Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Evoluční algoritmy a herní situace

Diplomová práce

Autor: Bc. David Vondráček
Studijní obor: Datová věda

Vedoucí práce: Ing. Karel Mls, Ph.D.

Hradec Králové srpen 2023

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 10.8.2023 Bc. David Vondráček

Poděkování:

Děkuji vedoucímu diplomové práce Ing. Karlu Mlsovi, Ph.D. za metodické vedení práce.

Anotace

Cílem této diplomové práce je uvést do problematiky evolučních algoritmů a jejich využití v herních situacích. Obsah zahrnuje nástin historie heuristiky a výklad hlavních typů heuristických algoritmů, jako jsou heuristické algoritmy, chamtivé algoritmy, lokální vyhledávací algoritmy a metaheuristické algoritmy. Práce dále zkoumá evoluční a genetické algoritmy s detailním popisem fitness funkce, selekce, křížení a mutace. Genetické programování je rovněž analyzováno a zdůrazňuje se jeho aplikace na příkladu genetického algoritmu. Teorie her je představena s historickým vývojem a dvěma přístupy k evoluční teorii her. Důraz je kladen na evoluční přístup ke hře dáma s hodnocením různých heuristik. Práce rozebírá také možnosti zlepšení her s využitím evolučních algoritmů a umělé inteligence ve hrách.

V závěru jsou shrnuty výsledky a prezentováno ukázkové řešení. Práce je uzavřena závěry a doporučeními. V seznamu použité literatury jsou uvedeny relevantní zdroje, které byly využity při tvorbě této práce. Celkově poskytuje tato práce ucelený přehled o aplikaci evolučních algoritmů v herním prostředí a přináší nové poznatky a doporučení pro další výzkum a vývoj herních systémů.

Annotation

Title: Evolutionary algorithms and game situations

The aim of this master thesis is to introduce evolutionary algorithms and their use in game situations. The content includes an outline of the history of heuristics and an explanation of the main types of heuristic algorithms, such as heuristic algorithms, greedy algorithms, local search algorithms, and metaheuristic algorithms. The work further examines evolutionary and genetic algorithms with a detailed description of the fitness function, selection, crossover, and mutation. Genetic programming is also analyzed, and its application is highlighted using the example of the genetic algorithm. Game theory is introduced with historical

development and two approaches to evolutionary game theory. Emphasis is placed on an evolutionary approach to the game of checkers with the evaluation of various heuristics. The work also discusses the possibilities of improving games using evolutionary algorithms and artificial intelligence in games.

The results are summarized at the end, and a sample solution is presented. The work is closed with conclusions and recommendations. The list of references includes relevant sources that were used in the creation of this work. Overall, this work provides a comprehensive overview of the application of evolutionary algorithms in the game environment and brings new insights and recommendations for further research and development of game systems.

Obsah

Seznam obrázků

Seznam tabulek

Seznam ukázek kódu

1 Úvod

Skupina počítačových metod známých jako evoluční algoritmy (EA) čerpá inspiraci z biologické evoluce. Aby bylo možné vytvořit chování umělé inteligence (UI), které se může časem přizpůsobovat a měnit, jsou tyto algoritmy často využívány při vytváření videoher.

EA lze použít k vytvoření hráčů UI ve hrách, kteří mohou v průběhu času rozvíjet a zdokonalovat svou strategii. Tyto algoritmy lze použít k napodobení procesu přirozeného výběru, ve kterém přežijí nejsilnější konkurenti a reprodukují se, zatímco ti slabší jsou eliminováni.

Jednou z aplikací EA v hraní je vytváření strategických her, kde hráči UI musí najít vítězné strategie. Nejlepší hráče lze najít simulací řady her s velkým počtem náhodně generovaných hráčů UI, z nichž má každý odlišný přístup vytvořený pomocí EA. Poté lze vytvořit novou generaci hráčů s využitím úspěšných jedinců, kteří se budou nadále vyvíjet a časem zdokonalovat.

Vytvoření neuronových sítí pro herní inteligenci je další způsob využití EA v hraní. Výkon UI lze zvýšit použitím EA k optimalizaci vah a parametrů neuronové sítě. Tento proces se nazývá neuroevoluce.

EA jsou účinným nástrojem pro vytvářeních chytrých a adaptabilních hráčů UI ve videohrách a lze je aplikovat pro různé herní žánry, od střílečeк z pohledu první osoby po strategické hry.

Pro použití EA v herních prostředích je potřeba vytvořit funkci fitness, která hodnotí, jak efektivně fungují ne hráčská postava (NHP) nebo herní prvky. Funkce fitness ve střílečce z pohledu první osoby může posoudit, jak efektivně může NHP hráče eliminovat. Rychlost, kterou může auto dokončit dráhu může být také měřena fitness funkcí.

EA lze použít k opakovanému generování NHP nebo herních mechanismů, které optimalizují funkci fitness. Pro vývoj nové generace NHP nebo herních parametrů EA nejprve vytvoří populaci NHP nebo herních sad parametrů a poté iterativně aplikuje výběr, reprodukci a mutaci. Nejlepší NHP nebo herní parametry jsou vybrány ze současné populace, aby byli rodiči následující generace.

Výběrové řízení může být řízeno funkcí fitness nebo dalšími standardy, jako je novost nebo rozmanitost. Reprodukce je proces sloučení vybraných NHP nebo herních nastavení za účelem vytvoření potomků. Při křížení dochází k výměně části DNA rodičů, nebo mutace, při které geny rodičů projdou náhodnou změnou.

EA je podrobena náhodnosti prostřednictvím mutace, což umožňuje zkoumání nových řešení. Funkce fitness se znovu používá k posouzení výkonu nově vytvořených NHP nebo parametrů hry. Dokud není nalezena dobrá odpověď, procesy selekce, reprodukce a mutace se opakují.

Najít rovnováhu mezi prozkoumáváním a využíváním při použití EA v herních nastaveních může být obtížné. Průzkum je hledáním nových řešení, zatímco využívání je vylepšením těch současných. Nadměrné prozkoumávání může vést k horším řešením. Pomocí strategií, jako je adaptivní mutace nebo selekce, je možné dosáhnout rovnováhy mezi průzkumem a využíváním.

EA mohou být účinným nástrojem pro navrhování flexibilních NHP a vylepšování nastavení hry. Efektivní používání EA vyžaduje definici fitness funkce, pečlivé vyladění parametrů a rovnováhu mezi využíváním a průzkumem. EA mohou vytvářet náročnější a zajímavější hry pro hráče, pokud jsou správně používány.

2 Cíl práce a metodika

Cílem této diplomové práce je zkoumat využití EA v rámci herních situací. Tato práce se zaměřuje na posouzení efektivity EA ve vývoji UI pro hry.

