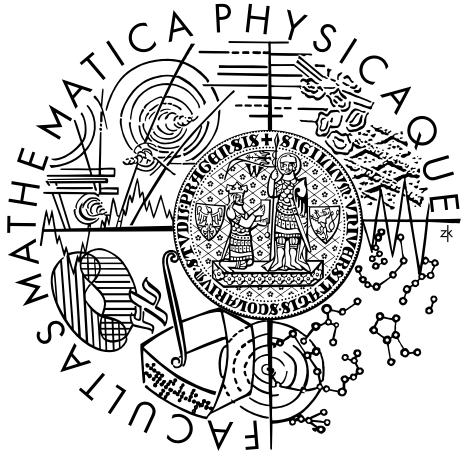
Univerzita Karlova v Praze

Matematicko-Fyzikální fakulta

**BAKALÁŘSKÁ PRÁCE**




Vojtěch Černý

# Umělá inteligence do hry Desktop dungeons

Katedra softwaru a výuky informatiky

Vedoucí bakalářské práce: Mgr. Filip Děchtěrenko

Studijní program: Informatika

Studijní obor: Obecná informatika

Praha 2015

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V ..... dne ..... podpis

Chtěl bych poděkovat především svému vedoucímu Filipu Děchtěrenkovi za možnost zpracovat zajímavé téma a za veškerou podporu. Dále bych rád poděkoval panu Dannymu Dayovi z QCFDesign za poskytnutí zdrojových kódů hry Desktop Dungeons, bez nichž by tato práce nemohla vzniknout.

Název práce: Umělá inteligence do hry Desktop dungeons

Autor: Vojtěch Černý

Katedra: Katedra softwaru a výuky informatiky

Vedoucí práce: Mgr. Filip Děchtěrenko, Katedra softwaru a výuky informatiky

Abstrakt: Počítačové hry žánru rogue-like jsou subžánrem počítačových RPG her pro jednoho hráče mezi jejichž znaky patří procedurálně generované prostředí a vysoká obtížnost. Uspět v nich je pro umělou inteligenci (AI) těžkou úlohou. Proto jsme implementovali framework pro tvorbu umělých hráčů pro rogue-like hru Desktop Dungeons. Rozebrali jsme možnosti přístupů k AI, a nakonec použili genetický algoritmus rekombinující hladové strategie. Výsledné AI hrálo hru na úrovni průměrného lidského hráče, vyhrávajíc v 72 % her. Tyto výsledky a framework mohou být použity ke zlepšení rogue-like her, procedurálních generátorů obsahu a umělé inteligence v podobných prostředích.

Klíčová slova: počítačové hry, umělá inteligence, evoluční algoritmy

Title: Artificial intelligence for the game Desktop dungeons

Author: Vojtěch Černý

Department: Department of Software and Computer Science Education

Supervisor: Mgr. Filip Děchtěrenko, Department of Software and Computer Science Education

Abstract: Rogue-like games are a subgenre of computer RPG games, featuring procedurally generated environment and permanent death. Winning them is a challenge for a human player, and more so for artificial intelligence (AI). In this work, we present a framework for implementing artificial players for a rogue-like game Desktop Dungeons. We then investigate options of suitable AI creation, and settle for using a genetic algorithm to fine-tune a greedy strategy. The resulting AI was as succesful as a mediocre human player, winning the game 72% of the time. This framework and results may be used to improve the quality of rogue-like games, procedural content generators, and artificial intelligence in similiar environments.

Keywords: computer games, artificial intelligence, evolutionary algorithms

# Obsah

# Úvod

Desktop Dungeons [1] je počítačová RPG hra pro jednoho hráče, která patří do užšího žánru rogue-like her. Cílem hry je vstoupit s hrdinou do jeskyně plné nestvůr, proklestit si cestu a nakonec porazit hlavního bosse (nejsilnější nestvůru). Úkol to není snadný a dobrému lidskému hráči se povede splnit jen jednou z řady pokusů. Jeskyně se totiž generuje částečně náhodně, a tak nelze vyhrát s jedinou, fixní strategií.

Vyvstává pak problém, že některé úrovně vůbec nelze vyhrát. Pokud je budeme hráči předkládat často, mohl by být frustrován touto nemožností a bezcílností jeho snažení. To je jedním z důvodů, proč má smysl vytvořit umělou inteligenci (AI), která hru hraje dobře, a lidskému hráči pak prezentovat pouze ty jeskyně, které tato inteligence vyhrát dokázala.

O tvorbu botů (umělých hráčů) pro jiné rogue-like hry se již pokusila řada lidí [2, 3, 4], někteří z nich úspěšně. Tyto boti byli zpravidla řízeni systémem ručně vytvořených pravidel. Ač je tento přístup efektivní, je neflexibilní, a samotný bot se spoléhá jen na expertní znalost jeho tvůrce. Přesná konfigurace takového bota obnáší hodně testování a ladění, které zabere hodně času.

K automatické konfiguraci a ladění můžeme výhodně použít evolučních algoritmů (EA) [5, 6]. Evoluční algoritmy jsou rodinou robustních prohledávajících algoritmů inspirovaných evolucí v přírodě, navržených především pro řešení těžkých problémů. Pro využití EA nám v podstatě stačí jen umět ohodnocovat kvalitu AI, ale nemusíme mít jasnou představu o jejím specifickém chování.

Evoluční algoritmy byly již pro umělou inteligenci v počítačových hrách použity, příkladem lze uvést práce toto úspěšně demonstrující v akční hře [7], ve strategii [8] a ve známé logické hře Tetris [9]. Dobrých výsledků bylo pomocí EA také dosaženo v klasických hrách jako jsou šachy, dáma nebo poker [10].

Cílem této práce je použít evoluční algoritmy pro vytvoření umělé inteligence hrající hru Desktop Dungeons. K tomu jsme implementovali framework pro návrh botů pro tuto hru, jelikož žádný takový dosud neexistuje.

## Struktura práce

V kapitole 1 si blíže vysvětlíme pojmy, se kterými budeme pracovat, především termíny umělé inteligence a evolučních algoritmů. Kapitolu 2 věnujeme popisu samotné hry Desktop Dungeons. Dále budeme pokračovat kapitolou 3, kde rozvedeme postup tvorby frameworku pro tvorbu botů. Přístupů k vytváření AI pro tuto hru je mnoho, významné z nich rozebereme v kapitole 4. V poslední kapitole 5 zhodnotíme návrh a výsledky našeho evolučního programu.

# 1. Příprava

Nejprve se seznámíme se základními pojmy umělé inteligence, rogue-like her a evolučních algoritmů.

## 1.1 Umělá inteligence

Umělá inteligence (AI, z angl. Artificial Inteligence) je obtížně uchopitelným pojmem. Vytvoření skutečně inteligentních chování se zdá býti velmi komplexní záležitostí, a proto byl obor AI rozčleněn do mnoha rozličných podoborů, od logického uvažování až po zpracování obrazu.

Odborné zdroje tento pojem definují jako "studium návrhu inteligentních agentů" [11, 12]. Z tohoto hlediska však narazíme na problematickou definici inteligence samotné [13]. Tomu se raději vyhněme, obdobně jako Russel s Norvigem [11], a místo inteligentních agentů zaveďme agenty racionální.

**Agentem** je cokoliv, co vnímá okolní prostředí prostřednictvím senzorů a ovlivňuje jej prostřednictvím akčních členů [11]. Jednoduchým agentem je tak například termostat, který vnímá prostředí senzorem teploty a ovlivňuje ho regulováním ústředního topení. Dalšími příklady může být robot v bludišti, klasický počítač, ale i člověk ve svém běžném životě.

Každý praktický agent má nějaký cíl či míru svého užitku, podle kterého můžeme usoudit, zda se chová dobře, nebo ne. Termostat má držet nastavenou teplotu, robot najít cestu z bludiště a člověk mít nějaký osobní užitek (který je subjektivní, mohou jím být peníze, úspěchy, kvalita života aj.). A tedy, **racionální agent** je agent, který vždy volí akci maximalizující jeho očekávaný užitek [11].

Tato definice je velmi obecná, a přitom značně omezující. Pro těžké (např. NP-úplné) problémy je velmi nepraktické volit nejlepší možnou akci, protože na její zjištění zpravidla nemáme čas. Často tak navrhujeme agenty, které sice nemůžeme podle původní definice považovat za racionální, ale jsou tím nejlepším, co zvládneme.

Z toho je zjevná důležitost experimentování s různými technikami v různých

prostředích, protože teorie nám neumožní zcela obecně upřednostnit jeden přístup před druhým a navrhnout dobře fungujícího agenta zvládneme jen díky pokusu a omylu.

## 1.2 Rogue-like hry

**Rogue-like** hry jsou velmi specifickým podžánrem počítačových RPG her. Hráč ovládá hrdinu, který vstupuje do prostředí hry (často podzemí nebo jeskyně) s nějakým obtížným cílem – zabití všech nestvůr, získání prastarého artefaktu apod.

Hlavními rysy rogue-like her jsou:

- Procedurálně generovaný obsah – každý běh má částečně náhodnou podobu.
- Permanentní smrt – absence možnosti hru ukládat a zpětně obnovovat, každý pokus je hrán znovu od začátku.
- Tahový systém – o každém kroku lze přemýšlet libovolně dlouho.
- Hra na mřížce – celý svět se skládá z čtvercových polí.
- Postupný vývoj hrdiny v obecném stylu role-playing her – hrdina se stává stále silnějším, získává lepší vybavení atd.

Svůj název podědily po prvním zástupci tohoto žánru, hře Rogue (1980). Od té doby byla vytvořena celá řada těchto her a některé ze starších, jako například NetHack (1987), jsou dosud vyvíjeny a hrány. V rámci navázání na tyto tradice je častým rysem všech rogue-like her taktéž minimalistická grafika, skládající se pouze z ASCII znaků. Některá moderní rogue-like již od této tradice upouštějí a raději nabízí hráčům vyšší komfort v plně grafickém prostředí.

Tento žánr také zažívá určitou renesanci v jím inspirovaných hrách [14], které přejímají jen vybrané mechaniky a jsou běžně označovány jako „roguelike-like“ nebo „rogue-lite“.

## 1.3 Evoluční a genetické algoritmy

**Evolučními algoritmy** (EA) [5] nazýváme obecně techniky hledání řešení založené na principu přirozené evoluce. Využívají přírodní mechaniky jako „silnější přežije“ k nalezení dobrého řešení. Nejčastěji používaným druhem EA jsou genetické algoritmy (GA), které si nyní obecně popíšeme. Detailněji rozebereme především ty části, které budeme dále v práci používat.

