního systému,
- Konzolové – pro automatické a rychlé otestování znalostního systému.

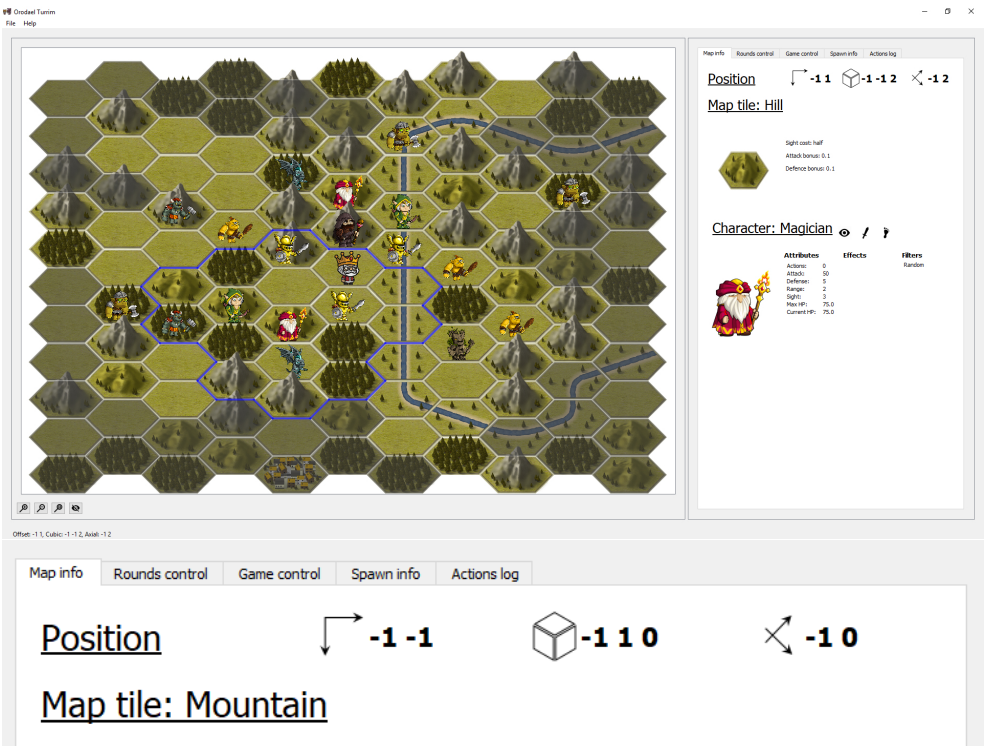
Díky modulárnosti návrhu je také možné rozšířit framework o další rozhraní, například webové, pro vyhodnocování řešení online.

4.3.1 Grafické rozhraní

Důležitou roli v Orodael Turrim má grafické rozhraní. V grafickém prostředí lze provádět veškeré operace, které umožňuje rozhraní poskytnuté pro znalostní systém. Přesto, že je možné celou hru ovládat pomocí klávesnice a myší, hra pro tento způsob ovládání nebyla navržena. Proto také některé prvky grafického prostředí jsou více zaměřeny na snadné získávání informací než na pohodlné hraní. Grafické prostředí by mělo opravdu sloužit pouze k pochopení herních mechanismů a ladění znalostního systému.

Hlavní okno na obrázku 4.6 je rozděleno na dvě části, zobrazení mapy na levé straně a ovládací prvky na pravé. Velkou část hlavního okna zabírá pak právě mapa samotná. Na ní lze vidět terén mapy, aktuální pozici veškerých herních objektů a například hranice viditelnosti.

Pravá část rozhraní pak obsahuje do záložek rozřazené ovládací prvky. Všechny záložky využívají pro zobrazení či ovládání také mapu.



Obrázek 4.6: Grafické rozhraní s detailním pohledem na menu

Zobrazení mapy tvoří hlavní část základního okna. V případě potřeby lze poměr mezi mapou a ovládacími prvky libovolně měnit. Zobrazení mapy mimo terén a jednotky také zobrazuje důležité hranice pozic a viditelnost. Na obrázku 4.6 je vidět, že některé pozice jsou zašedlé. To znázorňuje, že daná pozice není aktuálním hráčem viditelná. Dále je také na obrázku vidět ukázka hranice, která se v rozhraní využívá pro zobrazení dohledu jednotek, vybrané aktuální pozice a další. Hranice mají různé barvy podle jejich významu. Červená barva znázorňuje vybrané pole, modrá barva pozice, na které může jednotka útočit, a oranžová barva slouží ke zobrazení neurčitosti.

Záložka informací o mapě (Map info) zobrazuje veškeré dostupné informace ohledně vybraného políčka na mapě. Zobrazuje pozici ve všech souřadných systémech, vlastnosti terénu vybraného pole a případně informace o herním objektu, pokud se nějaký na zvoleném poli nachází.

Záložka řízení kol (Round control) obsahuje prvky pro simulování jednotlivých kol a nachází se zde ovládací prvky pro procházení historie bitvy. Posunutí akcí zpět do historie se také projeví na mapě, takže

je ihned vidět, jak daná situace vypadala v daném bodě historie. Tato funkcionalita umožňuje velice snadné a přehledné hledání chyb. Také je možné z této záložky spouštět inferenční mechanismus.

Záložka ručního ovládání (Game control) se využívá, pokud hráč nechce používat znalostní systém, ale ovládat své jednotky pomocí grafického prostředí. Nachází se zde seznam všech jednotek obránce včetně popisu jejich vlastností. Je zde také možnost nastavení filtrů pro jednotky, takže uživatel může vyzkoušet veškeré kombinace a chování filtrů.

Záložka příchodu nepřátel (Spawn info) zobrazuje očekávané pozice, na kterých se nepřítel v příštích kolech objeví. Jelikož informace o příchodech hráč nemá přesné, zobrazují se zde rozsahy políček a jejich pravděpodobnost.

Záložka historie akcí (Action log) slouží k přehlednému zobrazení veškerých akcí všech hráčů, které byly provedeny. Pro větší přehlednost se zde používá stromová struktura. V případě procházení historie log zobrazuje aktuální pozici v historii.