Hlavní výzkumné otázky se týkají historického vývoje a hlavních typů heuristik včetně jejich využití v herním prostředí. Dále způsobu, jakým evoluční a genetické algoritmy (GA) pracují a jak se dají použít pro vývoj UI ve hrách. Vlivu teorie her na vývoj EA, potenciálu pro zlepšení her pomocí EA a praktického využití těchto algoritmů a teorií pro simulaci EA, kde se hráči postupně zlepšují a snaží se dojít do cíle.

Pro analýzu a prezentaci dat z experimentů je využita vizualizace dat, konkrétně vytváření teplotních map, jakým způsobem se hráči dostávali do cíle.

Splnění cíle této práce je dosaženo aplikací PyCharm v Pythonu, kde bylo otestováno několik evolučních metod, jako jsou různé druhy selekce, křížení a mutace. Tyto testy byly vyhodnoceny a mezi sebou porovnány.

3 Heuristika

Heuristika umožňuje rychlé a efektivní řešení problémů. Toto urychluje rozhodování a umožňuje lidem plnit své úkoly, aniž by se neustále pozastavovali nad zvažováním dalšího postupu. Heuristika má své výhody a nevýhody. Může být užitečná za mnoha okolností, ale také může vést ke kognitivním zkreslením [1].

Jedním z hlavních úkolů je zjednodušit problém ignorováním některých informací nebo pohledem na část všech potenciálních řešení. Heuristika byla obvykle považována za účinné a nezbytné nástroje, které však přinášejí pouze druhé nejlepší výsledky.

Ačkoli se definice heuristiky v různých vědeckých oborech odlišuje, v mnoha z nich byly vzaty v úvahu heuristické metody. Je možné rozlišit dvě hlavní heuristické techniky. První technikou jsou normativní procesy, které vysvětlují, jak nalézt rozumné řešení s ohledem na omezení, jako je výpočetní náročnost a omezený čas ve filozofii, matematice a operačním výzkumu UI. Druhou technikou jsou deskriptivní modely v biologii a psychologii. Tyto modely pak popisují, jak jednotlivci a další tvorové shromažďují informace z vnějšího a vnitřního (paměťového) světa a činí rozhodnutí na základě těchto znalostí [2].

3.1 Historie heuristiky

Heuristické techniky byly původně vytvořeny ve filozofii a matematice k řešení problémů s algoritmickým řešením obtížných problémů. Pro demonstraci algoritmické tradice je použit mechanický aparát vytvořený ve třináctém století katalánským filozofem Raimundem Lullusem [3]. Tento nástroj poskytoval všechny možné kombinace filozofických a teologických argumentů. Základní třídy argumentu byly reprezentovány šesti soustřednými disky, z nichž každý obsahoval devět dalších atributů. Každá možná kombinace vlastností byla tvořena otáčením disků proti sobě, automatickým vytvářením různých argumentů i vytvářením nových. Německý filozof Gottfried Wilhelm Leibniz měl v 17. století podobný cíl: vytvořit algoritmus pro řešení jakéhokoli myslitelného problému pomocí univerzálního jazyka, který by umožňoval popis každého řešení [4]. Bylo zjevné, že taková algoritmická metoda povede ke složité kombinatorice. Pozdější výzkumníci

proto studovali heuristiku jako způsob, jak pomoci řešitelům problémů dospět k řešení, aniž by museli důkladně zkoumat prostor alternativ.

Obrázek 1 Mechanické zařízení vyvinuté katalánským filozofem Raimundem Lullusem
Zdroj: [2]

Francouzský matematik sedmnáctého století René Descartes vyvinul několik přímých pokynů, které měly řešitele nasměrovat spíše k příslušným prvkům než ke všem složkám problému. Tuto strategii dále rozvinul v devatenáctém století matematik a filozof Bernard Bolzano, který ve své teorii vědy navrhl různé heuristiky pro řešení problémů (např. pravdy prostřednictvím něčeho, o čem ještě není známo, že je pravdivé, spíše než vyvozováním pravdy ze známých pravd) [5]. Heuristika vyvinutá těmito výzkumníky sestávala z obecných technik, které byly vysvětleny nejednoznačnými slovy. Jejich hlavním cílem bylo vyhnout se svévolnému procesu řešení, poskytnout cestu k poznání nad rámec řádné dedukce a podpořit originální myšlení. Například několik heuristik hledalo analogické nebo metaforické reprezentace problému, které by byly vhodné.

Práce matematika George Pólyi [6] pomohla heuristickým metodám řešení problémů a jeho objevováním dopomohla získání většího významu v moderní matematice. Pólyovy procesy byly tvořeny přímočarými pokyny, jako je rozdělení fází vedoucích k řešení na menší, například nalezením analogie k problému, určením problému, který je více specializovaný, nebo rozdělením problému a jeho opětovným sestavením. Později byla vznikající disciplína UI ovlivněna také postupy

inspirovanými Pólyou. Ukázalo se, že heuristika má praktické využití v operačním výzkumu a v odvětví aplikované matematiky.

Od 50. let 20. století, kdy se počítač poprvé objevil jako výpočetní nástroj a metafora mysli, existují pokusy replikovat inteligentní chování ve strojích. Vytváření počítačových programů, které jsou schopny provádět úkoly, jako je hraní šachů, dokazování logických teorémů, nebo porozumění jazyku, bylo středem zájmu vznikajícího oboru UI. Heuristická pravidla ve výzkumu UI byla přesně formulována z hlediska počítačových modelů, na rozdíl od dřívějších postupů heuristiky, které je často zobrazovaly velmi široce [2].

3.2 Hlavní typy heuristik

Existuje několik druhů optimalizačních algoritmů, které se používají pro řešení daných otázek. Mezi ně můžeme zařadit následující [7]:

- Heuristické algoritmy
- Chamtivé algoritmy
- Lokální vyhledávací algoritmy
- Metaheuristické algoritmy

3.2.1 Heuristické algoritmy

Pokud je v nejhorším případě chování algoritmu nejisté, označuje se jako heuristický algoritmus. V důsledku toho není možné předvídat, jak by se tento algoritmus choval v různých scénářích uvažovaného problému. K posouzení účinnosti heuristického algoritmu lze použít výpočetní experiment. Pro experiment je vytvořena sada testovacích příkladů a nejlepší řešení objevená přesným algoritmem jsou porovnána s těmi, kterých bylo dosaženo pomocí vyhodnocované heuristiky [7].