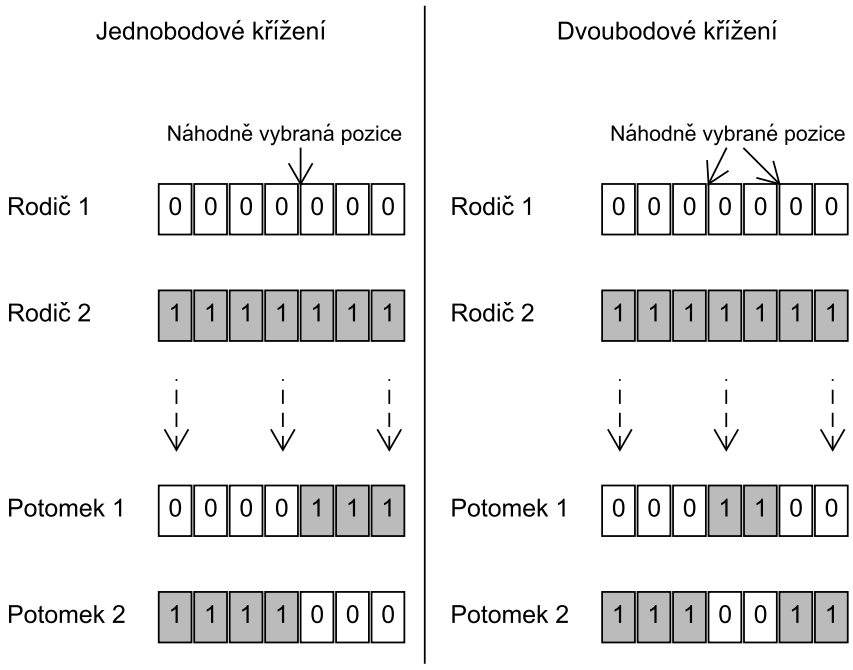
Genetický algoritmus pro své fungování využívá **populace**, která je skupinou **jedinců**. Každý jedinec má zakódované řešení problému ve svém **chromozómu**, který je posloupností **genů**. Chromozómy mívají většinou konstantní velikost. Gen obsahuje nějakou elementární hodnotu, může jí být prostý bit, ale často se používají celá čísla nebo čísla s plovoucí desetinnou čárkou. Každému jedinci se podle kvality v něm zakódovaného řešení přiřadí hodnota **fitness**.

Způsobů zakódování řešení jedince je mnoho, dokonce i pro jeden problém můžeme často sestrojit několik možností. Zpravidla má vybrání správné z nich velký vliv na výkon evoluce. Pokud například zvolíme možnost jak zakódovat jedince tak, aby nemohl obsahovat nepřípustná řešení, tak jsme pravděpodobně zmenšili prohledávaný prostor a tím celý GA zjednodušili.

Nad těmito strukturami se definují tři druhy operací, souhrně označované jako **genetické operátory**, a to **selekce**, **křížení** a **mutace**. Selekce může navíc být dvou druhů. **Rodičovská selekce** vybere z populace jedince, kteří budou sloužit jako základ následující populace. **Selekce prostředím** vybere z případného nadbytku jedinců část, která „přežije“. Pro oba typy lze využít stejné funkce. Klasickým příkladem je tzv. **ruletová selekce**, která vybírá jedince s pravděpodobností úměrnou jeho fitness.

Křížení je operace, která z jedinců – rodičů (zpravidla ze dvou) tvoří nové jedince – potomky. Mezi tradiční formy patří jedno-, dvou- a vícebodové křížení, pro jedince složené z reálných[1] čísel se také často používá křížení aritmetické. Jednobodové křížení vybere náhodnou pozici v jedinci a vytvoří dva potomky z rodičů tak, že zamění části za touto pozicí. Dvoubodové křížení funguje stejně jako dvě jednobodová křížení podle dvou různých náhodných pozic, a tak

[1] V počítači nelze reálná čísla přesně reprezentovat, v implementaci použijeme čísla s plovoucí desetinnou čárkou.




Obrázek 1.1: Ilustrace jednobodového a dvoubodového křížení.

umožňuje výměnu libovolné souvislé části chromozómu. Tato křížení jsou ilustrována na Obr. 1.1.

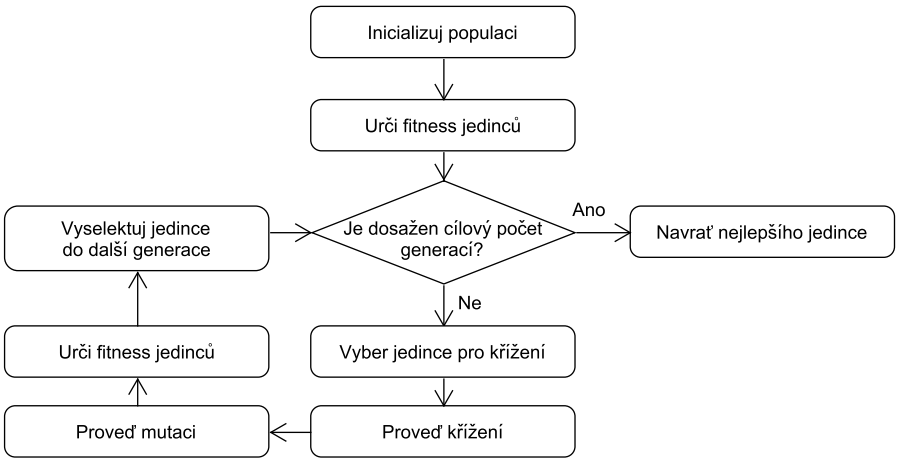
Aritmetické křížení vloží do genu potomka „konvexní kombinaci“ příslušných genů jeho rodičů, tedy hodnotu dle rovnice:

$$genPotomka = genA \cdot t + genB \cdot (1 - t) \qquad t \in [0, 1]$$

kde hodnoty $genA$ a $genB$ jsou hodnotami genu v rodičích, a $t$ je zvoleno náhodně z intervalu $[0, 1]$ a je využito pro celou operaci křížení. Druhého potomka lze případně získat opakováním operace s hodnotou $t' = 1 - t$.

Mutace je operací provedení malé změny v jedinci, jako například náhodné změny hodnoty jednoho genu. Pro zrychlení procesu evoluce může být použita tzv. **chytrá mutace**, která neprovádí změnu náhodně, ale sofistikovaněji. Může například prohledat blízká řešení a nejlepším z nich nahradit jedince, nebo nějak využít doménové znalosti.

Celý genetický algoritmus pak funguje následovně. Nejdříve se náhodně vytvoří populace jedinců. Poté N-krát (kde N je zamýšlený počet iterací) opakujeme rodičovskou selekci, samotné křížení, mutaci a selekci prostředím. Tento cyklus je vyobrazen na Obr. 1.2. Populaci jedinců v jedné iteraci nazýváme **generací**.




Obrázek 1.2: Základní schéma genetického algoritmu.

Výhodou genetických algoritmů je možnost jejich použití na široké spektrum problémů. Na otázku, které problémy jsou vhodné pro řešení genetickým algoritmem, neexistuje snadná odpověď. Pokud je však prostor řešení velký, podobná řešení mají podobnou fitness a neznáme pro tento problém specifický algoritmus, máme velkou šanci, že použití genetického algoritmu bude rozumným přístupem k hledání řešení.

# 2. Popis hry Desktop Dungeons

V této kapitole si vysvětlíme pravidla hry Desktop Dungeons, v dostatečné hloubce pro zdůvodnění našeho postupu při tvorbě AI. Kompletnější pravidla společně s tipy a triky pro hráče lze najít na Desktop Dungeons wiki [15].

Desktop Dungeons je počítačová RPG hra pro jednoho hráče, která patří do užšího žánru rogue-like. Mezi její charakteristiky tak patří vysoká obtížnost, procedurálně generované prostředí a permanentní smrt. Hráčem ovládaný hrdina vstupuje do jeskyně plné nebezpečných nestvůr s cílem zabít hlavního bosse (nejsilnější nestvůru). Úkol to není snadný a k jeho dosažení je třeba chytře využívat naskytnuvších se příležitostí.

Pro naší práci jsme využili alfa verzi hry. Plná verze je na rozdíl od ní placená a je její kompletní reimplementací. Nicméně se navzájem základními mechanizmy příliš neliší. Plná verze spíše nabízí hráčům hezčí grafiku, více příběhu a plynulejší zvyšování obtížnosti. Ani jednu z těchto nových vlastností však pro naši práci nepotřebujeme. Další výhodou alfa verze je možnost hraní v internetovém prohlížeči.

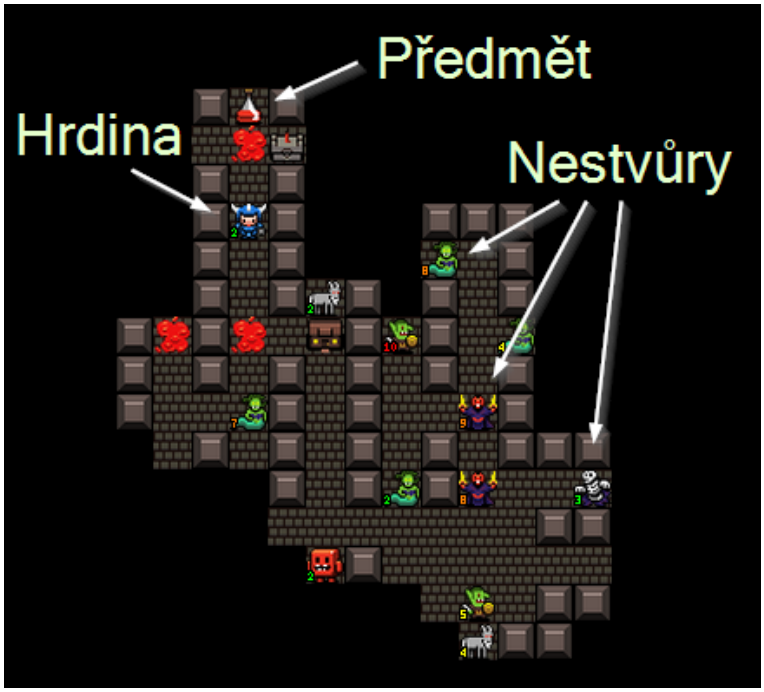
## 2.1 Hrdina

Hrdina je hráčem ovládaná postava v jeskyni. Provádí veškeré akce, které ovlivňují jeho vnitřní stav i zbytek jeskyně. Vnitřním stavem hrdiny je především jeho zdraví, mana, síla útoku, zkušenosti, sesbíraná kouzla a lektvary. Zdraví a síla se uplatní především při boji s nestvůrami, mana slouží k sesílání různých kouzel. Zkušenosti získá hrdina téměř výhradně poražením nestvůr.

Druhů akcí ve hře není mnoho. Hrdina prozkoumává nové kouty jeskyně, útočí na nestvůry, kouzlí a sbírá různé předměty.

## 2.2 Rasa a povolání

Před vstupem do jeskyně si hráč vybere rasu a povolání svého hrdiny. Mezi rasy patří možnosti jako trpaslík nebo elf, a povoláním může být třeba bojovník,




Obrázek 2.1: Screenshot jeskyně, obsahující hrdinu, nestvůry a předmět (lektvar zdraví). Černá pole okolo jsou dosud neprozkoumané části jeskyně.

kouzelník či kněz. Rasa pouze ovlivňuje, jaký bonus dostane hrdina za „konverzi“ glyfů kouzel (viz. 2.6), ale povolání může zcela jedinečně pozměnit pravidla hry. Hra obsahuje celkem 7 ras a 19 různých povolání.

## 2.3 Popis jeskyně

Jeskyni reprezentuje mřížka o rozloze $20 \times 20$ polí, přičemž na každém z nich je zpravidla jeden objekt, ale může jich být i více. Mezi tyto objekty patří hrdina, nestvůry, zdi a různé předměty. Kromě hrdiny jsou všechny tyto objekty nehybné[1]. Celá jeskyně však není v počátku hry odhalená, hráč vidí jen malé okolí svého hrdiny. Zbylý prostor je třeba odkrýt průzkumem. Ukázka jeskyně ze hry je na Obr. 2.1.

Samotná jeskyně je procedurálně generovaná, a tak je každá hra jiná a je třeba se umět dynamicky přizpůsobit.

[1] Některé vzácné efekty mohou pohnout např. nestvůrou, ale pro naše potřeby je lze zanedbat.

## 2.4 Průzkum

Pro prozkoumání libovolného pole stačí přijít hrdinou na pole sousední. Každé pole lze prozkoumat pouze jednou, odhalené zůstane již do konce hry. Průzkum ale navíc plní sekundární funkci. Za každé nově odhalené pole si hrdina regeneruje malou část zdraví a many. To znamená, že samotná jeskyně tvoří omezený zdroj, který je třeba využívat strategicky. Poznamenejme, že se při průzkumu obdobně léčí i nestvůry, a tak jej nelze dobře využít pro regeneraci v průběhu boje.