4.3.2 Konzolové rozhraní

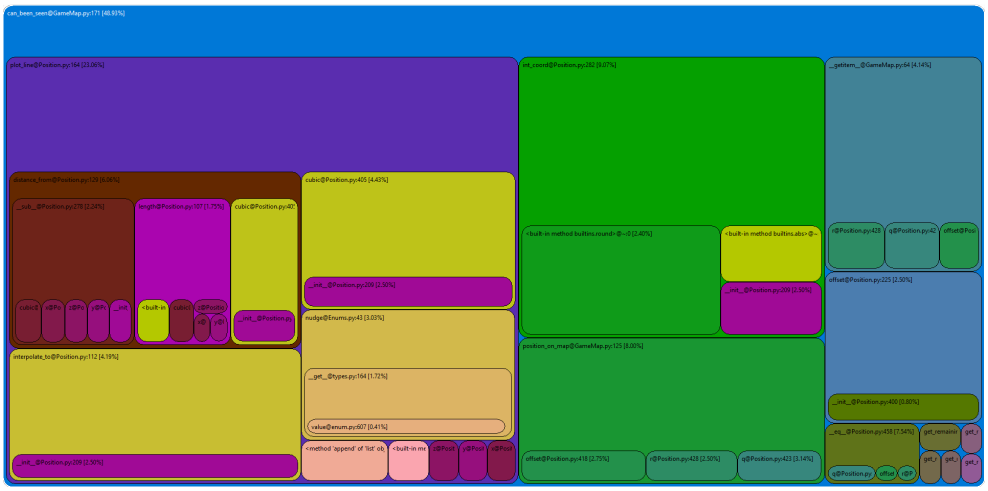
Konzolové rozhraní hry je primárně určeno pro výsledné testování kvality znalostního systému. Konzolové rozhraní při počítání jednotlivých kol nic nevykresluje a nijak neinteraguje s uživatelem. Díky tomu je mnohem rychlejší a hodí se pro počítání výsledků velkého množství kol. V průběhu hry rozhraní nepodává žádné informace o průběhu, pouze na konci zobrazí informace ohledně výsledku bitvy.

Pomocí přepínačů je možné částečně definovat chování, jako je například maximální počet simulovaných kol nebo použitý konfigurační soubor. Pokud uživatele zajímá, jak probíhal souboj, lze si nechat uložit soubor s celou historií bitvy ve formátu HTML. V tomto souboru lze najít veškeré akce všech herních objektů. Druhou možností, jak zjistit jak bitva probíhala, je uložit si počáteční nastavení náhodných algoritmů a toto nastavení následně použít v grafickém rozhraní. Pokud uživatel pomocí konfiguračního souboru nastaví stejné hodnoty náhodných generátorů, lze stejný průběh simulovat i v grafickém prostředí. Do hry samozřejmě nesmí být manuálně zasaženo, jinak se vývoj bitvy naprosto změní.

Detailní popis chování rozhraní a veškerých přepínačů je dostupný v dokumentaci a také přímo v nápovědě samotného rozhraní.

4.4 Profilování frameworku

Během implementace frameworku se velmi rychle ukázal problém s rychlostí výpočtu jednotlivých kol. Protože jedním z požadavků na framework je rychlá



Obrázek 4.7: Ukázka profilování funkce can_been_seen

odezva jednotlivých kol, bylo zapotřebí framework znatelně zrychlit. Pomocí nástrojů byla provedena jednak analýza využití procesoru, ale také paměťová náročnost frameworku.

4.4.1 Profilovací programy

Pro profilování v Pythonu existuje několik možností. Mezi nejpoužívanější se používá knihovna **cProfile**. Knihovna cProfile slouží k analýze procesorového času pro jednotlivé funkce a metody v implementaci. Výsledné informace poskytuje buď v textovém formát nebo v v binárním formátu, který lze následně načíst do analyzačních programů. Pro snadnou analýzu se v práci využívá programu `Run Snake Run`[18], který zobrazuje naměřená data v přehledném hierarchickém grafu 4.7.

V práci se kromě `cProfile` používá také pokročilý nástroj **vprof**[19]. Nástroj `vprof` zvládá kromě analýzy procesorového času také analýzu využití paměti. Také nabízí pokročilé metody analýzy kódu jako například `Code heatmap`, která obarvuje jednotlivé řádky kódu podle strávené doby na daném řádku.

4.4.2 Nalezené problémy

První nalezený problém při profilování byla funkce `get_visible_tiles`. Tato funkce se stará o zjištění viditelných pozic každé jednotky. Tato funkce je početně poměrně náročná, protože se zde musí provádět interpolace mezi všemi potencionálními pozicemi. Značného zrychlení se zde dosáhlo pomocí ukládání již vypočítaných výsledků. Jelikož se mapa během hry nemění, je možné

zde ukládat seznam viditelných pozic podle klíče pozice jednotky a dohled. Při využití vhodné hashovací funkce lze zrychlit již vypočítané hodnoty na konstantní rychlost. Toto zrychlení se projeví převážně po několika odehraných kolech, kdy se slovník napočítaných hodnot dostatečně naplní. Možné řešení by zde také bylo všechny možné kombinace vypočítat při inicializaci hry. Nicméně pro velké mapy by mohlo být načítání frameworku velice pomalé, a tudíž nevhodné pro rychlé testování.

Druhý nalezený problém se týkal práce s pozicemi. Problém je zde převážně při přepočítávání mezi jednotlivými pozičními systémy. Převážně přepočet mezi posuvným a kubickým pozičním systémem je časově náročnější. Na základě této analýzy byla provedena značná úprava práce s pozičními systémy. Až na výjimky se veškerá reprezentace převedla na kubický souřadný systém, ve kterém jsou všechny operace nejrychlejší. Tato úprava zrychlila běh celého frameworku více než třikrát.