3.2.2 Chamtivé algoritmy

Heuristika pokrývá spektrum technik. Mezi ně patří i takzvaný chamtivý algoritmus, který je jedním z příkladů jednoduché heuristiky. Chamtivý algoritmus se pokouší vybrat nejlepší možné dílčí řešení v každé fázi. Takový přístup může vyústit v nalezení ideálních řešení. Obecně však chamtivé algoritmy generují pouze

řešení, která nejsou optimální. V důsledku toho byly pro problémy, které jsou složitější, navrženy složitější algoritmy, včetně algoritmů místního vyhledávání [7].

3.2.3 Lokální vyhledávací algoritmy

Algoritmy lokálního vyhledávání jsou další instancí heuristiky. Tyto algoritmy začínají řešením a iterativně vytváří oblast kolem té, která je aktuálně nejlepší. Okolí je souborem všech řešení, kterých lze dosáhnout přijatelnými kroky ze současného řešení. Během procesu používají různé operátory, jejichž funkce závisí na konkrétním problému. Cílem těchto operátorů je vytvořit zcela nové funkční řešení ze stávajícího funkčního řešení.

Algoritmus lokálního vyhledávání používá metodu místního vyhledávání k nalezení nového řešení daného souseda ve všech možných krocích. Dokud není splněna podmínka pro zastavení, provádí se výše zmíněný přístup lokálního algoritmu k identifikaci nového řešení. V tomto případě se algoritmus zastaví, protože je velmi nepravděpodobné, že by došlo k dalšímu snížení minimalizované objektivní funkce [7].

3.2.4 Metaheuristické algoritmy

Metaheuristické algoritmy obsahují významnou podmnožinu heuristických algoritmů. Šablonou lokálního vyhledávacího algoritmu je metaheuristický algoritmus. Šablona obsahuje řadu nastavení ovládacích prvků, která ovlivňují okolnosti, ve kterých se algoritmus ukončuje a kvalitu generovaných řešení dané metaheuristickým algoritmem. Možnými příklady metaheuristiky jsou simulované žíhání, hledání tabu a EA. Tyto algoritmy využívají řadu složitých technik k vytvoření zcela nového řešení ze stávajícího řešení [7].

Simulované žíhání

Simulované žíhání je metaheuristická optimalizační technika, která se inspiruje procesem fyzického žíhání materiálu. Jeho hlavním cílem je nalézt přibližně optimální řešení pro problémy s velkým stavovým prostorem, kde by hledání exaktního řešení bylo časově náročné nebo nemožné. Algoritmus simulovaného žíhání začíná s náhodně zvoleným počátečním řešením. Poté

iterativně prochází stavovým prostorem, přičemž v každém kroku vybírá náhodně nějaké sousední řešení. Pokud je toto sousední řešení lepší než aktuální, je automaticky přijato jako nové aktuální řešení.

Jedním z klíčových prvků simulovaného žíhání je teplotní schéma. Teplota reprezentuje pravděpodobnost přijetí horšího řešení v každém kroku a postupně se snižuje s průběhem času (ochlazováním). Tím se redukuje pravděpodobnost přijetí horších řešení, což vede k postupnému "ochlazování" algoritmu a omezení skoků do horších stavů. Důležité je, že simulované žíhání občas přijme i horší řešení, což mu umožňuje uniknout z lokálních optimálních bodů a hledat celkově lepší řešení. Tato pravděpodobnostní strategie dává algoritmu schopnost procházet i výrazně složitými a nerovnoměrnými prostory řešení.

Simulované žíhání bylo úspěšně využito v mnoha aplikacích, jako jsou problémy obchodního cestujícího, optimalizace výrobních procesů, plánování tras nebo návrhy proteinů v bioinformatice. Díky své schopnosti vyhledávat přibližně optimální řešení v rozsáhlých a složitých prostorech je tato technika ceněná pro řešení různorodých praktických problémů [8].

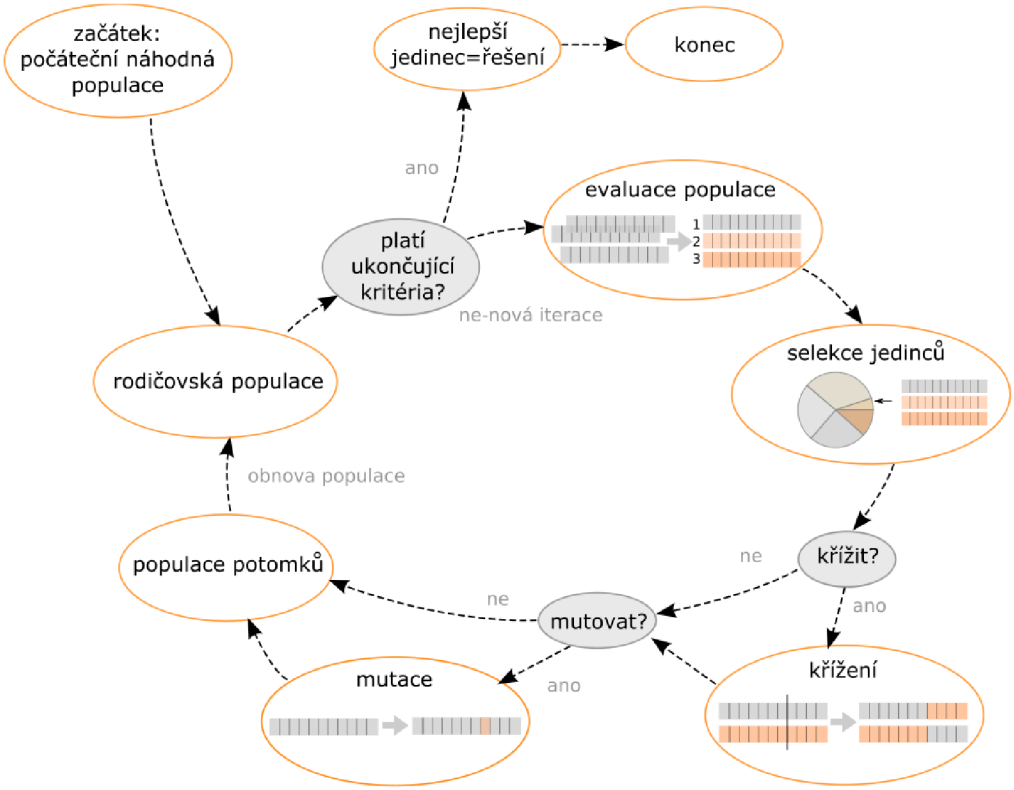
Hledání tabu

Hledání tabu je metaheuristická optimalizační technika pro řešení kombinatorických problémů. Inspiruje se principy zakazování kroků, aby se vyhnula uvíznutí v lokálních optimálních bodech. Algoritmus systematicky prochází stavový prostor, vyhodnocuje každé řešení podle optimalizačního kritéria a udržuje paměť – tabu seznam, kde ukládá nedávné kroky a zakázané operace.