## 2.5 Boj

Boj ve hře Desktop Dungeons je poměrně statickou záležitostí. Nestvůry stojí na svých místech a nepohybují se. Hrdina může k některé z nich přijít a provést akci útoku. V tu chvíli proběhne výměna úderů. Útočník s vyšší úrovní (v případě shody nestvůra) zaútočí a ubere druhému účastníkovi souboje *přesně* tolik životů, jaká je jeho síla útoku. Druhý útočník, pokud ještě žije, útok stejným způsobem opětuje. Některé nestvůry i druhy hrdinů mohou mít zvláštní schopnosti, které toto schéma jemně pozměňují, mohou např. útočit vždy první či blokovat část útoku, ale tyto efekty jsou hráči předem známy.

## 2.6 Kouzla

Kouzla sesílá hrdina z tzv. glyfů kouzel, které lze nalést v jeskyni. Zpočátku hrdina žádný nevlastní a s sebou jich může nést jen omezené množství. Každý glyf umožňuje hrdinovi seslat jeden druh kouzla. Stačí zaplatit příslušné množství many, a případně vybrat cíl. Příkladem efektů uveďme kouzlo „Ohnivá koule“, které dokáže přímo zranit nestvůru (bez toho, aby vrátila úder), či „Úhyb“, které po seslání hrdinu ochrání před příštím smrtícím útokem. Oficiálně mají kouzla poněkud kryptické názvy, a proto budeme v této práci místo nich používat jejich významové ekvivalenty.

Kromě sesílání kouzel lze glyf i konvertovat. Konverze promění glyf v bonus určený podle rasy hrdiny. Tyto bonusy zahrnují malé zvýšení zdraví, many či lektvar navíc.

## 2.7 Předměty

Několik druhů předmětů lze najít přímo ležících v jeskyni. Tyto „power-upy“ hrdinovi drobně zvýší zdraví, manu, či sílu útoku. Lze také najít lektvary, které jsou dvou druhů, lektvar zdraví a lektvar many. Ty může hrdina využít kdykoliv, nejčastěji v průběhu souboje, a doplnit si tak významnou část zdraví/many. Množství těchto předmětů je pevně dáno a je v každé jeskyni stejné.

Další, speciální druhy předmětů lze koupit v obchodech, které se v jeskyni také vyskytují. Tyto můžou mít podobný efekt jako „power-upy“, nebo mohou hru ovlivnit zcela zvláštním způsobem.

## 2.8 Božstva

V každé hře lze najít několik oltářů, u kterých může hrdina začít vyznávat víru nějakého konkrétního boha. Tento bůh pak po zbytek hry hráče odměňuje, či trestá za jeho akce, a tím může velmi snadno změnit, které strategie fungují a které ne.

## 2.9 Shrnutí

Desktop Dungeons jsou ve své podstatě tradičním zástupcem žánru rogue-like, ale v mnoha hlediskách jsou velmi neobvyklé. Jedním takovým je omezení herního světa na pouhou jednu obrazovku o rozloze $20 \times 20$ polí. To zhušťuje důležitá rozhodnutí v průběhu hry a také zkracuje herní dobu jednoho pokusu na 10 - 15 minut. Dalším netradičním prvkem je determinističnost akcí. Pro každou akci je přesně definováno, jaký bude její výsledek [2], jedinou neznámou je tak náhodně generovaná jeskyně.

Spolu se zajímavou mechanikou regenerace při průzkumu se jedná o unikátní hru, balancující na pomezí RPG a logických her, která lidskému hráči nabízí příjemnou výzvu v krátkém časovém úseku.

Z hlediska klasifikace prostředí agenta, které přejímáme z knihy Russela a Norviga [11], si Desktop Dungeons stojí následujícím způsobem:

[2] Existují akce, které mají částečně náhodný výsledek, ale je jich velmi málo a vyžadují specifické nastavení hry, které nebudeme používat.

- **Plně/částečně pozorovatelné**: Prostředí je částečně pozorovatelné, neb neznáme neprozkoumané části jeskyně.
- **Jedno-/více-agentní**: Naše prostředí můžeme bezpečně považovat za jedno-agentní. Nestvůry v jeskyni můžeme považovat za okolní prostředí, protože reagují přesně daným a jednoduchým způsobem.
- **Deterministické/stochastické**: Všechny akce jsou deterministické.
- **Epizodické/sekvenční**: Na pořadí akcí záleží, prostředí je tedy sekvenční.
- **Statické/dynamické**: Prostředí se samo nijak nemění, je tedy statické.
- **Diskrétní/spojité**: Množina stavů je diskrétní.
- **Známe/neznámé**: Všechna pravidla hry jsou nám známa.

Všimněme si, že kromě částečné pozorovatelnosti, jsme z hlediska klasifikace získali nejjednodušší možné prostředí. Právě ale částečná pozorovatelnost společně s nezanedbatelnou délkou hry dělají z Desktop Dungeons hru obtížnou.

# 3. Návrh Java Frameworku

Prvním krokem pro tvorbu AI pro Desktop Dungeons je vytvoření prostředků pro ovládání hry programem. Řešení tohoto problému uvedeme v této kapitole.

Desktop Dungeons jsou pro AI hrou velmi zajímavou. Lze uplatnit mnoho různých technik, v potaz přichází použití technik klasického prohledávání, plánování, evolučních algoritmů a mnohých jiných.

Pro možnosti budoucího experimentování jsme se rozhodli vytvořit framework pro tvorbu botů (umělých hráčů). Tento poslouží jako prostředník mezi ovládáním hry a samotnou logikou AI. Pro jeho realizaci bylo potřeba nejprve navrhnout způsob přístupu do hry.

K dispozici jsme měli dvě verze samotné hry, jednu psanou v prostředí GameMaker [16] a druhou v JavaScriptu [17], který běží v internetovém prohlížeči. Pro práci jsme nakonec použili JavaScript verzi, se kterou lze lépe pracovat konvenčními způsoby (zdrojové kódy mají textovou podobu), a také ji můžeme snadno spustit na různých operačních systémech.

## 3.1 Vytvoření API

Pro interakci se hrou psanou v JavaScriptu existuje několik možných přístupů:

- Psaní botů v jazyce JavaScript a interakce se hrou ve stejném kódu. Toto řešení je velmi přímočaré a principiálně postrádá náklady na režii. Skriptovací jazyk JavaScript však nemusí všem programátorům vyhovovat.

- Čtení grafických dat z obrazovky a emulace používání myši pro hráčské ovládání hry. Výhodou tohoto řešení je absence nutnosti modifikace zdrojového kódu hry, ale celé řešení je poněkud těžkopádné.

- Vytvoření frameworku v jazyce Java, který poběží jako applet přímo v prohlížeči, ve stejném okně jako hra. Tento applet pak dokáže komunikovat se zbytkem stránky [18]. Obdobná řešení lze realizovat i pomocí některých jiných technologií, které běží v prohlížeči.

- Využití HTML protokolů GET/POST a standardních postupů dynamického získávání informací webové stránky od námi vytvořeného serveru.
- Vytvoření lokálního (nebo vzdáleného) síťového spojení mezi hrou a frameworkem.

Po rozvaze a krátkém testování jsme se rozhodli použít klasický síťový protokol, po kterém budeme posílat textové zprávy. Tato možnost nám dává nejvíce flexibility v návrhu, uživatelská aplikace může být psána v téměř jakémkoliv jazyce a textová komunikace zjednoduší ladění chyb v programu. Stejné řešení přitom používá platforma Pogamut [19], která slouží pro návrh AI v akční hře Unreal Tournament 2004.

Při tomto řešení jsme využili příchodu HTML5, při kterém byl navržen protokol WebSocket [20], který realizuje plně funkční, obousměrné TCP spojení z JavaScriptu. Hlavní prohlížeče jako Google Chrome či Mozilla Firefox tento protokol podporují.

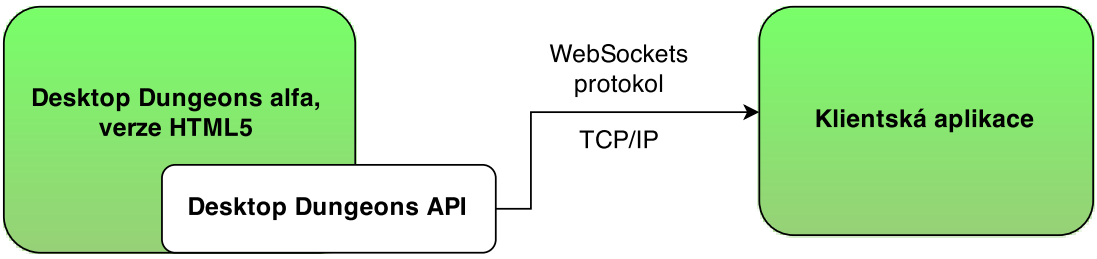
Modifikovanou část zdrojového kódu hry, která se stará o toto spojení, budeme nazývat Desktop Dungeons API, nebo jen zkráceně API. V rámci specifikace protokolu WebSocket je v JavaScriptu implementována pouze klientská strana protokolu. To pouze znamená, že API může spojení jen navazovat, nikoli přijímat, i když pro naše využití by opačný směr byl více logický. Nikterak nás to ale neomezuje, po navázání spojení jsou již obě strany ekvivalentní a mohou komunikovat libovolně.

Schéma propojení je znázorněno na Obr. 3.1, WebSocket protokol je popsaný v příslušném RFC [20] a popis přes něj zasílaných zpráv je v přiložené dokumentaci (viz. Příloha A).

## 3.2 Java Framework

Po vyřešení způsobu připojení ke hře jsme mohli začít tvořit framework, ve kterém lze psát AI. Pro tento framework jsme se rozhodli použít jazyk Java [21], který je populární, uživatelsky přívětivý a společně s JS verzí hry multiplatformní.

Požadavky na framework se různí podle toho, k čemu jej budou uživatelé chtít využít. Nicméně, protože má hra deterministické akce, můžeme předpokládat, že




Obrázek 3.1: Schéma propojení hry a klientské aplikace (například našeho frameworku). Desktop Dungeons API je modifikovaná část hry, která se připojí ke klientské aplikaci a poté interpretuje a odpovídá na zprávy.

prohledávání stavů hry bude využíváno hojně. Navrhneme tedy obecný přístup ke hře, s přihlédnutím k prohledávání.

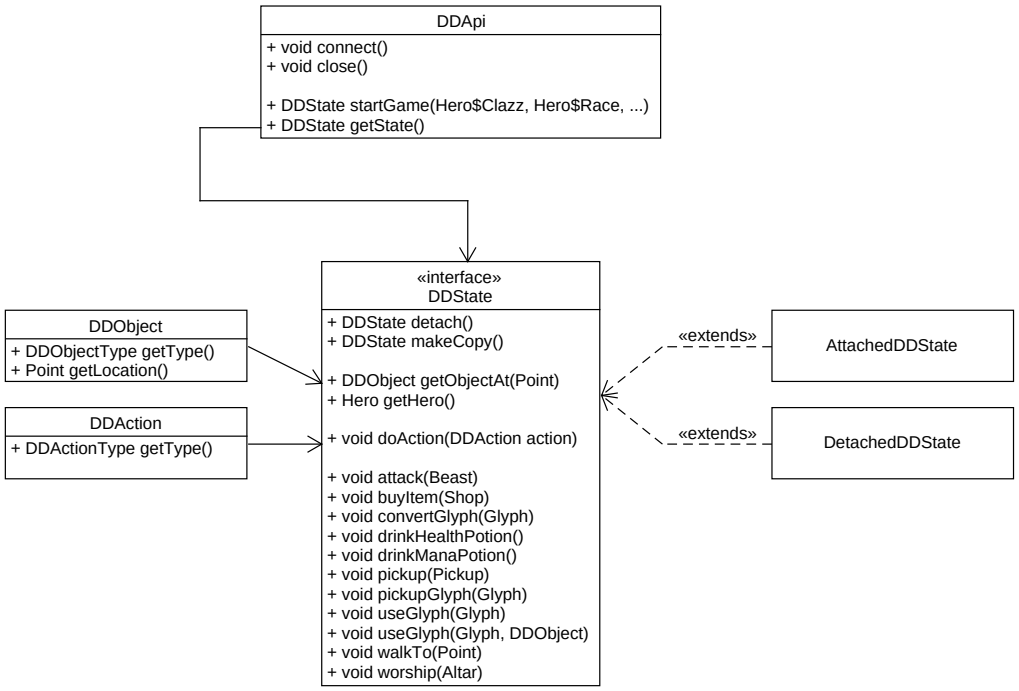
Pro komunikaci přes protokol WebSocket jsme vytvořili vlastní, jednoduchou knihovnu.