Poslední problém, co analýza ukázala, je problém samotného Pythonu. Python má poměrně velký overhead při volání metod tříd. Nejedná se o žádné dramatické číslo, nicméně při velkém množství volání metod je problém znatelný. V případě frameworku se jednalo o velice častou práci s pozicemi a dotazování na jednotlivé složky pozice. Částečně lze tento problém řešit pomocí magické metody `__slots__`, která optimalizuje třídu pro rychlou práci s atributy a také menší paměťovou náročnost. Tato úprava ale nebyla dostačující pro potřebné zrychlení. Z tohoto důvodu framework striktně vyžaduje Python verze 3.7 a vyšší. V této verzi totiž byla provedena velice výrazná optimalizace volání funkcí a metod. Analýza následně ukázala, že v případě Pythonu 3.7 je framework dvakrát až třikrát rychlejší. Zrychlení se netýkalo pouze volání funkcí a metod, ale také veliké zrychlení některých vestavěných knihoven.

4.5 Zajímavé části implementace

Programovací jazyk Python patří mezi jazyky, které jsou vhodné pro začátečníky díky své jednoduchosti. Nicméně pokud nezůstaneme u implementačních základů, umožňuje Python také velice pokročilé funkce a přístupy k řešení problémů. Zde jsou popsána tři zajímavá implementační řešení, která nemusí být na první pohled jasná a snadno realizovatelná.

4.5.1 Privátní atributy v jazyce Python

Kromě výše zmíněných výhod intepretovaného jazyka, jsou zde také určité nevýhody. Jednou z nevýhod intepretovaných jazyků je absence privátních metod. Znemožnění přístupu k privátním metodám se v kompilovaných jazycích řeší pomocí kompilace. V interpretovaných jazycích tuto možnost nemáme.

Python ve svém standardu podporuje privátní proměnné, nicméně se jedná pouze o konvenci. Standard definuje, že proměnné a metody začínající dvěma podtržítky jsou privátní. Bohužel metoda či proměnná se nestane nepřístupná,

pouze bude automaticky přejmenována (mangling) a stále přístupná pod novým jménem.

Nicméně pro zamezení přístupu do herní logiky a proměnných je nutné nasimulovat chování privátních metod, aby uživatelé nemohli podvádět. Jedna z možností, která řeší privátní metody, je vytvoření rozhraní mezi třídami pomocí jazyka C. Bohužel tato metoda vyžaduje dodatečný překladač, čímž působí problémy s přenositelností a zbytečně komplikuje práci s výsledným frameworkem.

Metoda, která je použita v implementaci pro dosažení privátních metod, nepotřebuje žádné jiné jazyky ani externí knihovny. Pro zamezení přístupu k herní logice probíhá veškerá komunikace mezi hráči a hlavním rozhraním pomocí proxy. Tento zástupce po uložení ukazatelů na povolené funkce smaže svůj ukazatel na hlavní rozhraní, tudíž přes něj nejde k hlavnímu rozhraní přistoupit. Aby nebylo možné podvodně vyměnit hlavní rozhraní, po prvotní inicializaci třída smaže svůj konstruktor. Tento druh privátních metod je možné obejít, ale vyžaduje pokročilé znalosti fungování jazyka Python a je velmi snadno odhalitelný. Pro potřeby zamezení snadného podvádění ze strany studentů je tato metoda zcela dostačující.

4.5.2 Dynamické načítání filtrů

Jelikož herní návrh umožňuje uživatelům definici vlastních pohybových a útočných filtrů, je zapotřebí implementace automatického načítání těchto filtrů. V tomto případě lze opět využít vlastností intepretovaného jazyka. V Pythonu je možné za běhu frameworku pomocí vestavěné knihovny `inspect` získat informace o cílovém souboru či třídě jako například jméno, metody, atributy metod a další. Následně je možné tyto informace využít pro inicializaci třídy za běhu frameworku.

Mimo jiné knihovna `inspect` také umožňuje zjištění vlastností tříd. Díky tomu je možné zkontrolovat, že třída splňuje veškeré definované vlastnosti jako například dědičnost, přítomnost potřebných metod a další. Tyto kontroly znemožní uživatelům jednak podvádět při vytváření filtrů, ale také zamezí nechtěným problémům. Díky této metodě při načítání filtru s chybou aplikace nespadne, ale pouze nahlásí důvod nenačtení filtru.

Možnost získání argumentů metod se v implementaci využívá k pracování s dynamickým počtem parametrů v konstruktoru. Filtry podle definice umožňují při inicializaci předávat vlastní parametry. Aby bylo možné tyto parametry zadávat i pomocí grafického rozhraní, je zapotřebí dynamicky přidávat prvky na základě parametrů konstruktoru. Díky této vlastnosti si může uživatel vyzkoušet i vlastní definované filtry s parametry přímo v grafickém rozhraní.

4.5.3 Komunikace mezi objekty

```
class Connector(QObject, metaclass=QtSingleton):
    redraw_ui = pyqtSignal()
    redraw_map = pyqtSignal()
    display_border = pyqtSignal(dict, list)

    def subscribe(self, name: str, target: Callable) -> None:
        try:
            getattr(self, name).connect(target)
        except AttributeError:
            sys.stderr.write('Signal {} not defined!\n'.format(name))

    def emit(self, name: str, *args, **kwargs) -> None:
        try:
            getattr(self, name).emit(*args, **kwargs)
        except AttributeError:
            sys.stderr.write('Signal {} not defined!\n'.format(name))
```

Ukázka kódu 2: Implementace třídy Connector

Při vývoji počítačových her se často využívá vzor komunikace pomocí zpráv (messaging pattern). Ve hrách častokrát dochází k velké provázanosti událostí, které vyvolávají mnoho následků. Díky komunikaci pomocí zpráv je možné tuto provázanost velice snížit. Jelikož je práce z velké části postavena na herním principu, tento problém s vysokou provázaností zde vzniká také.

Framework využívá zprávy k informování všech komponent o událostech, které nastaly během simulace jednotlivých kol. Velice podobný princip využívá také zvolená grafická knihovna. Knihovna PyQt5 používá pro ovládání GUI signály. Signály vyvolávají jednotlivé prvky grafického rozhraní na základě uživatelských akcí. Signály jsou následně zachyceny a přeposlány na předem definované metody a funkce.

Aby se zamezilo zdvojení jedné funkcionality, jsou signály z grafické knihovny rozšířeny o funkcionalitu standardního vzoru posílání zpráv. V rámci toho vznikla třída `Connector`, která implementuje obě potřebné funkcionality. Třída je také navržena jako singleton, což znamená že celý framework využívá pouze jednu společnou instanci této třídy. Tato vlastnost umožňuje snadné propojení signálů napříč celým systémem.