Hledání tabu má tři hlavní části: inicializaci, hlavní cyklus a aktualizaci tabu seznamu. Na začátku se nastaví počáteční řešení a vytvoří se prázdný tabu seznam. V hlavním cyklu se opakovaně prochází stavový prostor, hledají se sousední řešení a vybírají nejlepší kroky. Přitom se dbá na to, aby se neopakovaly zakázané kroky z tabu seznamu. Tabu seznam se průběžně aktualizuje. Hledání tabu se často kombinuje s jinými metodami a nabízí adaptabilitu díky různým modifikacím [9].

4 Evoluční a genetické algoritmy

Genetické učení odkazuje na evoluční výpočetní techniky, vyhledávací algoritmy jsou založeny na přirozeném výběru a genetických mechanismech. Během evolučního procesu jsou genetické operátory včetně křížení, mutace a selekce využívány k postupnému zvyšování kvality populace. Reprezentace potenciálních řešení, jako jsou GA a optimalizace roje částic, obvykle používá vektorovou reprezentaci s pevnou délkou. K zobrazení jednotlivců lze použít i jiné reprezentace, jako jsou stromy proměnlivé délky používané v genetickém programování. Aby bylo možné vyřešit problémy s optimalizací ve strojovém učení, tak evoluční učení využívá evoluční výpočty (EV). EV jsou technika výpočetní inteligence, která se inspirovala přirozenou evolucí založenou na populaci. Úspěch EV závisí na generačním pokroku populace. V EV existují dvě základní kategorie. První kategorií je inteligence roje, která zahrnuje optimalizaci roje částic a optimalizaci kolonií mravenců. Druhou kategorií jsou EA, které zahrnují GA, genetické programování, evoluční strategie a evoluční programování [10].



Obrázek 2 Princip EA
Zdroj: [11]

4.1 Optimalizace pro více cílů

Optimalizace pro více cílů jsou oblast optimalizačních problémů, které se zabývají hledáním optimálních řešení, zohledňující více než jeden konfliktní cíl. V těchto problémech neexistuje jediné "nejlepší" řešení, ale spíše sada kompromisních řešení, která tvoří takzvanou Paretovu optimální množinu nebo Paretovu optimální frontu [12]. Paretova optimální fronta obsahuje řešení, kde nelze zlepšit jeden cíl bez zhoršení jiného cíle. To znamená, že žádné řešení na této frontě není lepší než ostatní, ale každé z nich je nejlepší v nějakém smyslu.

Optimalizace pro více cílů se zaměřují na nalezení této fronty nebo její aproximaci. Optimalizace pro více cílů používá metody, které zahrnují EA, GA, evoluční strategie, diferenciální evoluci a další.

Tyto algoritmy umožňují procházet prostor možných řešení a hledat efektivní kompromisní řešení, která se nacházejí na Paretově optimální frontě. Optimalizace pro více cílů jsou klíčovou disciplínou v různých oborech a průmyslových odvětvích, kde existuje potřeba řešit problémy, které zahrnují více než jeden konfliktní cíl. Příklady zahrnují rozhodování o investicích, plánování výroby, optimalizaci dopravních tras, návrh systémů a mnoho dalších. Díky své schopnosti hledat kompromisní řešení je optimalizace pro více cílů klíčovým nástrojem pro rozhodování v komplexních a reálných problémových doménách [13].

4.2 Evoluční výpočty

EA začíná vytvořením populace jedinců, z nichž každý představuje potenciální řešení problému. První populace může být vytvořena náhodně. K posouzení každého jednotlivce se používá funkce fitness a výstup funkce odhaluje, jak efektivně může každý jednotlivec řešit zkoumaný problém. Poté, aby se vytvořili noví jedinci, jsou na vybrané rodiče aplikovány genetické operátory křížení, mutace a reprodukce. Jedinci s vyšší fitness budou pravděpodobněji vybráni jako rodiče potomků, kteří budou následně vytvořeni. Dokud není splněn požadavek na ukončení (například dosažení určitého počtu generací), tak tento proces pokračuje. V důsledku EA je vybrán nejlepší jedinec.

Různé EA lze rozlišit podle jejich reprezentace, což je klíčová vlastnost. Jednou z běžných reprezentací je vektorová reprezentace s pevnou délkou (také známá jako binární nebo reálná hodnota), která využívá vektor k reprezentaci jednotlivce. Příklady běžných vektorově založených EA jsou optimalizace roje částic a GA [10].

4.2.1 Fitness funkce

Funkce fitness je použita v modelovací fázi GA a hodnotí, jak GA úspěšně ohodnotil jednotlivá řešení. Tato funkce zahrnuje optimalizační strategie a umožňuje kombinovat hodnoty funkce pro jednotlivé cíle, když je potřeba současně maximalizovat více cílů, například pomocí váženého součtu. Hodnocení kvality řešení je klíčové a obecně platí, že méně kvalitní řešení by měla mít nižší hodnotu fitness funkce. Snaha o optimalizaci GA často spočívá v minimalizaci počtu volání funkcí fitness, zejména pokud jsou volání výpočetně nákladná nebo časově náročná. Efektivita GA pro řešení problému je často posuzována počtem vyhodnocení, které jsou nutné pro nalezení optimálního řešení nebo pro dosažení požadované přesnosti aproximace. Snížení počtu volání funkcí je zásadní, zejména v případech, kdy je každé vyhodnocení náročné, například pokud vyžaduje spuštění simulačního modelu nebo generování konstrukčních prvků [14].

4.2.2 Selekce

Optimalizační nástroj GA inspirovaný procesem přirozeného výběru závisí na selekci. V GA výběr určuje, kteří jedinci z populace budou vybráni jako rodiče pro reprodukci v závislosti na jejich zdatnosti nebo kvalitě. Selekce řídí využívání a prozkoumávání vyhledávacího prostoru upřednostňováním jedinců s lepší kondicí a postupným rozšiřováním populace v následujících generacích.

GA používají různé selekční techniky, z nichž každý má jedinečné vlastnosti a kompromisy. Často používané metody výběru jsou následující [15]:

- Ruletový výběr
- Turnajový výběr
- Výběr na základě pořadí
- Elitismus

Ruletový výběr

Každému členovi populace je dána pravděpodobnost výběru na základě jeho relativní zdatnosti pomocí výběru v ruletě. K výběru kandidátů se používá ruleta a kandidáti, kteří mají větší funkci fitness, mají větší šanci, že budou vybráni. Je vytvořeno kumulativní rozdělení pravděpodobnosti, s pravděpodobností, že každý jedinec bude nepřímo korelován s jeho zdatností. Po vygenerování náhodného čísla jsou jedinci vybíráni na základě toho, kde toto číslo leží v kumulativním rozdělení [15].