Základem frameworku je třída `DDApi`, obsahující metodu `connect()`, která se pokusí o připojení k JavaScriptové verzi hry s API. Přes tuto třídu je také možno komunikovat některé základní příkazy, jako například spuštění hry s určitým hrdinou.

Jakmile je hra spuštěna, středobodem dění se stává objekt stavu hry, zvaný `DDState`. Ten obsahuje kompletní informace o stavu jeskyně, včetně informací o hrdinovi a jiných. Na stavu hry je možno vyvolávat všechny možné herní akce, jako je pohyb, průzkum, útok atd.

Objekt stavu hry může být v jednom ze dvou stavů: připojený (attached) a odpojený (detached). Připojený stav hry přeposílá všechny provedené akce přímo do hry samotné, které se v ní okamžitě promítnou. Slouží tak k přímému ovládání hry. Odpojený stav je určen pro techniky prohledávání, akce se pouze „odsimulují“ v tomto stavu.

Po spuštění hry existuje pouze jeden připojený stav. Ten lze z objektu `DDApi` získat pomocí metody `getState()`. Připojený stav nelze ručně vytvořit, ani zničit, a bude tedy po celou dobu hry existovat právě jeden. Odpojené stavy lze odvodit z tohoto stavu zavoláním metody `detach()`. Výsledkem bude nový, odpojený stav, který je kopií připojeného.




Obrázek 3.2: Diagram klíčových objektů a metod našeho frameworku.

Schéma zachycující strukturu hlavních objektů je na Obr. 3.2.

Nyní si probereme některé řešené problémy a funkcionalitu nad rámec základního chování frameworku.

## Simulace hry

Pro správné fungování akcí na odpojených stavech bylo třeba najít způsob, jak hru simulovat, tedy provádět akce tak, aby nebyly aplikovány v samotné hře. Možností řešení bylo jen pár: podpořit funkcionalitu odpojených stavů i na straně API nebo reimplementovat mechanismy hry na straně frameworku.

Zvolili jsme druhou možnost. Ač pracnější a poměrně špatně odladitelná, slibuje větší rychlost, celou simulaci provede framework a žádná data při ní nebudou téci síťovým spojením. Navíc může být použití simulace více otevřeno uživateli.

Toho jsme dále využili, kvůli rychlosti jsme zavedli metody simulace souboje bez nutnosti kopírování celého stavu hry. Jsou to především metody na objektu typu `Hero` – `performAttack()` a `performFireball()`.

Vzhledem k rozsáhlosti hry nebyla simulace implementována kompletně, nýbrž pouze v rozsahu, který využijeme pro zbytek práce. Přesné vymezení tohoto roz-

sahu je uvedeno v Příloze B. V případě použití některé z neimplementovaných vlastností hry se mohou výsledky simulovaných akcí lišit od reálných.

### Průzkum

Průzkum je důležitou částí hry a probíhá vždy, když se hráč přiblíží na dotek k neprozkoumanému poli. Jakýkoliv pohyb, který neprozkoumá pole, je v podstatě nulovou akcí, ve stavu hry neprovede žádnou změnu (kromě nové pozice hrdiny). Uživatel si však nemusí průzkumné tahy přepočítávat sám, seznam těchto míst lze získat pomocí metody `getExplorationMoves()` na objektu `DDState`.

Důležité je také zmínit, jak funguje průzkum v odpojených stavech. Hráč neví, co se skrývá v neprozkoumaných částech, tedy simulace neví, co odhalit. Tento problém jsme vyřešili tak, že simulace vždy odhalí nejhorší možnou variantu – zeď. Ta blokuje hrdinův postup, a tím zaručíme, že posloupnost akcí provedená v simulaci bude vždy proveditelná i v samotné hře.

### Obecný interface akcí

Pro použití metod prohledávání a různého experimentování je vhodné vědět, které akce jsou proveditelné v daném stavu hry. Toto je podporováno pomocí obecného objektu `DDAction`, který popisuje akci. Ze stavu lze všechny proveditelné akce získat metodou `getActions()` a provádět je posléze můžeme pomocí metody `doAction()`.

### Modifikace prostředí hry

Svět Desktop Dungeons je velmi komplexní, obsahuje mnoho typů nestvůr, kouzel atp. Pro vytváření AI může být výhodné tuto doménu omezit a věnovat se pouze nějaké její jednodušší variantě. Z tohoto důvodu obsahuje framework metody `setGeneratedBeasts()` a `setDungeon()`, které nastaví druhy nestvůr, které se v jeskyni mohou objevovat, resp. nastaví přímo rozložení jeskyně. To lze využít až k implementaci vlastního procedurálního generátoru prostředí a jeho použití v našem frameworku.

Pro zjednodušení technik prohledávání umožňuje náš framework také variantu řešení plně odhalené verze hry. Pak nemusí AI řešit problémy s částečnou

pozorovatelností. Stačí při spuštění hry pomocí metody `startGame()` nastavit parametr `cheatingVision`.

Díky těmto možnostem modifikace je možné využít framework k řešení řady podproblémů hry nebo si vytvořit i vlastní. Lze z hry extrahovat generované úrovně, ty pozměňovat a posléze vracet, čímž jsme schopni měnit obtížnost hry, zapínat a vypínat jednotlivé mechaniky, jako jsou určitá kouzla, božstva a podobně. Můžeme tak hru dostat až do stavu, který je vhodný pro přímé učení agenta v neznámém prostředí. Také můžeme odebrat stěny, aby jeskyně nepřipomínala bludiště, a snažit se hru řešit v této „otevřené“ variantě. V ní se můžeme pokoušet optimalizovat pořadí zabíjení monster, pevné používání lektvarů a podobně.

Podrobná dokumentace frameworku je součástí elektronické přílohy (viz. Příloha A).

# 4. AI pro Desktop Dungeons

V této kapitole rozebereme možné přístupy k tvorbě AI pro Desktop Dungeons a zdůvodníme výběr toho našeho.

Seznam přístupů k AI je velký a s dostatečnou snahou by většina z nich šla použít pro hraní Desktop Dungeons. Zde si vyjmenujeme a rozvedeme námi zvažované z nich. Uveďme je postupně od jednodušších po složitější.

## 4.1 Náhodné akce

Jako první pokus o AI lze často použít bota (umělého hráče) provádějícího náhodné akce. Seznam možných akcí v Desktop Dungeons je konečný, a proto můžeme snadno jednu z nich náhodně vybrat.

Tento přístup však naráží velmi brzy – v jeskyni téměř okamžitě objevíme silné nestvůry, kterých bychom si neměli v této fázi hry vůbec všímat. Ale náš náhodný bot na takovou nestvůru velmi brzo narazí a zemře při prvním pokusu o útok.

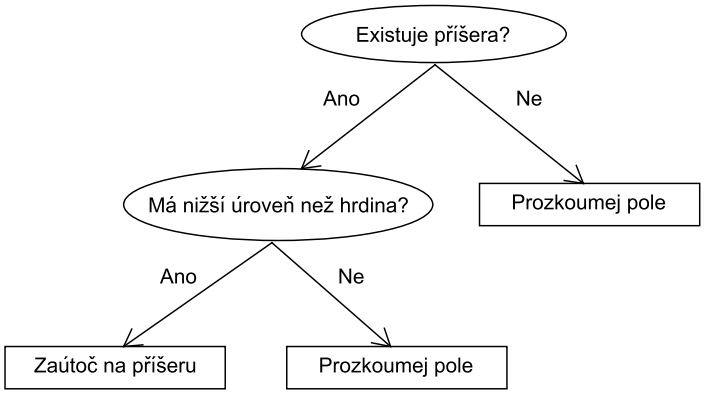
Máme sice k dispozici simulaci a mohli bychom tak z akcí vynechat ty, které vedou přímo k smrti a prohře. Nicméně i tak provádí bot velmi nesmyslné akce, útočí na nestvůry, které jej těžce zraní, a má problémy i se základními principy hry, jako zaútočit na stejnou nestvůru vícekrát v řadě.

Náhodné akce se tedy nezdají příliš vhodné.

## 4.2 Rozhodovací pravidla

Další přímočarou AI je vytvoření tzv. **reflexního agenta** [11, Kap. 2], který vybírá akci pouze na základě aktuálního vjemu, tj. v našem případě celé pozorovatelné části stavu. Herních stavů je však obrovské množství, a proto nelze pro každou z nich agentovi nastavit vhodnou akci. Místo toho vytvoříme jednoduchou funkci nebo program, který takovou akci zvolí.

Pokud konstrukci programu nijak neomezíme, řešíme stále stejný obecný problém vytvoření AI. Jednoduchým takovým omezením je použití pouze progra-




Obrázek 4.1: Jednoduchá inteligence zaznamenaná pomocí rozhodovacího stromu.

mových **if-then pravidel** nad množinou primitiv, např. vybraných aspektů stavu, jako je počet nestvůr, zdraví hrdiny atp. Využíváme tak metodologie, že pro správné komplexní rozhodování agent nepotřebuje v každém kroku zohlednit kompletní stav světa [22]. If-then pravidla můžeme zaznamenat pomocí rozhodovacího stromu. Jednoduché AI používající tyto konstrukce demonstrujeme na Obr. 4.1.

Jakkoli se tento návrh zdá elementární, lze pomocí něj reprezentovat komplexní strategie, které dobře postupují pro dosahování cílů. Rozšířením tohoto přístupu jsou behaviorální stromy [23], které se používají pro AI v moderních hrách.

Výhodou tohoto přístupu k AI je vysoká rychlost jeho provádění a lidská uchopitelnost struktury rozhodování. Nevýhodou je obtížné vytváření vhodných rozhodovacích pravidel pro komplikovanější domény.

## 4.3 Prohledávání

Determinističnost všech akcí ve hře silně poukazuje na možnost využití **prohledávání** [11, Kap. 3]. Nad tímto nápadem jsme dlouho spekulovali, na jeho implementaci však nedošlo. Jsou zde totiž i fakta, která od prohledávání odrazují. Větvící faktor (počet možných akcí ve stavu) totiž často přesahuje deset a celkový počet akcí pro dohrání hry je v řádu stovek, tedy přímočaré prohledávání není možné použít kvůli jeho rozsáhlosti. Dalším problémem je neprozkoumané

prostředí. Tvar jeskyně nejsme schopni předem odhadnout, a tak nevíme, které akce budeme chtít využít poté, co prozkoumáme nějakou novou část.

Řešením obou problémů je užití **heuristické funkce** [11, Kap. 3.6], která ohodnocuje stavy, a tak algoritmus preferuje slibně vypadající akce, čímž můžeme problém značně zjednodušit. Pak ale často přehlédneme chytrý tah v akci, která na první pohled slibně nevypadá.