V ukázce kódu 2 můžeme vidět hlavní část implementace. Třída využívá pro ukládání zpráv signály z grafické knihovny a přidává funkcionalitu vysílání signálů z libovolného místa v kódu. Tato implementace je také bezpečná v rámci vícevláknového zpracování, kterého se využívá při vyhodnocování herních kol. Zároveň také nevyžaduje grafické rozhraní, které se v případě konzolového rozhraní vůbec neinicializuje. Jedinou podmínkou je zde statická inicializace signálů, protože signály z knihovny PyQt5 se musí připojit na rozhraní Qt C++ knihovny.

4.6 Rozšířitelnost projektu

Aby Orodael Turrim mohl být úspěšný, je zapotřebí zajistit jeho případnou rozšířitelnost a snadnou úpravu. Také je velice žádané, aby jednotlivé moduly frameworku byly znovupoužitelné například pro jiné hry či modifikaci původní hry. Rozšiřitelnost a modularita projektu patří mezi základní požadavky na systém.

4.6.1 Problém s opakujícím se zadáním

V případě programů, které se využívají ve výuce, je velice důležitá rozšířitelnost a snadná úprava. Pokud se při zadávání úkolů pro studenty každoročně používá naprosto stejné zadání, velice brzy se řešení problému objeví na veřejných fórech či jiných platformách. Z důvodu stále stejné domény znalostního systému má Orodael Turrim naprosto stejný problém, a je zapotřebí umožnění snadné modifikace zadání pro jednotlivé ročníky.

Tento problém Orodael Turrim částečně řeší velice jednoduše změnou herních objektů. Herní objekty a jejich vlastnosti velice ovlivňují výsledný znalostní systém a jeho systém plánování. Pokud se tedy například naprosto změní vlastnosti terénu, musí znalostní systém řešit naprosto odlišné problémy a znovupoužitelnost starších řešení se velice komplikuje. Terén se může změnit na tolik, že hráč již nebude hledat místo s nejlepší obranou ale pozici, která umožní jeho jednotkám alespoň přežít (například sopečná krajina). Pro tyto případy je součástí systému knihovna grafický podkladů pro mapu, která obsahuje přes 70 různých terénů.

Další možností je samozřejmě také změna jednotek, obránců či útočníků. Situace je zde velice podobná jako v případě terénu. Změnou jednotek či jejich atributů vzniká zcela odlišný problém, který musí znalostní systém řešit.

V neposlední řadě ke snadné modifikaci zadání přispívá modularita a obecnost celého frameworku. Zcela oddělen je zde modul pro práci se znalostním systémem. Aktuální implementace vyžaduje bázi znalostí definovanou na základě pravidel. Nicméně velice snadno lze tento modul vyměnit za implementaci využívající například rámce či graf.

Kombinací všech předešlých metod lze docílit natolik odlišného zadání, že znovupoužitelnost předchozích řešení je prakticky nulová. Zároveň lze vytvořit velké množství různých kombinací, pro jednotlivé ročníky.

4.6.2 Znovupoužitelnost jednotlivých modulů

Velkou výhodnou modulárnosti frameworku je také znovupoužitelnost jednotlivých modulů. Jednotlivé moduly jsou naprogramovány tak, aby je bylo možné využít samostatně. Tato vlastnost také umožňuje využití modulu pro zpracování znalostního systému jako část nové implementace. Velice snadno

se například dá znovupoužít pro jiné hry, které budou mít naprosto odlišná pravidla a žánr, ale budou ovládány znalostním systémem.

Další možností znovupoužitelnosti je samotný modul uživatele. Tento modul je navržen natolik obecně, že je možné jej velice snadno přesunout z role obránce na roli útočníka. Lze toho docílit pomocí pár změn napojení rozhraní a předefinování role hráče. Přesunutím modulu hráče z obránce na útočníka hra poskytuje velké množství nových možností a zadání pro studenty.

4.6.3 Možnost dalšího rozšíření

Orodael Turrim má také velký potenciál pro rozšíření své funkcionality. Cílem práce bylo vytvoření frameworku se základními moduly, které jsou nutné pro výuku znalostních systémů. Nicméně framework je připraven na případné rozšíření ve všech směrech.

Umělá inteligence

První velké rozšíření se týká umělé inteligence protivníků. Student ať již v roli útočníka či obránce potřebuje pro zkoušení svého znalostního systému protivníka. Pokud tento protivník bude dostatečně chytrý a dokáže se přizpůsobit různým situacím, bude pro studenty mnohem obtížnější vytvořit znalostní systém, který se nepříteli vyrovná. Silný protivník se především hodí, pokud studenti chtějí mezi sebou soupeřit a vyvinou ten nejlepší znalostní systém v ročníku. Umělou inteligenci je možné vytvořit jak pro roli obránce, tak útočníka. Umělá inteligence může být založena například také na znalostním systému nebo jiných pokročilých metodách, jako jsou například neuronové sítě.

Další velký potenciál frameworku je ve vytvoření sofistikované umělé inteligence, která by řídila útočníka. Jelikož je pro testování znalostního systému umělá inteligence důležitá, je v systému implementovaná základní umělá inteligence. Umělá inteligence pouze na základě náhodných hodnot posílá jednotky do pole. Aby se docílilo dostatečně silného protivníka, je zde jednoduchost umělé inteligence kompenzována zvýšeným množstvím peněz útočníka. To dříve či později vyústí k porážce obránce z důvodu přečíslení útočících jednotek. Nicméně na základě poskytovaných informací od herního enginu umělá inteligence může být mnohem sofistikovanější.