Turnajový výběr

Při turnajovém výběru je náhodně vybrána podmnožina jedinců z populace. Jedinec s nejvyšší úrovní fitness je vybrán jako rodič poté, co se jedinci utkají mezi sebou. Tento krok se několikrát opakuje, pokud je třeba vybrat několik rodičů. Vzhledem k tomu, že turnajový výběr podporuje ty, kteří mají vysokou funkci fitness a zároveň dává šanci být vybrán i jedincům s nižším fitness, tak umožňuje rovnováhu mezi průzkumem a využíváním [15].

Výběr na základě pořadí

Namísto výběru jedinců na základě jejich skutečné hodnoty fitness, výběr na základě pořadí vybírá jedince podle toho, kde se v populaci nacházejí. Pořadí jednotlivce se stanoví klasifikací populace podle fitness a přiřazením pořadí podle toho. Jedinci s vyšší funkcí fitness mají větší šanci, že budou vybráni. Tím, že omezuje vliv vysokých hodnot fitness, podporuje tato strategie rozmanitost v populaci [16].

Elitismus

Malá skupina nejlepších jedinců je udržována v nezměněné podobě z generace na generaci prostřednictvím elitismu. Tito jedinci jsou automaticky

vybráni jako rodiče, aby se dosud nejlepší řešení neztratilo. Elitismus může urychlit konvergenci GA a udržet určitý standard kvality v populaci [17].

4.2.3 Křížení

Vždy by měly být vybrány nejlépe padnoucí chromozomy podle procesu výběru EA. Je však možné, že se tyto chromozomy objeví v následujících generacích více než jednou, což povede k populaci, která se skládá výhradně z mnoha kopií stejné potenciální odpovědi. Neexistuje žádná záruka, že počáteční populace má globální optimální řešení nebo dokonce řešení, které je považováno za adekvátní pro řešený problém, takže pokud je počáteční populace příliš malá, může nastat problém. V této situaci povede EA k populaci, která se skládá výhradně z kopií optimálního řešení, které bylo původně nalezeno v počáteční populaci. Reprodukční procedury, též nazývané jako operátory křížení, byly vyvinuty, aby se vyhnuly tomuto omezení a jsou nyní nezbytnou součástí jakéhokoli EA pro efektivní vývoj populací směrem k optimálním řešením.

Nejběžnější používané operátory pro binární kódování jsou následující [18]:

- Jednobodové křížení
- Vícebodové křížení
- Uniformní křížení

Jednobodové křížení

V jednobodovém křížení jsou dva párující se chromozomy nebo vybraná rodičovská populace říznuty v bodě, který je náhodně vybrán a je známý jako pivot nebo bod křížení. V bodě řezu se genetická informace nalevo (nebo napravo) od bodu prohodí mezi dvěma rodičovský chromozomy, aby vznikly dva chromozomy potomků (děti) [18].

Obrázek 3 Jednobodové křížení
Zdroj: [18]

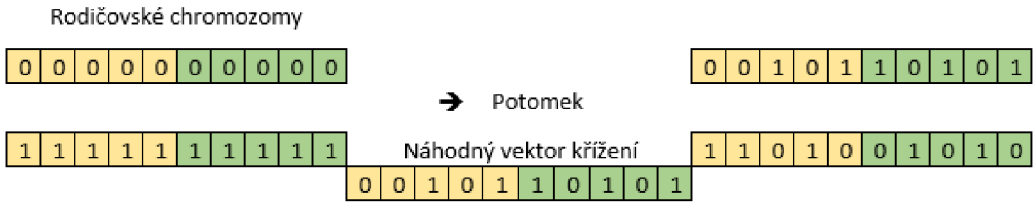
Vícebodové křížení

Vícebodové křížení funguje s více pivoty než jednobodové křížení. Původní rodičovské chromozomy jsou v důsledku toho vážněji narušeny. V případě výběru dvou míst křížení napravo od sudého (nebo lichého) počtu pivotů, jejich genetická informace se přepne, aby se vytvořili dva potomci [18].

Obrázek 4 Vícebodové křížení
Zdroj: [18]

Uniformní křížení

Při uniformním křížení je zkoumán vždy pouze jeden konkrétní gen. Generuje náhodný vektor křížení, který je vyplněn binárními hodnotami, kde 1 označuje výměnu konkrétní genové pozice mezi rodiči a 0 označuje zachování tohoto genu každým rodičem [18].



Obrázek 5 Uniformní křížení
Zdroj: [18]

4.2.4 Mutace

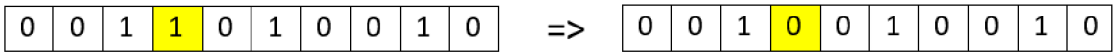
Mutace je obecně chápána jako malá, náhodná změna provedená na chromozomu za účelem vytvoření nového výsledku. Obvykle se používá s nízkou pravděpodobností a používá se k zachování a pestrosti genetické populace. Když je pravděpodobnost dostatečně vysoká, GA se zjednoduší na náhodné vyhledávání.

Mutace využívá mutační operátory, mezi které náleží [19]:

- Mutace prohozením bitu
- Náhodné resetování
- Mutace záměnou
- Mutace zamícháním
- Inverzní mutace

Mutace prohozením bitu

Tento druh mutace vybere jeden nebo více náhodných bitů a prohodí je. Tento druh operátoru je využíván pro binárně kódované GA.



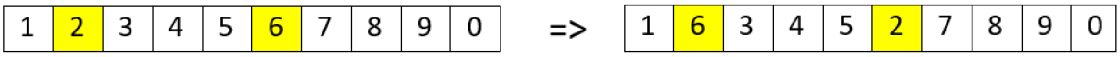
Obrázek 6 Mutace prohozením bitu
Zdroj: [19]

Náhodné resetování

Náhodné resetování je rozšířením mutace prohozením bitu pro celočíselnou reprezentaci, kdy se nejedná o binární hodnoty. V tomto případě je ze sady hodnot přidělena náhodná hodnota zvolenému genu.

Mutace záměnou

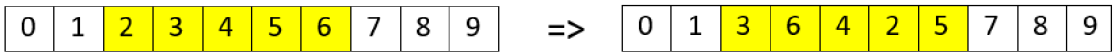
Mutace záměnou tvoří náhodný výběr dvou umístění chromozomu a prohození jejich hodnot. Toto je typické v kódování založeném na permutaci.