Prohledávání jako takové tedy stojí a padá s návrhem heuristické funkce. Výhodou tohoto přístupu je ovšem jistá obecnost, algoritmus zvažuje použití všech možných akcí při cestě za cílem.

## 4.4 Plánování

Plánování [24] je technika hledání sekvence akcí, které vedou z počátečního stavu do stavu cílového. Od obecného prohledávání se liší daným způsobem popisu problému (v praxi se nejčastěji používá PDDL [25]). Takto zaznamenaný problém pak lze předložit obecnému plánovači a ten se pokusí najít řešení.

Výhodou je pak fakt, že plánování má bohatou historii, moderní plánovače jsou efektivní, ozkoušené a obsahují dobré doménově nezávislé heuristiky.

Na obtíž však stále zůstane velikost problému, která nemusí být bez použití doménově závislých heuristik vůbec uchopitelná. Také bychom si museli nějak poradit s neprozkoumanou částí jeskyně, neboť plánování často předpokládá, že možné akce jsou vždy známy. A jako další negativum, zapsání poměrně komplexního světa Desktop Dungeons do PDDL by bylo malou výzvou samo o sobě.

## 4.5 Hladový přístup

Konceptuálně velmi jednoduchým přístupem ke hře Desktop Dungeons jsou hladové strategie stylu „zkus zabít co nejsilnější nestvůru, jinak prozkoumej“. Prozkoumávání totiž vyčerpává v podstatě jediný zdroj regenerace a čím méně prozkoumáváme zpočátku, tím více se můžeme regenerovat v pozdějších fázích hry.

Problémem je, že zabití příšery není elementární akcí, ale je potřeba jejich sekvence. Tento problém se dá však opět řešit hladově. Můžeme simulovat prosté útoky a zjistit, zda nakonec přežije hráč nebo nestvůra. To sice nebude optimální

způsob — zabíjení nestvůr je v podstatě složitou hádankou — ale může to být relativně dobrou a jednoduchou strategií.

Všimněme si, že samotný hladový algoritmus je sadou rozhodovacích pravidel. Ty mají pouze jednotný tvar „pokud lze provést $A$, proveď $A$“.

Tento přístup jsme také implementovali. Avšak abychom lépe využívali možnosti hry, zesložitili jsme jej. Základní hladovou strategií tvořily tyto tři jednoduché strategie, seřazené podle priority.

1. Zabij co nejsilnější nestvůru.
2. Seber náhodný předmět.
3. Prozkoumej náhodné pole.

Ze všech strategií, které byly ve stavu hry proveditelné, se provedla nejvýše umístěná v seznamu.

Zabíjení nestvůr jsme řešili také hladově, ale komplikovaněji než prostým cyklickým zkoušením útoků. Akce zkoušené na nestvůry lze vypsat obdobným seznamem:

1. Použij kouzlo „Ohnivá koule“ k útoku na nestvůru.
2. Zaútoč na nestvůru běžným způsobem.
3. Vypij nějaký lektvar.

Zde jsme opět simulovali používání proveditelné akce s co nejvyšší prioritou, dokud neumřela nestvůra nebo hrdina. V případě úspěšné sady akcí jsme tyto provedli přímo ve hře. Jako vylepšení jsme této hladové AI v souboji proti nestvůrám nízkých úrovní omezili používání lektvarů, protože jsou příliš cenné a vyplácejí se používat až v pozdějších fází hry.

Tento přístup fungoval až překvapivě dobře. Bot vcelku často dokázal vyčistit jeskyni od všech nestvůr kromě finálního bosse. Když jsme ho nechali běžet dostatečně dlouho (dny času, desetitisíce běhů), dokonce se několikrát stalo, že hru přímo vyhrál. Potřebuje k tomu ale velmi příznivě vygenerovanou jeskyni, kde potkává právě včas nestvůry, které je schopen zabíjet.

Všimněme si, že tento hladový algoritmus kombinuje použití předchozích přístupů, konkrétně používáme specifický druh if-then pravidel a jednoduché prohledávání.

Hladové algoritmy jsou tedy vcelku dobrou a jednoduchou AI. Jejich slabinami jsou především přílišná chamtivost (někdy je lepší odložit souboj na čas, kdy při něm nespotřebujeme tolik zdrojů) a v podstatě absence dlouhodobého plánu.

## 4.6 Evoluční algoritmy

Evoluční algoritmy (EA) jsou rodina robustních prohledávacích algoritmů inspirovaných přírodou, použitelných na široké spektrum problémů. Genetické algoritmy (GA) jsou nejpoužívanějším druhem EA, které reprezentují řešení jako posloupnost genů, a posléze na ně aplikují operace jako selekci, křížení a mutaci. Podrobněji jsme si je popsali v kapitole 1.3.

Na doménu Desktop Dungeons lze evoluci použít mnoha způsoby. V podstatě bychom mohli pomocí EA zlepšovat libovolné z dosud navržených řešení.

Náhodným akcím bychom mohli udělit váhy, s jakou pravděpodobností budou prováděny, a tyto evolučně křížit. Protože však náhodné akce byly samy o sobě poměrně špatným přístupem, není jeho zlepšování příliš slibnou cestou.

Zobecněním evoluce rozhodovacích pravidel je tzv. **genetické programování** [26], které slouží k evoluci stromových struktur, kterými lze zapsat počítačové programy, matematické vzorce nebo námi popsaná if-then pravidla. Obtížnou částí návrhu genetického programu je poskytnout správné stavební bloky pro jeho funkčnost. Bude-li jich málo, omezíme tím možnosti, kterých můžeme dosáhnout, bude-li jich příliš mnoho, evoluce bude komplikovaná a nedoběhne v praktickém čase.

Při prohledávání lze evolvovat heuristickou funkci. Ohodnocení stavu běžně závisí na jeho částech, jako je počet zabitých nestvůr, odhalených polí či například zdraví hrdiny. Je ale těžké odhadnout, jakou vahou přispívá který člen. To by ovšem opět mohl nastavit EA podle čistě praktického hlediska. Obdobný způsob úspěšně použilo např. AI pro hru Tetris [9].

Hladový přístup lze také snadno evolvovat. V tomto přístupu jsme použili akce

seřazené podle priority, kterou jsme odhadli na základě doménových znalostí, tj. zkušeností se hrou. EA by mohl tyto priority a případně i jejich použití nastavit sám, bez jakýchkoliv lidských odhadů a pouze z hlediska toho, co opravdu funguje.

Kombinace použití evolučních algoritmů a plánování je již více teoretická či experimentální, ale obdobně jako při prohledávání bychom mohli evolvovat heuristické funkce.

To ale nejsou jediné možnosti, jak můžeme využít evoluci pro AI v Desktop Dungeons. Mohli bychom využít toho, že EA jsou ve své podstatě prohledávající algoritmus a my máme prohledáváním řešitelný problém. Tedy bychom mohli do choromozómu jedince zakódovat sekvenci akcí, a ty mezi sebou křížit a mutovat, dokud bychom nedošli k řešení konkrétní jeskyně.

Jak je vidět, evoluční algoritmy jsou silným nástrojem, který lze použít mnoha různými způsoby. Nevýhodou může být dlouhá doba běhu a těžké rozhodování, kterou variantu a jak použít.

## Rozhodnutí

Pro naši práci jsme se rozhodli použít genetický algoritmus, kterým zoptimalizujeme hladové strategie. Problém s přílišným využíváním zdrojů vyřešíme obdobně jako v ručně navrženém hladovém algoritmu – pro každou akci budou existovat podmínky, za kterých ji lze využít, které také nastaví GA.

Oproti technikám prohledávání má tento návrh výhodu, že vytvoří AI, která se snaží jeskyně řešit univerzálně, tedy po skončení evoluce budeme mít algoritmus, který hraje hru rychle. Navíc si tento přístup najde správné techniky sám a není ovlivněn naším intuitivním přístupem k hraní hry.

Podrobnosti implementace rozebereme v následující kapitole.

# 5. Použití genetických algoritmů

Pro návrh AI hrající Desktop Dungeons jsme se rozhodli použít genetický algoritmus (GA). V této kapitole si představíme návrh našeho GA a jeho výsledky. Genetické algoritmy obecně jsme již vysvětlili v kapitole 1.3.

Pro implementaci genetických algoritmů v jazyce Java jsme využili knihovnu JGAP [27]. Pro drobnou práci s XML pro potřeby persistentního uložení stavu evoluce na disk byla použita knihovna Xerces2 [28].

## 5.1 Omezení domény

Hra Desktop Dungeons je značně komplikovaná a obsahuje mnoho okrajových případů, které budou jedinci evoluce jen velmi obtížně potkávat a přizpůsobovat se jim. Proto jsme pro naši AI omezili hru následujícími způsoby:

- Žádní bohové - nejsou povoleny akce uctívání bohů. Jak již bylo zmíněno, tyto akce výrazně mění hru a jejich používání nám může více uškodit než pomoci.
- Žádné obchody - obdobně s předchozím, nepovolujeme nakupování předmětů v obchodech. Hra obsahuje celkem 47 předmětů s různými efekty. V rámci nízké komplexity se jich proto vyvarujme zcela.
- Omezené množství glyfů - redukovali jsme počet glyfů na 5 základních, které se objevují v popisu strategií na Desktop Dungeons wiki [15]. Celkově je ve hře 13 glyfů, a tak jsou jejich vzájemné synergie a kombinace těžko uchopitelné.

Všechna tři omezení se týkají pouze malé části hry a zkušený hráč dokáže i s nimi hru úspěšně dokončit.

## 5.2 Návrh jedince

Pro Desktop Dungeons AI jsme se rozhodli použít GA pro rekombinaci hladových strategií. Původní návrh hladového algoritmu, na kterém tento přístup

zakládáme, je uveden v kapitole 4.5. Obdobně jako v něm použijeme dva seznamy **elementárních strategií**, které svým uspořádáním dávají celkové chování AI. Strategie v prvním seznamu nazvěme **herní strategie** a v druhém **bojové strategie**.

Herní strategie popisují možná chování, která lze provádět při samotné hře. Jdou ale nad rámec elementárních akcí hry a obsahují pouze provádění smysluplných činností. Mezi herními strategiemi tak nenajdeme pohyb na pole, ze kterého neprozkoumáme část jeskyně, a stejně tak zde není ani prostý útok na nestvůru. Nemá totiž smysl na nestvůru jednou zaútočit, aniž abychom ji měli v plánu velmi brzy zabít. Bojové strategie místo toho obsahují například komplikovanější chování „Zabij nestvůru“, které rovnou provede sekvenci akcí, která nestvůru zabije. Sestrojit takovou sekvenci je však problémem samo o sobě, a právě k tomu využijeme seznam bojových strategií.

Bojové strategie pouze popisují způsoby boje s nestvůrou. Patří tak mezi ně obyčejný útok, kouzlo „Ohnivá koule“, použití lektvaru a i některé komplikovanější taktiky jako otrávení nestvůry a následný průzkum.

Každá elementární strategie může mít nějaké předpoklady použití a parametry svého chování. Například bojová strategie „Vypij lektvar zdraví“ má jako parametr maximální počet lektvarů, které smí vypít při souboji s nestvůrou dané úrovně. Ten je nastaven genetickým algoritmem. Předpokladem použití této strategie je pak vlastnictví lektvaru zdraví a dosavadní nevypití počtu lektvarů, které určuje parametr.

Nyní si přiblížíme všechny elementární strategie. Při navrhování jsme se inspirovali tipy a triky uvedenými na stránce Desktop Dungeons wiki [15]. Jejich podrobná specifikace je součástí elektronické přílohy (viz. Příloha A). Připomeňme, že oficiální herní názvy kouzel jsou poněkud kryptické, a proto zde uvádíme názvy odvozené od jejich efektu.