Webové rozhraní

Další velký potenciál pro rozšíření je vytvoření webového rozhraní pro automatické vyhodnocování implementací znalostních systémů. Jelikož se Orodael Turrim bude používat jako zadání úkolů velkého množství studentů, bylo by velice užitečné, alespoň částečně vyhodnocovala kvalitu řešení automaticky. Webové rozhraní může od studenta přijmout jeho implementaci modulu uživatele a automaticky vyhodnotit výsledek (počet přežitých kol) proti jednotlivým nepřátelům a na různých mapách. Tyto výsledky je pak například možné

zobrazovat v žebříčku nejlepších řešení, což velice podporuje soutěživého ducha. V neposlední řadě by také rozhraní mohlo ukládat průběhy veškerých bitev, kde by bylo možné získané informace využít například pro trénování neuronové sítě realizující umělou inteligenci.

Kapitola 5

Testování

Poslední kapitola se věnuje testování výsledného frameworku. Celý framework byl nejprve otestován pomocí jednotkových a integračních testů, které kontrolovaly základní funkcionalitu a uživatelské rozhraní. Následně byl framework podroben uživatelským testům. Kapitola se věnuje převážně postupu testování, problematickým částem a vyhodnocení testování.

5.1 Knihovny využité pro testování

Pro vytváření jednotkových a integračních testů existuje v Pythonu několik různých knihoven. Mezi nejznámější patří Unittest[20] a Pytest[21]. Unittest je ve vestavěná knihovna Pythonu pro testování. I když se jedná o vestavěnou knihovnu, oproti knihovně Pytest má značné nedostatky. V práci je využita knihovna Pytest kvůli hlavním výhodám, které velice usnadňují a zpřehledňují práci:

- využité vestavěné testování pomocí `assert`,
- libovolné strukturování testů,
- podpora parametrických testů,
- existence velkého množství rozšiřujících knihoven pro usnadnění práce,
- podpora napojení na další testovací frameworky.

Při testování se také využívá knihovna pro Pytest `flexmock`, která slouží pro snadné mockování testů. Mockování je metoda, při které testy upraví chování třídy či funkce tak, aby se chovala pro potřeby testování. Knihovna `flexmock` se využívá pouze u integračních testů k nasimulování herních situací, jako je nedostatek peněz nebo bitva jednotek s nízkým počtem životů.

5.2 Jednotkové testy

Jednotkové testy se zaměřují na testování jednotlivých funkcí a měly by fungovat nezávisle na ostatních třídách. Jejich účelem je odhalení chyb při implementaci před využitím funkcí v dalších metodách. Jednotkové testy vznikaly zároveň s frameworkem a v mnoha případech velice usnadnily hledání chyb.

5.2.1 Testování pozičních systémů

Velká část jednotkových testů se věnuje testování veškeré funkcionality pozičních systémů. Kvůli využití tří různých pozičních systémů vzniká při práci s frameworkem velké množství kombinací vstupních parametrů a přepočtů. Jednotkové testy v tomto případě hrály důležitou roli při vývoji veškerých metod. Díky testům bylo odhaleno velké množství chyb, které vznikaly pouze ve speciálních případech.

Z důvodu zrychlení celého frameworku se v průběhu vývoje celý poziční systém přepisoval na rychlejší způsob. Při tomto přepisu testy pozičního systému sloužily jednak pro kontrolu funkcionality, ale také se pomocí nich měřila míra zrychlení.

5.2.2 Testování jazyka pravidel

Druhá hlavní část jednotkových testů se věnuje testování jazyka pravidel. V případě pravidel se jednalo o vývoj řízený testy. Na základě definice gramatiky nejprve vznikl seznam všech výrazů, které gramatika podporovala. Testy těchto výrazů byly seřazeny podle složitosti, aby bylo možné iterativně rozšiřovat implementaci. Následně na základě testů vznikala implementace gramatiky v jazyce `ANTLR` a také stromové reprezentace pravidel. V tomto případě vývoj řízený testy velice urychlil celý proces implementace a zaručil kvalitu výsledného řešení.

5.3 Integrační testy

Integrační testy se věnují celkové funkcionalitě větších částí frameworku. V implementaci integrační testy ověřují funkcionalitu všech proxy, které používají studenti pro komunikaci s hlavním modulem. Druhá část integračních testů ověřuje podmínky, které musí splňovat umělá inteligence. Všechny testy včetně jednotkových testů pokrývají 70 % implementace.

5.3.1 Testování proxy rozhraní

První část integračních testů se věnuje testování všech proxy rozhraní, které využívají uživatelé pro ovládání hry. Testy jsou vytvořeny na základě pravidel,

která jsou nadefinovaná pro hru. Testy pokrývají veškeré omezení na poskytované informace a ověřují, zda se pomocí rozhraní nedají zjistit nepovolené informace. Převážně se testuje funkcionalita ze strany obránce, protože zde se dají očekávat případné pokusy o podvádění.

Testy na rozhraní vznikly na konci celého vývoje před odesláním frameworku na uživatelské testování. Testy objevily některé problematické části, které se převážně týkaly bezpečnosti. V některých případech bylo možné získat informace, které uživatel nesmí vědět. Veškeré chyby nalezené na základě testů byly odstraněny a následně byl framework rozeslán na uživatelské testování.

5.3.2 Testování umělé inteligence

Druhá část integračních testů se zaměřuje na ověření správné funkcionality umělé inteligence řídící útočníka. Jedná se převážně o otestování, zda umělá inteligence poskytuje pravdivé informace u nasazování jednotek pro příští kola. Z tohoto důvodu je v testech odehráno celé kolo hry. Po doběhnutí jednoho kola testy zkontrolují, zda umělá inteligence nasadila jednotky přesně tak, jak udala ve svém plánování.

Tato sada testů je implementována převážně z důvodu budoucího rozvoje aplikace. Pokud bude do frameworku přidán nový vylepšený modul umělé inteligence, lze snadno zkontrolovat, že splňuje všechny podmínky definované frameworkem.

5.4 Uživatelské testování

Důležitou části testování frameworku bylo uživatelské testování. Po dokončení implementace byl framework zpřístupněn veřejnosti pomocí portálu GitLab. Následně byli osloveni uživatelé, kteří by rádi aplikaci vyzkoušeli, a jako výsledek testování sepsali krátký report ohledně doporučených změn a nalezených chyb. Osloveni byli převážně studenti, kteří již předmět BI-ZNS absolvovali. Díky tomu měli dostatečné znalosti fungování znalostních systémů, a mohli tudíž vyzkoušet pokročilé funkce frameworku. Zároveň součástí reportu mohli napsat srovnání s předchozím frameworkem. Důležitý tester byla také cvičící předmětu BI-ZNS Ing. Klára Hájková, která v průběhu testování zohlednila veškeré potřeby, které budou při výuce předmětu využívány.