Obrázek 7 Mutace záměnou
Zdroj: [19]

Mutace zamícháním

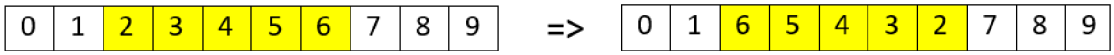
Permutační reprezentace je také často používána v mutaci zamícháním. To zahrnuje výběr genů z celého chromozomu a náhodné přeuspořádání nebo zakódování jejich hodnot.



Obrázek 8 Mutace zamícháním
Zdroj: [autor]

Inverzní mutace

V inverzní mutaci je zvolena podmnožina genů podobně jako u mutace zamícháním, ale místo náhodného přeuspořádání se celý výběr genů invertuje.



Obrázek 9 Inverzní mutace
Zdroj: [autor]

4.3 Genetické programování

Genetické programování je jedním z nejznámějších EA, které mohou automaticky vytvářet počítačové programy k řešení problémů. S ohledem na další EA vystupuje jako klíčový faktor genetického programování stromová reprezentace s proměnnou délkou.

Genetické programování je významnou metodou strojového učení, která využívá principy EA, jako je křížení, mutace a selekce, k nalezení efektivních řešení pro složité problémy. V průběhu evolučního procesu hledání optimálního řešení vytváří genetické programování populaci jedinců, kteří jsou reprezentováni jako programy. Tito jedinci jsou poté hodnoceni na základě své fitness funkce, která

určuje, jak dobře se daný program chová vzhledem k řešení daného problému viz 4.2 [10].

4.3.1 Stromová reprezentace

V genetickém programování je stromová reprezentace používána k vyjádření řešení problémů pomocí hierarchických struktur ve formě stromů. Každý jedinec v populaci je reprezentován jako strom, kde každý uzel představuje operátor nebo funkci, a listy stromu představují terminální symboly, jako jsou konstanty, proměnné nebo akce. Díky této stromové struktuře jsou schopni být vytvořeny složité programy nebo modely, které mají různé účely a vykonávají různé úkoly.

Jednotlivé uzly stromu zastupují operace, které se aplikují na data, a listy obsahují hodnoty, jež slouží jako vstupy pro tyto operace. Celková struktura stromu určuje, jak jsou operace a terminální symboly kombinovány, což vede ke vzniku výsledného řešení. Díky flexibilitě stromové reprezentace lze vytvářet různé kombinace operátorů a hodnot, což napomáhá nalezení optimálního řešení pro daný problém.

Délka a složitost jedinců v populaci mohou být různé, což je výhodné, zejména pokud se jedná o problémy s proměnlivou délkou řešení. V procesu evoluce jsou jedinci v populaci podrobováni operacím křížení a mutace, které upravují jejich stromovou strukturu. Křížení spočívá v kombinaci částí stromů od dvou rodičů, čímž vzniká potomek s prvky obou rodičů. Mutace pak zahrnuje náhodné změny stromové struktury, což umožňuje průzkum nových oblastí řešení a diverzifikaci populace.

Stromová reprezentace je klíčovým prvkem genetického programování a představuje silný nástroj pro tvorbu adaptivních programů a modelů. Díky této reprezentaci má genetické programování schopnost řešit rozmanité problémy, jako je optimalizace, strojové učení, evoluční návrh, symbolická regrese a další úkoly [20].

4.3.2 Lineární genetické programování

Lineární genetické programování je EA, který slouží k automatizovanému nalezení programů nebo modelů pro řešení různých úloh. Tato technika je součástí genetického programování, které spadá pod širší oblast EA.

V lineárním genetickém programování je jedinec reprezentován jako lineární řetězec genů, což je posloupnost instrukcí nebo symbolů, které společně tvoří program nebo model. Každý gen v řetězci kóduje určitou operaci, funkci nebo akci, která může být prováděna v rámci řešení daného problému.

Algoritmus lineárního genetického programování pracuje ve více generacích, přičemž v každé generaci probíhá proces selekce, křížení a mutace. Během selekce jsou vybíráni nejlepší jedinci z populace na základě jejich fitness hodnoty, která vyjadřuje kvalitu jejich řešení dané úlohy. Tito vybraní jedinci jsou pak kombinováni v procesu křížení, což vytváří nové potomky, kteří zdědí části svých rodičů. Mutace náhodně mění některé geny v genetickém kódu, což přispívá k diverzifikaci populace a průzkumu nových oblastí řešení.

Cílem lineárního genetického programování je najít nejlepší řešení problému v podobě optimálního programu nebo modelu, který dosahuje požadovaného výstupu pro daný vstupní soubor dat.

Lineární genetické programování poskytuje schopnost hledání efektivních a adaptivních programů, které jsou schopné řešit různé typy úloh, jako je symbolická regrese, logické výrazy, strojové učení, evoluční návrh a mnoho dalších.

Díky své jednoduché lineární reprezentaci a schopnosti pracovat s rozmanitými úlohami, je lineární genetické programování široce využíváno v oblasti automatizovaného strojového učení, evolučního návrhu algoritmů, symbolického strojového učení a dalších aplikací [21].

4.4 Příklad genetického algoritmu

```
import random
# Parametry genetického algoritmu
POPULATION_SIZE = 100
CHROMOSOME_LENGTH = 50
GENERATIONS = 100
MUTATION_RATE = 0.1

# Definování funkce fitness
def fitness_function(chromosome):
    # Vypočítání fitness jako součet hodnot chromozomů
    return sum(chromosome)

# Vygenerování počáteční populace
def generate_population():
    population = []
    for _ in range(POPULATION_SIZE):
        chromosome = [random.randint(0, 1) for _ in
range(CHROMOSOME_LENGTH)]
        population.append(chromosome)
    return population

# Jednobodové křížení mezi dvěma rodiči
def crossover(parent1, parent2):
    crossover_point = random.randint(1, CHROMOSOME_LENGTH -
1)
    child1 = parent1[:crossover_point] +
parent2[crossover_point:]
    child2 = parent2[:crossover_point] +
parent1[crossover_point:]
    return child1, child2
```

```
# Mutace prohozením bitu v chromozomu
def mutate(chromosome):
    mutated_chromosome = []
    for gene in chromosome:
        if random.random() < MUTATION_RATE:
            mutated_chromosome.append(1 - gene)  # Prohození
bitu
        else:
            mutated_chromosome.append(gene)
    return mutated_chromosome

# Výběr rodičů pomocí turnajového výběru
def tournament_selection(population):
    tournament_size = int(POPULATION_SIZE * 0.1)  # Výběr
nejlepších 10 % jako kandidátů
    candidates = random.sample(population, tournament_size)
    best_candidate = max(candidates, key=fitness_function)
    return best_candidate

# Provedení genetického algoritmu
def genetic_algorithm():
    population = generate_population()

    for _ in range(GENERATIONS):
        new_population = []

        while len(new_population) < POPULATION_SIZE:
            # Turnajový výběr
            parent1 = tournament_selection(population)
            parent2 = tournament_selection(population)

            # Křížení
            child1, child2 = crossover(parent1, parent2)
```

```
            # Mutace
            child1 = mutate(child1)
            child2 = mutate(child2)

            new_population.append(child1)
            new_population.append(child2)

        population = new_population

# Nalezení nejlepšího jedince v konečné populaci
    best_individual = max(population, key=fitness_function)
    best_fitness = fitness_function(best_individual)

    return best_individual, best_fitness

# Spuštění genetického algoritmu a výsledky
best_solution, best_fitness = genetic_algorithm()
print("Nejlepší řešení:", best_solution)
print("Nejlepší fitness:", best_fitness)
```