**Herní strategie:**

- Prozkoumej pole.
- Zabij nestvůru. Způsob zabití nestvůry zjisti použitím bojových strategií.

- Seber předmět ležící v jeskyni.
- Seber glyf ležící v jeskyni. Narozdíl od ostatních předmětů nemusí být glyf vždy sebratelný.
- Sešli kouzlo „Silný úder“/„První úder“/„Úhyb“ (tři různé strategie). Tato kouzla aplikují na hrdinu potenciálně dlouhodobý efekt. Mohou tak být efektivním způsobem využití many, kterou pak lze při průzkumu regenerovat.
- Konvertuj glyf. Za konverzi glyfu dostane hrdina malý bonus, a tak může konvertovat glyfy kouzel, které nepředpokládá, že využije.

**Bojové strategie:**

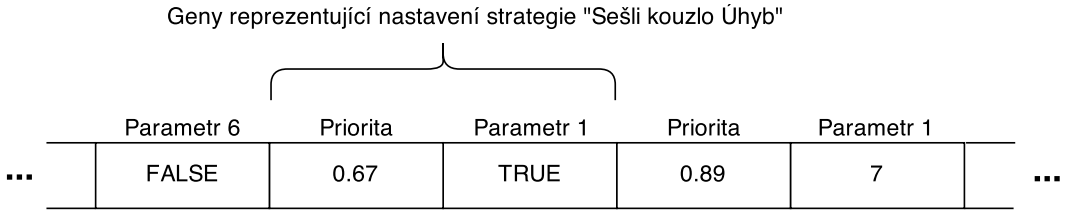
- Zaútoč.
- Sešli kouzlo „Ohnivá koule“. Toto kouzlo způsobí přímé zranění nestvůře a přitom nevyvolá útok nestvůry.
- Zaútoč pod vlivem kouzla „Silný úder“. Toto kouzlo zvyšuje sílu následujícího útoku hrdiny.
- Zaútoč pod vlivem kouzla „První úder“. Toto kouzlo umožní hrdinovi útočit v souboji jako první.
- Zaútoč pod vlivem kouzel „Silný úder“ a „První úder“.
- Sešli kouzlo „Úhyb“. Toto kouzlo zachrání život hrdiny před příští smrtící ránou.
- Otrav nestvůru a prozkoumej. Pomocí kouzla „Jed“ může hrdina otrávit nestvůru, a ta pak ztratí možnost regenerace při akcích průzkumu. Tak může hrdina zregenerovat klidně všechno svoje zdraví uprostřed souboje.
- Zkonvertuj glyf. Tato akce má smysl i uprostřed souboje, hrdina mohl v začátku boje kouzlo plně využívat, ale ke konci má možnost dorazit nestvůru díky bonusu z konverze.

- Použij lektvar zdraví.
- Použij lektvar many.
- Zabij nestvůru první úrovně a získej tím novou úroveň. Tato strategie aplikuje jednu z pokročilejších taktik zkušených hráčů Desktop Dungeons. Při získání nové úrovně se hráči obnoví všechno zdraví a mana. Dobrým přístupem tak může být v průběhu souboje se silnou nestvůrou si „odskočit“ zabít nestvůru velmi slabou, získat tím novou úroveň, zregenerovat zdraví a mít možnost pokračovat v souboji déle.
- Prozkoumej do finálního úderu. Pokud jsme v situaci, kdy náš příští útok nestvůru zabije, avšak její také zabije hrdinu, můžeme zkusit prozkoumat správné množství polí a zregenerovat právě takové množství zdraví nám i nestvůře, aby náš útok nestvůru stále zabil, ale její hrdinu už ne.
- Zaútoč při použití ochrany před smrtí. Tato strategie povolí útok i pokud by jím hrdina vypotřeboval cenný efekt ochrany před smrtí, který jej ochrání před smrtícím úderem.

Celkově tak máme 8 herních a 13 bojových strategií. Jejich pořadí a nastavení parametrů pak definuje chování našeho bota (umělého hráče). Bot v každém kroku hry použije herní strategii, která je v dané situaci použitelná a je co nejvýše v jeho seznamu. Použití strategie „Zabij nestvůru“ je možné jen pokud je pro nějakou nestvůru nalezen **plán útoku**. Tyto plány se pro každou nestvůru hledají také hladově, jen s použitím bojových strategií – bot simuluje opakované použití nejvýše umístěné bojové strategie, a pokud dojde k zabití nestvůry dříve než k zabití hrdiny, posloupnost použitých strategií tvoří plán útoku. Herní strategie „Zabij nestvůru“ pak vykoná tento plán.

Bot skončí hru ve chvíli, kdy nemůže použít žádnou herní strategii. Definice jednotlivých herních strategií zaručuje, že tato situace vždy nastane, protože každá spotřebovává nějaký omezený zdroj. Naše umělá inteligence tak nikdy neskončí zacyklením.

Pořadí strategií a jejich parametry pak musíme zakódovat do chromozómu jedince. Omezíme data v našich genech pouze na hodnoty typu boolean (prav-




Obrázek 5.1: Ukázka části chromozómu jedince s vnitřními hodnotami genů.

da/lež), celá čísla a čísla s plovoucí desetinnou čárkou. Každá strategie bude mít v jedinci svou prioritu, kterou bude číslo z intervalu $[0, 1]$. Podle těchto priorit budou strategie při spuštění jedince sestupně setříděny.

Každá strategie tak typicky potřebuje jen několik genů pro zakódování. Jeden pro uložení priority a další pro parametry. Například herní strategie „Sešli kouzlo Úhyb“ má jeden parametr, který určuje, zda se má provést pouze pokud má hrdina maximum many nebo vždy. Tuto strategii tak uložíme celkem do dvou genů. Některé strategie nemají parametr žádný, jiné jich mají více. Všechny strategie dohromady tvoří 64 genů. Ilustrace části chromozómu je na Obr. 5.1.

## 5.3 Fitness funkce

Fitness jedince by měla být úměrná jeho výkonu v jeskyni. Našim cílem je zabití hlavního bosse, ale to je úkol nadmíru obtížný, a tak prostá fitness měřící procentuální úspěšnost by vracela samé nuly. Potřebujeme tedy nějakým způsobem změřit, jak dobře si bot v jeskyni počínal.

Jednoduchým způsobem by bylo využít skóre hry, které je uděleno po každém pokusu. Skóre však bere v potaz některé hodnoty, které přímo nepodpoří úspěšnost našeho bota. Příkladem těchto hodnot je nízký čas dokončení nebo neklesnutí pod 20 % zdraví. My jsme se výchozí skórovací funkcí inspirovali, ale zjednodušili

jsme ji. Naše fitness se počítá takto:

$$\begin{aligned} fitness = 10 \cdot \text{zkušenosti} &+ 150 \cdot \text{lektvaryZdraví} \\ &+ 75 \cdot \text{lektvaryMany} \\ &+ \text{zdraví} \end{aligned}$$

Hlavním příspěvkem jsou získané zkušenosti (při dobrém běhu jsou jich stovky). Ty hrdina získává za zabíjení nestvůr, což je k vyhrání hry klíčové. Dále odměňujeme lektvary, které botovi zůstaly až do konce hry. To děláme z důvodu, že prvotní běhy byly příliš „chamtivé“ – boti často využili všechny lektvary v počáteční části hry, a přestože si celkově vedli dobře, neměli v podstatě šanci vyhrát finální souboj s bossem.

V případě, že bot hlavního bosse zabije a tím splní cíl hry, ztrojnásobíme jeho fitness. To se na první pohled zdá trochu nadhodnocené, ale je třeba mít na paměti, že vyhrání hry považujeme za obtížný úkol, a ve chvíli, kdy jej některý jedinec dokáže splnit, chceme podstatně rozšířit jeho gen v populaci. Také jsme předpokládali, že naše AI nebude zabíjet bosse obzvlášť pravidelně, spíše se zhruba 5-10% pravděpodobností.

Takto definovaná fitness je velmi vyrovnaná a pro GA je přirozené nejprve vyvinout jedince, kteří zabijí hodně nestvůr, poté je zefektivnit tak, že při tom nespotřebují tolik lektvarů, a nakonec je nechat využít tyto lektvary při dlouhém finálovém souboji s hlavním bossem. Ponechání alespoň několika lektvarů pro tento souboj je zpravidla nutností pro vyhrání hry.

Bota vpustíme do jeskyně celkem třikrát a jeho fitness přes všechny běhy sečteme. Tři opakování jsou málo pro určení stabilní hodnoty (fitness měřená přes tři běhy se může výrazně lišit od hodnoty získané přes jiné tři běhy), ale výsledky genetického algoritmu byly lepší, než při použití jen jednoho běhu. Použití výrazně více než tří běhů by trvalo příliš dlouho.

## 5.4 Genetické operátory

Vhodnou reprezentací jedince jsme umožnili využití klasického dvoubodového křížení. Ačkoliv funguje dobře, ukázalo se jako suboptimální a lepší výsledky jsme

získali s dodatečným využitím aritmetického křížení pro hodnoty priorit. Tomu bylo pravděpodobně proto, že potomci vzniklí aritmetickým křížením obsahují strategii lépe zkombinovanou z obou rodičů.

Mutaci můžeme také realizovat běžnými způsoby. Pozměnění hodnoty genu na náhodnou jinou z jeho domény funguje dobře. Dále jsme pro zrychlení GA implementovali chytrou mutaci, která se snaží měnit zjevně neefektivní hodnoty. Tato chytrá mutace kontroluje využívání lektvarů proti nestvůrám nízkých úrovní a také dává pozor, aby zabíjení nestvůr mělo vyšší prioritu než prozkoumávání.

Jako selekci jsme použili klasickou ruletovou selekci, která vybírá jednotlivé jedince s pravděpodobností úměrnou jejich fitness.

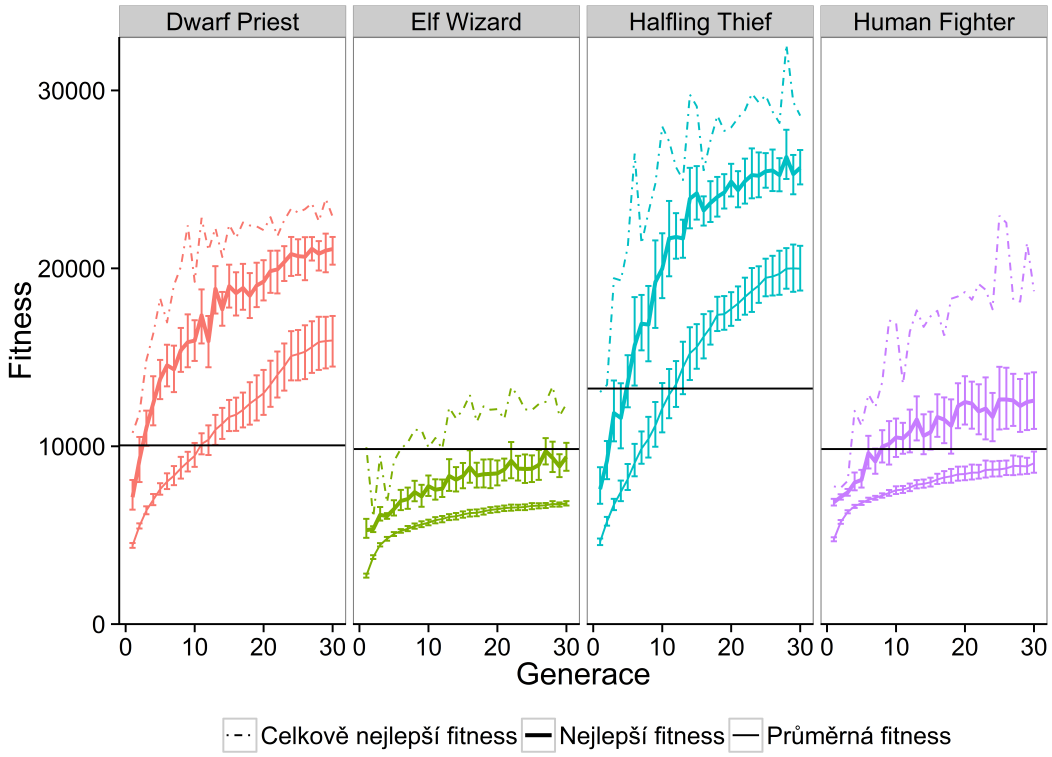
Všechny parametry operátorů, jako jsou pravděpodobnosti křížení, mutace a samotný jejich výběr, jsme vyladili metodou pokusu a omylu na malých populacích s nízkým počtem generací.