5.4.1 Testovací scénář

Testeři dostali doporučený scénář, který je provede celým testováním. Scénář byl rozdělen na jednotlivé body:

1. Přečtěte si přiloženou dokumentaci frameworku. Z dokumentace by měly být jasné veškeré herní principy, zprovoznění prostředí a práce s frameworkem.

2. Nainstalujte prostředí potřebné pro framework a spusťte aplikaci.
3. Spusťte grafické rozhraní frameworku a odehrajte jednu nebo více bitev pouze pomocí ovládacích prvků grafického prostředí.
4. Rozšiřte inferenční mechanismus o funkcionalitu nasazování jednotek.
5. Vytvořte vlastní útočný filtr a následně ho použijte při nasazení své jednotky.
6. Odsimulujte jednu nebo více bitev pomocí konzolového rozhraní.
7. Na základě informací z konzolového rozhraní odsimulujte stejnou bitvu v v grafickém rozhraní (stejná mapa, stejné chování nepřítele). Následně pomocí procházení historie zjistěte, kde byla největší slabina vaší obrany.
8. Vyzkoušejte nasazení více druhů jednotek a pokuste se porovnat, která z jednotek obránců je nejsilnější.

Testeři si měli během testování zapsat veškeré nalezené chyby nebo nedostatky frameworku a na konci testování vytvořit ucelený report s celkovým hodnocením aplikace.

5.4.2 Výsledky testování

Testování se zúčastnilo několik studentů, přičemž každý z nich se věnoval testování do různé hloubky a zaměřoval se na různé body scénáře. Na základě získaných reportů byl vytvořen seznam úprav, kterým následně byla přiřazena priorita. Závažné chyby, které brání ve vývoji frameworku, byly ihned opraveny, aby testeři mohli pokračovat v testování. Výsledky testování jsou strukturovány podle bodů scénáře, nikoliv podle uživatelů, kteří je našli.

1. bod scénáře

Ohledně dokumentace testeři nenalezli žádné závažné problémy. Dokumentace jim přišla srozumitelná a dobře členěná. V dokumentaci testeři nalezli poměrně velké množství překlepů. Na základě reportů a merge requestů do repositáře s frameworkem byly veškeré překlepy praveny.

2. bod scénáře

Největší problémy během testování se ukázaly při instalaci frameworku. Jelikož framework není vytvořen jako samostatně spustitelný balíček, testeři si museli doinstalovat potřebné prostředí pro běh frameworku. Všichni testeři během instalace prostředí narazili alespoň na jeden problém. V některých případech se problém týkal instalace Pythonu verze 3.7, v některých případech chybějící

Tabulka 5.1: Tabulka otestovaných operačních systémů

Operační systém	Verze	Prostředí Pythonu	Výsledek
Windows	10	Anaconda 3.7	Funkční
	7	Anaconda 3.7	Funkční
Ubuntu	18	Python 3.7	Funkční
	18	Anaconda 3.7	Funkční
OpenSUSE	15	Python 3.7	Funkční
	15	Anaconda 3.7	Funkční
Mint	9	Python 3.7	Funkční
	9	Anaconda 3.7	Funkční
Fedora	30	Python 3.7	Funkční

systémové knihovny pro fungování knihovny PyQt 5. Testeři pro spuštění frameworku použili různé operační systémy a různé prostředí pro běh Pythonu.

Aby se co nejvíce eliminovaly problémy při instalaci prostředí, byla dokumentace[13] rozšířena o detailní popis instalace prostředí pro různé operační systémy. Dokumentace se snaží pokrýt veškeré běžně používané operační systémy. Pro účel rozšíření dokumentace o instalační postupy a otestování funkčnosti, bylo vytvořeno několik virtuálních počítačů s různými operačními systémy. V každém operačním systému byla provedena instalace frameworku a na základě postupu byla vytvořena dokumentace. Seznam pokrytých operačních systémů se nachází v tabulce 5.1. Testování neodhalilo žádný běžně používaný operační systém, na kterém by nebylo možné framework spustit. V některých případech bylo zapotřebí doinstalovat dodatečné závislosti.

3. bod scénáře

Při testování grafického rozhraní aplikace byla nalezena jedna závažná chyba. Pokud uživatel spustil simulaci na více kol, aniž by nasadil svého krále, systém vytvořil pro každé kolo okno s informací o prohře. Navíc po dopočítání všech kol bylo stále možné ve hře pokračovat. Chyba vznikla na základě problému se synchronizací mezi vlákny, které vyhodnocují jednotlivá kola. Na základě reportů byla chyba opravena a vydána nové verze frameworku pro testery.

Na základě testování grafického prostředí vzniklo také velké množství podmětů na úpravu některých prvků. V případě jednoduchých úprav byly náměty ihned zapracovány. V případě složitějších úprav, které nemají zásadní vliv na hraní hry, byly zaznamenány do systému pro správu změn frameworku. Přehled všech reportovaných úprav se nachází v tabulce5.2.

4. bod scénáře

Hlavním cílem tohoto bodu bylo vyzkoušet, jak dobře se pracuje s modulem pro prací se znalostním systémem. Hodnocení testerů bylo vesměs pozitivní

Tabulka 5.2: Tabulka podmětů na změnu prvků v grafickém prostředí

Popis úpravy	Závažnost	Status
Číslo v závorkách u jména jednotky není snadno identifikovatelné. Přidat informaci, že se jedná o cenu jednotky.	Vysoká	Opraveno
Grafické prostředí neumožňuje restartovat bitvy bez vypnutí celého frameworku.	Střední	Neopraveno
Výraznější grafické rozlišení obránců a útočníků	Nízká	Neopraveno
Nevýrazná barva okraje při zobrazení příchozích jednotek	Střední	Opraveno
Přidání typů na veškeré elementy grafického rozhraní	Střední	Opraveno
Umožnění skoku do historie podle události v záznamu bitvy	Střední	Opraveno

a během testování nebyl nenalezen žádný zásadní problém. V reportech se nejčastěji objevovala zmínka o chybových hláškách. Podle testerů v některých případech nebyly chybové hlášky jednoznačné a dalo problém odhalit skutečný problém.