Ukázka kódu 1 Příklad genetického algoritmu
Zdroj: [autor]

Kód začíná definováním parametrů pro GA, včetně velikosti populace, délky chromozomů, počtu generací a rychlosti mutace. Je definována fitness funkce, která vypočítá fitness chromozomu jako součet jeho hodnot. Tato funkce představuje cíl, který má být maximalizován. Funkce `generate_population()` generuje počáteční populaci náhodných binárních chromozomů. Funkce `crossover()` provádí jednobodové křížení mezi dvěma rodičovskými chromozomy v náhodně zvoleném bodu křížení. Vytváří dva potomky. Funkce `mutate()` náhodně převrací bity v chromozomu na základě rychlosti mutace. Funkce `tournament_selection()` provádí turnajový výběr rodičů pro reprodukci. Náhodně vybírá podmnožinu populace jako kandidáty a vybírá nejlepšího kandidáta na základě fitness. Hlavní funkce `genetic_algorithm()` inicializuje populaci a poté iteruje po zadaný počet

generací. V rámci každé generace vytváří novou populaci prováděním operací selekce, křížení a mutace. Proces pokračuje, dokud není dosaženo zadaného počtu generací.

4.5 Výhody a nevýhody genetických algoritmů

Ve srovnání s konvenčními optimalizačními algoritmy mají GA několik výhod. Dvě nejvýznamnější jsou paralelnost a schopnost řešit složité problémy. Ať už je cílová fitness funkce lineární, nelineární, spojitá, nespojitá nebo podléhá náhodnému šumu GA, dokážou si poradit s řadou optimalizačních problémů. Několik potomků v populaci se chová nezávisle, což umožňuje populaci současně prozkoumat různé oblasti. Algoritmy lze díky této schopnosti implementovat paralelně. Je možná současná manipulace s několika parametry, a dokonce i s více skupinami kódovaných řetězců.

GA mají také významné nevýhody. Například nevhodné využití následujících funkcí: Konstrukce fitness funkce, použití velikosti populace, výběr klíčových faktorů včetně rychlosti mutace, křížení a výběrová kritéria pro novou populaci. Jakékoli nesprávné rozhodnutí buď zabrání konvergenci algoritmu, nebo povede k nepoužitelným výsledkům. GA jsou i přes tyto nevýhody jednou z nejpoužívanějších optimalizačních technik v současné nelineární optimalizaci [22].

5 Teorie her

Formální studie rozhodování v situacích, kdy několik hráčů musí učinit rozhodnutí, která mohou mít dopad na ostatní hráče, se nazývá teorie her. Podle principů teorie her si každý hráč zvolí strategii, jak čelit tahům ostatních hráčů způsobem, který by maximalizoval jeho vlastní zisk.

Hry lze rozdělit do dvou kategorií: nekooperativní a kooperativní hry, podle toho, jak hráči interagují [23].

- Hry hrané samostatně (takzvané nekooperativní hry) jsou například Nashova rovnováha a Stackelberg [24]. V nekooperativních hrách si každý hráč optimalizuje svůj osobní zisk na základě znalostí o akcích ostatních. Hráči se neshodnou na nejlepších společných akcích skupiny. Hráči nejsou zapojeni do žádné komunikace před hrou.
- Spolupráce ve hrách je hlavním aspektem kooperativních her. Příkladem je koncept Paretova optima [12]. V kooperativních hrách se hráči mohou účastnit fáze komunikace před hrou, mohou také hledat přiměřenou odměnu, která zajistí stabilitu koalic a mohou využívat další funkce k udržení koalic. Hráči hledají společné akce, které mohou proměnit skupinu na optimální.

Byly vytvořeny vědecké vztahy mezi teorií her a evolucí, přičemž EA byly inspirovány Darwinovskou evoluční teorií. Lewontin původně navrhoval spojení mezi evoluční biologií a teorií her [25]. Strategie „maxmin“ (nejlepší míra přežití, když příroda udělá to nejhorší) sleduje taktiku, která minimalizuje nebezpečí vyhynutí. Tato strategie byla použita k zobrazení evoluce genetického mechanismu jako hry mezi dvěma hráči: druhů a přírody. Později byly vyvinuty další nápady, jak poznamenal Maynard Smith [26]. Studium složitých interakcí mezi biologickými entitami bojujícími o zdroje bylo navrženo pomocí evoluční teorie her.

Evoluce je vnímána jako hra, ve které organismy usilují o přežití tím, že získávají rysy (taktiku) od svých rodičů. Evoluční teorie her je spojena s Riechmannovou teorií učení GA [27], které poznamenává, že proces GA učení poskytuje Nashovu rovnováhu, která je blízká. V roce 1983 Vincent navrhl použití

teorie her jako nástroje pro návrh. Použití EA/metaheuristiky jako praktických optimalizačních nástrojů pro multidisciplinární optimalizaci návrhu bylo zahrnuto do důkladného přezkoumání a analýzy kooperativní/nekooperativní formulace stejně jako hierarchických/nehierarchických dekompozičních přístupů Lewis a Mistree [28].

EA a teorie her spadají do dvou kategorií: 1. Myšlenky teorie her, které byly začleněny do paradigmatu EA buď za účelem řešením různých typů problémů s optimalizací, nebo pro zvýšení výkonu v řadě aplikací. 2. Problémy teorie her byly řešeny pomocí EA [23].