## 5.5 Výsledky

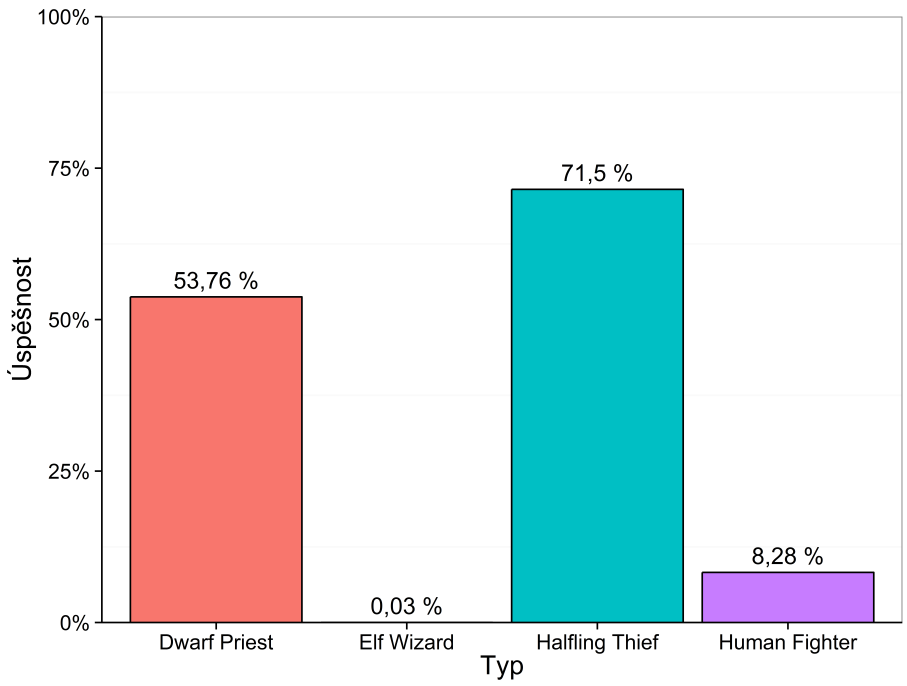
Pro finální běh evoluce jsme zvolili použití 100 jedinců, které evolvujeme po 30 generací. Pro tyto hodnoty jsme se rozhodli na základě experimentování s nimi. Vcelku vysoký počet jedinců dobře zastupuje a prohledává prostor řešení, a testovací běhy konvergovaly podstatně dříve, než po 30-ti generacích. Nejlepší jedince testovacích běhů jsme běžně nalézali v generacích 15-20, a poté už nepozorovali výrazné zlepšení.

Pro omezení vlivu náhody jsme celkem provedli 62 běhů evoluce pro 4 různé kombinace rasa-povolání hrdiny. Tyto jsme nezvolili nijak namátkově, ale použili jsme doporučené kombinace ze stránky Desktop Dungeons wiki [15]. S každou z těchto kombinací bylo provedeno minimálně 12 běhů. Průběh fitness v těchto bězích přehledně zaznamenává graf na Obr. 5.2. Celá evoluce běžela 7 dní na čtyřech moderních počítačích (intel i5-3470 na 3.2GHz, 4GB RAM), kde byly na každém z nich spuštěny 2 instance paralelně.

Jedince s nejvyšší fitness z celé evoluce pro každou kombinaci rasa-hrdina jsme podrobili bližšímu zkoumání a nechali je 10000krát zkusit vyhrát hru. Výhernost těchto zástupců jednotlivých kombinací ilustrujeme na Obr. 5.3.




Obrázek 5.2: Graf znázorňující vývoj fitness funkce během evoluce. Jednotlivé křivky představují celkově nejlepší fitness v generaci přes všechny běhy, největší fitness v dané generaci a průměrnou fitness v generaci. Vodorovná linie představuje maximální hodnotu fitness teoreticky získatelnou bez vyhrání hry. Tato je rozdílná pro jednotlivé kombinace rasa-povolání proto, že některé z nich dokáží generovat více zdraví či lektvarů, z čehož obojí je zohledněno ve výpočtu fitness funkce (Kap. 5.3).




Obrázek 5.3: Výhernost zástupců jednotlivých konfigurací s nejvyšší fitness. Hodnoty vyjadřují procentuální zastoupení výher v 10000 běhů.

Také si každá kombinace rasa-povolání vyvinula svoje vlastní chování a používala jiné strategie. Frekvenci používání jednotlivých strategií jsme zaznamenali v Tabulce 5.1.

Genetický algoritmus fungoval tak, jak jsme předpokládali. Je vidět, že hra neobsahuje žádné triviální chyby, kterých by náš algoritmus dokázal zneužít. Výslední jedinci jsou pouze dobře a relativně jemně optimalizované strategie.

Kompletní logy evoluce spolu se spustitelnými zástupci běhů jsou součástí elektronické přílohy (viz. Příloha A).

## 5.6 Diskuze

Evolvovaný bot byl schopen hru vyhrát za každé ze čtyř povolání s příslušnou rasou, ale s některými byl výrazně úspěšnější. To je pravděpodobně tím, že každé z těchto povolání vyžaduje odlišný přístup ke hře, některé z nichž jsou hůře reprezentovatelné v jedinci evoluce. U některých povolání může být více důležité přizpůsobit se stavu generované jeskyně a zvolit strategii až podle ní, což návrh naší evoluce neumožňuje. Také pravděpodobně nejsou všechna povolání stejně

| Název strategie | Dwarf Priest | Elf Wizard | Halfling Thief | Human Fighter |
|---|---|---|---|---|
| Konvertuj glyf | 1,12 | 2,51 | 3,12 | 2,59 |
| Prozkoumej | 91,56 | 76,78 | 61,03 | 95,06 |
| Seber glyf | 3,89 | 4,83 | 5,42 | 4,52 |
| Seber předmět | 22,11 | 19,97 | 25,10 | 22,83 |
| Sešli „Silný úder“ | 9,82 | 0,30 | 6,04 | 0,45 |
| Sešli „Úhyb“ | 8,59 | 2,65 | 7,24 | 9,62 |
| Sešli „První úder“ | 5,25 | 0,30 | 5,78 | 3,85 |
| Zabij nestvůru | 31,23 | 24,74 | 32,88 | 31,82 |
| Konvertuj glyf | 0,00 | 0,00 | 0,00 | 0,01 |
| Lektvar zdraví | 2,00 | 0,02 | 4,78 | 0,26 |
| Lektvar many | 0,00 | 0,01 | 4,23 | 2,69 |
| Otrav nestvůru a prozkoumej | 0,00 | 1,10 | 7,66 | 0,00 |
| Prozkoumej do finálního úderu | 0,30 | 1,27 | 0,80 | 0,00 |
| Sešli kouzlo „Ohnivá koule“ | 4,38 | 24,89 | 12,36 | 0,00 |
| Sešli kouzlo „Úhyb“ | 2,98 | 2,58 | 3,75 | 5,46 |
| Zabij nestvůru 1. úrovně | 0,50 | 0,57 | 0,65 | 0,77 |
| Zaútoč | 37,51 | 44,73 | 55,93 | 48,49 |
| Zaútoč s „Prvním úderem“ | 23,53 | 0,08 | 3,04 | 2,76 |
| Zaútoč se „Silným úderem“ | 13,58 | 0,00 | 0,12 | 0,00 |
| Zaútoč s obojím z nich | 0,28 | 0,00 | 0,14 | 0,00 |
| Zaútoč skrz ochranu před smrtí | 11,11 | 4,96 | 10,66 | 15,23 |

Tabulka 5.1: Četnost použití jednotlivých elementárních strategií zástupci kombinací rasa-povolání. Zprůměrováno přes 1000 spuštění. První část tabulky obsahuje všechny herní strategie, druhá část bojové. Při bližším prozkoumání je patrné, že různá povolání si vyvinula velmi rozdílné strategie.

silná, což ale není přímo vývojářským cílem, spíše je důležité, aby každé z nich představovalo přiměřenou a originální výzvu pro hráče.

Nicméně, celkově nás naše AI příjemně překvapila. Dvě ze čtyř povolání vyhrála hru v nadpolovičním počtu případů, což je mnohem více, než jsme předpokládali. Kombinace Human Fighter, která se doporučuje začínajícím hráčům, byla schopná vyhrát jen zhruba 8 % her. Síly a slabiny naší AI se totiž skrývají jinde, než u lidského hráče. Náš bot dokáže velmi rychle simulovat útoky a přesně spočítat pro každou nestvůru, zda jí dokáže zabít. To je pro člověka nezábavné a také v tom často udělá chybu. Na druhou stranu, na rozdíl od lidského hráče, nemá bot žádný dlouhodobý plán ani reakci na výjimečné situace.

Omezení domény nám nezabránilo vytvořit kvalitní AI. Přestože bot neměl přístup k některým mechanikám hry, dokázal pravidelně vyhrávat, alespoň za některá povolání. Návrh bota, který tyto mechaniky využít dokáže, by mohl naší výhernost ještě zvýšit, a teoreticky by mohl velmi výrazně pomoci povolání Wizard, které vyhrálo pouze 3 hry z 10000. Komplikovanější návrh je však obtížné správně nastavit.

Možnou slabinou evolučního algoritmu byl výpočet fitness pouze přes 3 běhy AI. Více běhů by přesněji ohodnotilo výkonnost daného jedince a GA by mohl detailněji ladit konfiguraci hladového algoritmu. Evaluace podstatně více než tří běhů by však spotřebovala mnoho výpočetního času.

### 5.6.1 Rozličnost evolvovaných jedinců

Po kratším pozorování Tabulky 5.1 je patrné, že různá povolání upřednostňují rozdílné strategie. Můžeme si všimnout několika zajímavostí.

Četnost použití elementární strategie „Prozkoumej“ je nižší pro povolání Thief a Wizard. Vysvětlením pro Thiefa je fakt, že často v souboji používá strategii otrávení nestvůry, při které prozkoumá další pole. Jeho povolání také generuje více předmětů v jeskyni, a tak objeví více pole při jejich sbírání. Wizard byl pouze slabší než ostatní, a častěji skončil tím, že mu některá nestvůra bránila v prozkoumání další části jeskyně.

Kouzla „Silný úder“ a „První úder“ si osvojila jen 2, resp. 3 povolání. To zbývající, které takřka nepoužilo ani jedno z těchto kouzel, byl Wizard, který se

místo toho soustředil na kouzlo „Ohnivá koule“. To je ovšem od něj vzhledem k jeho slabému základnímu útoku rozumné.

Fighter se naučil co nejvíce využívat efektu ochrany před smrtí. Toto povolání s tímto efektem začíná hru, a tak je pokus o vytěžení maxima z něj rozumný.