Na základě těchto reportů byl modul uživatele rozšířen o vrstvu, která některé systémové chyby doplňuje o detailní informace. Zároveň byl framework rozšířen o další 4 výjimky, které usnadňují odchytávání chyb uživatelem.

5. bod scénáře

Při vytváření útočných filtrů byla nalezena závažná chyba, která se týká grafického prostředí. Pokud uživatel vytvořil vlastní útočný filtry, který vyžadoval dodatečný parametr, nebylo možné tento filtr použít z grafického prostředí. Grafické prostředí neumožňovalo žádnou možnost zadání tohoto parametru a při použití vytvořeného filtru framework spadl s nejasnou chybou.

Tento problém byl opraven a do grafického prostředí byl přidán prvek, který umožňuje zadat požadované parametry v textové podobě. Filtry nyní navíc kontrolují, že vstupní parametry jsou pouze textového nebo číselného typu. Pokud by filtr podporoval i jiné parametry, mohlo by se jednat o bezpečností problém frameworku. V takovém případě by mohl uživatel do filtru přidat nepovolené akce.

6. bod scénáře

Během testování konzolového rozhraní nebyly objeveny žádné závažné problémy. Z reportů vyplynulo pár návrhů pro zlepšení, jako například přidání

možnosti exportu historie ve formátu XML. Všechny podměty od testerů byly zapracovány do frameworku.

7. bod scénáře

Se sedmým bodem scénáře měli téměř všichni testeři problém. V dokumentaci nenalezli žádnou zmínku o tom, jak by bylo možné odsimulovat bitvu z konzolového rozhraní v grafickém prostředí. Na základě intuice se pokoušeli úkol splnit pomocí exportování historie bitvy do formátu XML. Bohužel systém momentálně nepodporuje možnost načtení XML souboru. Nikoho z testerů nenapadlo využít hodnoty seedu generátoru mapy. Navíc po přiblížení tohoto postupu měli problém jak s nalezením seedu v konzolovém rozhraní, tak v nastavení seedu pro grafické rozhraní.

Na základě těchto problému vznikla nová sekce v dokumentaci, která se zabývá problematikou přenášení bitvy mezi konzolovým a grafickým prostředím. Je zde detailně popsán postup, jak lze docílit splnění sedmého bodu scénáře testování. Po přečtení dokumentace testeři již neměli žádný problém úkol splnit.

8. bod scénáře

Výsledky v tomto bodě scénáře se napříč uživateli částečně lišily. V rámci všech reportů však vyšla jako nejsilnější jednotka obránce Mág. Po důkladném porovnání na základě ostatních balančních kriterií byl mág částečně oslaben. Jeho útok byl snížen o 5.

Bohužel pro směrodatné testování sily všech jednotek bylo testerů příliš málo. Dodatečné balancování jednotek bude zapotřebí provést ještě jednou, až budou k dispozici odevzdané znalostní systémy od studentů předmětu. Pokud by se balancování provedlo na základě malého počtu vzorků, mohlo by se stát, že by balancování naopak zhoršilo vyrovnání jednotek. Toto by mohlo nastat například v případě hloupého znalostního systému, který by silné jednotky stavěl na naprosto nevhodná místa a nechráněná. V takovém případě by měření mohlo ukázat, že silná jednotka potřebuje ještě posílit.

Závěr

Cílem této diplomové práce bylo vytvoření frameworku pro výuku znalostních systémů v předmětu Znalostní systémy na Fakultě informačních technologií.

Hlavní částí vytvořeného frameworku je počítačová hra Orodael Turrim. Jedná se o hru žánru věžová obrana. Hra je jednak samostatně hratelná pomocí grafického rozhraní, ale hlavní přínos je vytvořené rozhraní pro ovládání pomocí znalostního systému. Orodael Turrim díky rozhraní tvoří ideální problém, na kterém je možné vyzkoušet vytváření znalostního systému.

Studenti si s pomocí frameworku vyzkouší, jak probíhá celý vývoj znalostního systému. Díky počítačové hře si studenti během práce vyzkouší, jak probíhá analýza cílové domény. Musí se seznámit s veškerými herními principy z dokumentace a následně díky hraní získat potřebné znalosti pro vytvoření báze znalostí. Následně za pomoci podpůrných modulů frameworku si vytvoří vlastní inferenční mechanismus, pomocí kterého můžou hru ovládat plně automaticky.

Díky modulárnosti celého systému má framework velký potenciál pro rozšiřitelnost. Největší potenciál se skrývá ve vytvoření webového rozhraní pro automatické opravování studentských řešení. Důležité bude také rozšíření herních možností systému, aby bylo možné vytvořit nové zadání práce pro příští běhy předmětu Znalostní systémy.

Literatura

[1] Dvořák, J.: Expertní systémy. 2004, [cit. 2019-04-16]. Dostupné z: http://www.uai.fme.vutbr.cz/~jdvorak/Opory/ExpertniSystemy.pdf

[2] Giarratano, J. C.; Riley, G.: *Expert systems*. Thomson Course Technology, čtvrté vydání, 2006.

[3] Jirina, M.: Znalostní systémy, soubor přednášek FIT ČVUT. 2018.

[4] Weitzman, L.; Hutchins, E. L.; Hollan, J. D.: STEAMER: An Interactive Inspectable Simulation-Based Training System. *AI Magazine*, 1984. Dostupné z: https://pdfs.semanticscholar.org/1081/e4cbebf3b3b881442b0a72dc47d590df8300.pdf

[5] Riley, G.: CLIPS: A Tool for Building Expert Systems. 2019, [cit. 2019-04-16]. Dostupné z: http://www.clipsrules.net/

[6] Microsoft: Age of Empires II HD - Age of Empires. 2019, [cit. 2019-04-30]. Dostupné z: https://www.ageofempires.com/games/aoeii/

[7] Entertainment, B.: Blizzard Entertainment:Warcraft III. 2019, [cit. 2019-04-30]. Dostupné z: http://us.blizzard.com/en-us/games/war3/

[8] Arts, E.: Plants vs Zombies Video Games. 2019, [cit. 2019-04-30]. Dostupné z: https://www.ea.com/studios/popcap/plants-vs-zombies

[9] Supercell: Clash Royale: Enter the Arena. 2019, [cit. 2019-04-30]. Dostupné z: https://clashroyale.com/

[10] Games, C.: Dungeon Defenders. 2019, [cit. 2019-04-30]. Dostupné z: https://forums.dungeondefenders.com/

[11] Patel, A.: Red Blob Games: Hexagonal Grids. 2019, [cit. 2019-04-19]. Dostupné z: https://www.redblobgames.com/grids/hexagons/

[12] Pecinovský, R.: *Návrhové vzory*. Computer Press, první vydání, 2007, 189 s.

[13] Horáček, J.: Orodael Turrim documentation. 2019, [cit. 2019-04-30]. Dostupné z: `https://orodaelturrim.readthedocs.io`

[14] Foundation, P. S.: What's New In Python 3.7. 2019, [cit. 2019-04-29]. Dostupné z: `https://docs.python.org/3/whatsnew/3.7.html`

[15] autorů, K.: Kivy: Cross-platform Python Framework for NUI. 2019, [cit. 2019-04-29]. Dostupné z: `https://kivy.org/`

[16] Computing, R.: Riverbank | Software | PyQt. 2018, [cit. 2019-04-29]. Dostupné z: `https://www.riverbankcomputing.com/software/pyqt/intro`

[17] Parr, T.: ANTLR. 2019, [cit. 2019-04-25]. Dostupné z: `https://www.antlr.org/`

[18] Fletcher, M.: RunSnakeRun Python (c)Profile Viewer. 2019, [cit. 2019-04-29]. Dostupné z: `http://www.vrplumber.com/programming/runsnakerun/`

[19] Volynets, N.: Visual profiler for Python. 2019, [cit. 2019-04-29]. Dostupné z: `https://github.com/nvdv/vprof`

[20] Foundation, P. S.: unittest — Unit testing framework. 2019, [cit. 2019-04-29]. Dostupné z: `https://docs.python.org/3/library/unittest.html`

[21] Krekel, H.: pytest: helps you write better programs. 2019, [cit. 2019-04-29]. Dostupné z: `https://docs.pytest.org/en/latest/`

Příloha A

Seznam použitých zkratek

GUI Graphical user interface

BI-ZNS bakalářský předmět Znalostní systémy

AoE Age of Empires

ZS Znalostní systém

UML Unified Modeling Language

XML Extensible Markup Language

HTML Hypertext Markup Language

LL Left Leftmost

Příloha **B**

Gramatika jazyka pravidel

$G = (N, \Sigma, P, S)$, kde :

N : { *rulesSet*, *singleRule*, *condition*, *conclusion*, *number*, *functionExpr*, *identifier*, *args*, *arg*, *operator*, *rFunctionExpr*, *numberEnd*, *identifierStart*, *identifierStart*, *identifierEnd* }

Σ : {;, *IF*, *THEN*, *WITH*, (,), *TRUE*, *FALSE*, [0-9], [*a*-*z*], [*A*-*Z*]}

S : *rulesSet*

P :

$$
\begin{aligned}
\mathit{rulesSet} &\rightarrow \mathit{singleRule}\ \mathit{ruleSet} \mid \varepsilon \\
\mathit{singleRule} &\rightarrow \mathit{IF}\ \mathit{condition}\ \mathit{THEN}\ \mathit{conclusion};\ \mid \\
&\quad\ \mathit{IF}\ \mathit{condition}\ \mathit{THEN}\ \mathit{conclusion}\ \mathit{WITH}\ \mathit{number}; \\
\mathit{condition} &\rightarrow \mathit{condition}\ \mathit{AND}\ \mathit{condition} \mid \\
&\quad\ \mathit{condition}\ \mathit{OR}\ \mathit{condition} \mid (\mathit{condition}) \\
&\quad\ \mathit{functionExpr} \mid \mathit{functionExpr}\ [\mathit{number}] \mid \\
\mathit{functionExpr} &\rightarrow \mathit{identifier}\ \mathit{args} \mid \\
&\quad\ \mathit{identifier}\ \mathit{args}\ \mathit{operator}\ \mathit{number} \mid \\
&\quad\ \mathit{identifier}\ \mathit{args}\ \mathit{operator}\ \mathit{identifier} \mid \\
&\quad\ \mathit{TRUE} \mid \mathit{FALSE} \\
\mathit{args} &\rightarrow \mathit{arg}\ \mathit{args} \mid \varepsilon \\
\mathit{arg} &\rightarrow \mathit{number} \mid \mathit{identifier} \\
\mathit{operator} &\rightarrow < \mid > \mid <= \mid >= \mid == \mid != \\
\mathit{conclusion} &\rightarrow \mathit{conclusion}\ \mathit{AND}\ \mathit{conclusion} \mid \\
&\quad\ (\ \mathit{conclusion}\ ) \mid \\
&\quad\ \mathit{rFunctionExpr}
\end{aligned}
$$

$$
\begin{aligned}
rFunctionExpr &\rightarrow identifier\ args\ | \\
&\quad\ identifier\ args\ :=\ number\ | \\
&\quad\ identifier\ args\ :=\ identifier \\
number &\rightarrow [0\text{-}9]\ numberEnd \\
numberEnd &\rightarrow [0\text{-}9]\ |\ \varepsilon \\
identifier &\rightarrow identifierStart\ identifierEnd \\
identifierStart &\rightarrow [a\text{-}z]\ |\ [A\text{-}Z]\ |\ \_ \\
identifierEnd &\rightarrow identifierStart\ end\ |\ [0\text{-}9]\ identifierEnd\ |\ \varepsilon
\end{aligned}
$$

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `zrojove-soubory`
  - `framework` ..... zdrojové kódy implementace
  - `text` ..... zdrojová forma práce ve formátu LaTeX
  - `dokumentace` ..... Dokumentace frameworku ve formátu HTML
- `text`
  - `prace.pdf` ..... text práce ve formátu PDF