Bojová strategie „Zabij nestvůru 1. úrovně“ je dalším překvapením. Tato strategie obsahuje pokročilejší techniku hráčů hry, která při souboji s nestvůrou zabije jinou, velmi slabou nestvůru (např. 1. úrovně). To udělá ovšem v okamžiku, kdy i zabití slabé nestvůry stačí pro získání nové zkušenostní úrovně, čímž se hrdinovi doplní životy a mana. Díky tomu může hrdina v souboji klasickými způsoby pokračovat v podstatě dvakrát déle. Tato technika je náročná na provedení i pro lidského hráče, který si pro ní dlouze vytváří vhodnou situaci. Avšak i naše vcelku jednoduchá AI byla schopna tuto techniku použít, a to v průměru zhruba jednou za dvě hry.

Naopak, bojovou strategii „Konvertuj glyf“ si neosvojil nikdo. Konvertovat glyf při souboji je zřejmě výhodné jen ve velmi vzácných situacích.

Jak je vidět, povolání značně ovlivňují hru a hráč se jim musí přizpůsobit pro správný výběr strategie.

### 5.6.2 Použití mimo Desktop Dungeons

Pokud se budeme obdobným způsobem snažit o vytvoření AI do jiných rogue-like her [29], narazíme především na dva zásadní problémy – mnohem větší herní svět a částečně náhodné akce. Větší herní svět je problémem především pro fitness funkci, kde již nemůžeme hodnotit celé běhy kvůli jejich délce. Tento problém však lze řešit například zvýšením obtížnosti hry (pokud to hra nepodporuje, tak vlastním umělým způsobem, jako například zavedením chybovosti akcí), čímž snížíme čas běhu. Zde je ovšem potřeba dát pozor, aby dobré strategie v těžší variantě byly dobré i v té původní. Alternativním řešením může být evaluace bota přes náhodnou část hry, tedy ne nutně od začátku.

Náhodné akce jsou problémem vlastně jen pro naše prohledávání, které se snaží najít sekvenci bojových akcí pro zabití nestvůry. Toto ovšem můžeme v obecných rogue-like ignorovat a použít jen obyčejnou hladovou strategii, bez prohledávání, tedy s jiným způsobem určení, zda se má bot danou nestvůru pokusit

zabít. Klasická rogue-like toto rozhodnutí často nechávají na hráčově intuici, a tak jej můžeme zkusit modelovat jednoduchým způsobem, například if-then pravidly.

Takováto AI může pomoci zlepšit procedurální generátor prostředí rogue-like her. Lze jej použít jako automatického hodnotitele, který bude ukazovat nejen úspěšnost svého hraní, ale i rozmanitost generovaného obsahu přes výhodnost použití různých strategií. Současně tak může sloužit pro poukázání faktu, že některé herní mechanismy jsou pro hráče nevýhodné, a měli by jim vývojáři věnovat pozornost. Také jej lze použít za samotného běhu hry jako filtr obtížnosti, aby hráči nebyly předkládány nemožné úrovně, ale zároveň ani příliš jednoduché.

Kromě rogue-like her by bylo možné tento přístup použít i na jemu příbuzné žánry, například obecná RPG pro jednoho hráče, či s trochou modifikace i akční střílečky.

# Závěr

V rámci této práce jsme vytvořili Java framework pro tvorbu botů pro rogue-like hru Desktop Dungeons. Tento framework poskytuje jednoduchý přístup ke hře, jejímu ovládání a získávání informací ze hry. Struktura frameworku podporuje techniky prohledávání, pro něž byla mechanika hry reimplementována v Javě. Mezi další funkcionality patří možnost implementace vlastního generátoru úrovní pro hru.

Pomocí tohoto frameworku jsme implementovali hladovou strategii ke hře, kterou jsme dále optimalizovali genetickým algoritmem. Výsledkem této evoluce byla celkově úspěšná AI, která výrazně překonala ručně navržený hladový algoritmus. Nejlepší jedinec evoluce vyhrává hru ve zhruba 72 % případů, což je srovnatelné s průměrným lidským hráčem.

## Možná budoucí rozšíření

Framework umožňuje experimentování s implementací různých přístupů k AI v netradičním prostředí. Mohou tak být testovány techniky jako prohledávání, plánování a mnohé další. Lze také zkusit jiné možnosti evolučního algoritmu, jakým je např. genetické programování [26], které by dovolovalo vyvinutí obecnějších, nejen hladových, strategií.

Vzhledem k tomu, že framework umožňuje návrh vlastního procedurálního generátoru úrovní, lze se i jeho pokusit vylepšit. K tomu by bylo možné využít již hotového umělého hráče, buď jako automatického hodnotitele generovaných úrovní či jako filtru těch nevyhratelných.

Velmi zajímavé by také bylo postupné zlepšování generátoru a umělého hráče zároveň pomocí jeden druhého.

# Bibliografie


[1] QCF Design. *Desktop Dungeons*. `http://www.desktopdungeons.net/`. (cit. 2015-05-19).

[2] *Tactical Amulet Extraction Bot (TAEB) - Other Bots*. `http://taeb.github.io/bots.html`. (cit. 2015-05-19).

[3] Krajíček, J. „Framework pro implementaci botů pro hru NetHack“. Diplomová práce. Univerzita Karlova v Praze, 2015 (Čeká na obhájení).

[4] Mauldin, M. L. et al. „ROG-O-MATIC: a belligerent expert system“. In: *Fifth Biennial Conference of the Canadian Society for Computational Studies of Intelligence*. 1983.

[5] Mitchell, M. *An Introduction to Genetic Algorithms*. MIT Press, 1996. isbn: 0-262-13316-4.

[6] Sweetser, P. M. „How to build evolutionary algorithms for games“. In: Rabin, S. *AI Game Programming Wisdom 2*. Charles River Media, 2004, s. 627–638. isbn: 1-58450-289-4.

[7] Esparcia-Alcázar, A. et al. „Controlling bots in a First Person Shooter game using genetic algorithms“. In: *IEEE Congress on Evolutionary Computation*. 2010.

[8] Ponsen, M. a Spronck, P. „Improving Adaptive Game AI with Evolutionary Learning“. In: *Proceedings of the Computer Games: Artificial Intelligence, Design and Education Conference*. 2004, s. 389–396.

[9] Böhm, N., Kóokai, G. a Mandl, S. „An Evolutionary Approach to Tetris“. In: *Proceedings of the 6th Metaheuristics International Conference*. 2005, s. 137–148.

[10] Lucas, S. M. a Kendall, G. „Evolutionary computation and games“. In: *IEEE Computational Intelligence Magazine* 1 (2006), s. 10–18. issn: 1556603X.

[11] Russell, S. a Norvig, P. *Artificial Intelligence: A Modern Approach*. 3. vyd. Prentice Hall Press, 2009. isbn: 978-0136042594.

[12] Poole, D. L., Mackworth, A. a Goebel, R. G. „Computational Intelligence and Knowledge“. In: *Computational Intelligence: A Logical Approach*. 1998, s. 1–22. isbn: 978-0195102703.

[13] Legg, S. a Hutter, M. „A Collection of definitions of intelligence“. In: *Frontiers in Artificial Intelligence and Applications* 157 (2007), s. 17–27.

[14] Garda, M. B. „Neo-rogue and the essence of roguelikeness“. In: *Homo Ludens* 1.5 (2013).

[15] *Desktop Dungeons - DDwiki*. `http://www.qcfdesign.com/wiki/Desktop Dungeons`. (cit. 2015-05-19).

[16] YoYo Games. *GameMaker: Studio*. `http://www.yoyogames.com`. (cit. 2015-05-19).

[17] Flanagan, D. *JavaScript: the definitive guide*. 6. vyd. O'Reilly Media, 2011. isbn: 978-0596805524.

[18] Oracle Corporation. *Invoking JavaScript Code From an Applet (Java Documentation)*. `https://docs.oracle.com/javase/tutorial/deployment/applet/invokingJavaScriptFromApplet.html`. (cit. 2015-05-19).

[19] Gemrot, J. et al. „Pogamut 3 can assist developers in building AI (Not only) for their videogame agents“. In: *Agents for Games and Simulations*. Springer, 2009, s. 1–15.

[20] Fette, I. a Melnikov, A. *RFC 6455: The WebSocket Protocol*. `https://tools.ietf.org/html/rfc6455`. (cit. 2015-05-19).

[21] Oracle Corporation. *Java Platform, Standard Edition (Java SE)*. `http://docs.oracle.com/javase/`. (cit. 2015-05-19).

[22] Brooks, R. A. „Intelligence without representation“. In: *Artificial Intelligence* 47.1-3 (1991), s. 139–159. issn: 00043702.

[23] Champandard, A. J. *Behavior Trees for Next-Gen Game AI*. `http://aigamedev.com/insider/presentations/behavior-trees`. (cit. 2015-05-19).

[24] Ghallab, M., Nau, D. a Traverso, P. *Automated Planning: Theory & Practice*. Elsevier, 2004. isbn: 1-55860-856-7.

[25] Ghallab, M. et al. *PDDL-the planning domain definition language*. 1998.

[26] Poli, R., Langdon, W. B. a McPhee, N. F. *A Field Guide to Genetic Programming*. Lulu Enterprises, 2008. ISBN: 978-1409200734.

[27] *JGAP: Java Genetics Algorithm Package*. `http://jgap.sourceforge.net/`. (cit. 2015-05-19).

[28] *Xerces2 Java Parser*. `https://xerces.apache.org/xerces2-j/`. (cit. 2015-05-19).

[29] *Rogue Basin*. `http://www.roguebasin.com`. (cit. 2015-05-19).

# Seznam zkratek

AI - artificial intelligence, umělá inteligence (viz Kap. 1.1)

EA - evoluční algoritmus (viz Kap. 1.3)

GA - genetický algoritmus (viz Kap. 1.3)

JS - JavaScript, skriptovací jazyk, který často běží v internetovém prohlížeči

RPG - role-playing game, hra na hrdiny

# A. Obsah elektronické přílohy

Elektronická příloha obsahuje tyto adresáře se zde uvedeným obsahem.

- `bin` - Spustitelné Java soubory evoluce a ukázkových botů.
- `doc` - Programátorská dokumentace evoluce a frameworku.
- `evolData` - Data získaná z hlavního běhu evoluce, včetně skriptů pro generování grafů uvedených v této práci.
- `lib` - Desktop Dungeons Framework jako připravená knihovna pro použití. Také ostatní knihovny potřebné pro zkompilování zdrojových souborů.
- `spec` - Specifikace protokolu API a evolvovaného bota.
- `src` - Zdrojové kódy genetického programu, frameworku a ukázkových botů.
- `text` - Tato práce ve formátu PDF.
- `webdungeonAPI` - Alfa verze hry Desktop Dungeons včetně API.

Adresáře se zdrojovými kódy obsahují stručné `README.txt` vysvětlující způsob kompilace, obdobně adresáře se spustitelnými programy obsahují krátkou uživatelskou dokumentaci.

# B. Implementovaná část hry pro simulaci

Vzhledem k rozsáhlosti hry byla pro její simulaci v Java části frameworku implementována jen její podčást. Zde si vyjmenujeme mechaniky a nastavení hry, které *nejsou* simulací podporovány:

- Používání oltářů a obchodů.
- Sesílání kouzel kromě následujících pěti: „Ohnivá koule“, „Silný útok“, „První útok“, „Úhyb“ a „Jed“.
- Povolání kromě základních čtyř: Fighter, Priest, Thief a Wizard.
- Speciální efekty monster kromě následujících osmi: Goblin, Meat Man, Warlock, Zombie, Goat, Gorgon, Serpent, Wraith.

Všechny ostatní mechaniky a nastavení může uživatel používat. V případě užití některé z neimplementovaných částí hry nemusí, a nejspíše nebude, simulace akcí produkovat správné výsledky.