5.1 Historický vývoj

R. A. Fisher jako první vytvořil evoluční teorii her ve snaze vysvětlit stejný poměr pohlaví u savců [29]. Proč je poměr pohlaví zhruba stejný u mnoha druhů, kde se většina samců nikdy nerozmnožuje, byla hádanka, kterou se snažil Fisher vyřešit. Nepářící se samci těchto druhů by se zdáli být zbytečnými figuranty, které nosí zbytek populace. Fisher došel k závěru, že individuální zdatnost závisí na genderové distribuci populace, pokud je měřená z hlediska očekávaného počtu vnoučat. Muži mají vyšší individuální zdatnost, když je v populaci více žen. Ženy pak opačně mají vyšší individuální zdatnost, když je v komunitě více mužů. Podle Fishera v takovém scénáři evoluční dynamika způsobí, že se poměr pohlaví ustálí na stejném počtu samců a samic. Skutečnost, že fyzická zdatnost člověka je ovlivněná podílem mužů a žen v populaci, dodává evoluci strategický element.

Nejprve se věřilo, že problém výběru rovnováhy tradiční teorie her lze vyřešit pomocí evoluční teorie her. Pro jakoukoli hru s konečným počtem hráčů a strategií, za předpokladu, že byly povoleny smíšené strategie, měla Nashova rovnováha významné nevýhody. Kromě toho, že se ne vždy zdálo, že odpovídá rozumnému výsledku a občas je v rozporu s intuicí lidí o tom, co by mělo být považováno za racionální výsledek, nebyla Nashova rovnováha vždy zaručena jako jedinečná.

Brzy se zjistilo, že strukturálně identické problémy s výběrem rovnováhy existují také v evoluční teorii her. Bylo předloženo několik soupeřících definic evoluční stability, z nichž každá má jinou hodnotu. Bylo objeveno, že existuje shoda

mezi statickými představami o evoluční stabilitě s dynamickou stabilitou. Dynamické modely evoluční teorie her přinesly výsledky, které byly z pohledu tradiční teorie her iracionální, jako je přetrvávání přísně ovládaných strategií [30].

5.2 Dva přístupy k evoluční teorii her

K evoluční teorii her lze přistupovat dvěma různými způsoby. První metoda využívá myšlenku evolučně stabilní strategie jako svůj hlavní analytický nástroj a je založena na práci Maynarda Smithe a Price [26]. Druhá metoda vytváří explicitní model metody, kterou se mění frekvence strategií populace, a zkoumá aspekty evoluční dynamiky modelu.

První strategii lze tedy považovat za nabízející statickou koncepční analýzu evoluční stability. „Statická“, protože navzdory nabízeným definicím evoluční stability často nebere v potaz základní mechanismus, kterým se chování (nebo strategie) v populaci mění. Druhá metoda se naproti tomu nesnaží vytvořit koncept evoluční stability. Místo toho využívá všechny tradiční myšlenky stability pro analýzu dynamických systémů poté, co byl vytvořen model populační dynamiky [30].

5.2.1 Definice evoluční stability

Nashova rovnováha je primární myšlenkou řešení v teorii her. Jedná se o profil strategií (přiřazení strategií každému hráči), ve kterém neexistuje žádná motivace pro hráče, aby se odchýlil od strategie, kterou si zvolil.

Pro pochopení pojmu evoluční stability mimo rámec klasického konceptu herního řešení Nashovy rovnováhy je hra v tabulce 1. Ve strategiích existují dva body Nashovy rovnováhy: (S_1, S_1) a (S_2, S_2). Nashova rovnováha poskytuje možnost, že hráč, který se odchýlí od své rovnovážné strategie, získá stejnou odměnu, protože se jedná o soubor vzájemně nejlepších odpovědí, takže žádný hráč nemůže zvýšit svoji odměnu použitím jiné strategie. Rovnováha (S_2, S_2) je toho příkladem. Protože počítá s možností odklonu od rovnováhy, a nakonec vede k nahrazení existující strategie, Nashova rovnováha je nedostatečná pro evoluční stabilitu.

Pokud by se vyvinul mutant se strategií S_1, odměna mutanta by byla stejná jako odměna populace jako celku, proto by na mutanta nebyl žádný selekční tlak.

Pokud by se objevil druhý mutant, výhoda ze spojení $S_1 - S_1$ by pro oba vytvořila zdatnost, která by byla vyšší než průměrná zdatnost populace. To by umožnilo mutantovi S_1 se rozšířit, a nakonec převládnout ve zbytku populace.

	S_1	S_2
S_1	**(2, 2)**	(1, 1)
S_2	(1,1)	(1, 1)

Tabulka 1 Nashova rovnováha
Zdroj: [autor]

Jednostranná změna hráčovy rovnovážné strategie způsobí, že se tento hráč v Nashově rovnováze zhorší. Lze demonstrovat, že striktní Nashova rovnováha je příliš silná na to, aby zachytila myšlenku evoluční stability. Nashova rovnováha zachycuje jeden význam evoluční stability (lze ji považovat za typ „lokálního optima“) [30].

5.3 Herní principy

Podle mnoha měřítek je herní sektor větší než Hollywood. Konkurenční videoherní byznys je obrovský. Jednotlivci tráví spoustu času a peněz hraním her a někteří dokonce platí za sledování ostatních, jak je hrají. Gamifikace je přítomna všude – v podnikatelské kultuře, vzdělávání a v inovačním sektoru [31].

Klíčové herní principy, které jsou důležité pro vývoj her jsou například [32]:

- Herní mechaniky: systémy a zákony, které řídí, jak se hra chová. Tyto herní mechanismy řídí, jak hráči interagují s okolním světem, dosahují cílů a postupují ve hře. Úspěšné herní mechanismy by měly být snadno pochopitelné, snadno uchopitelné a obtížně zvládnutelné.
- Herní rovnováha: označuje se jako relativní obtížnost hry. Zatímco příliš náročná hra může být frustrující a odrazující, hra, která je příliš snadná, rychle ztratí zájem hráče. Pečlivé zvážení herních mechanismů, hráčských schopností a designu úrovní je nezbytné pro dobrou rovnováhu hry.
- Uživatelská zkušenost: popisuje, jak se hráč obecně cítí při zapojení do hry. Uživatelský dojem by měl být poutavý, přirozený a vizuálně přitažlivý. Tento

design znamená vzít v úvahu cestu hráče hrou, vytvářet uživatelsky přívětivá rozhraní, podmanivé obrázky a zvuk.

- Herní příběh: popisuje děj hry a prvky budování světa. Poutavý příběh může zlepšit zážitek hráče a podpořit silnější emocionální pouto ke hře. Scény, dialogy nebo vyprávění o prostředí – to vše jsou způsoby, jakými mohou videohry zprostředkovat příběhy.
- Hráčské zastoupení: popisuje schopnost hráče činit významná rozhodnutí ve hře. Nabídnout hráčům možnost volby může zvýšit jejich emocionální investici a úroveň zapojení. Tím, že umožní hráčům vytvářet své postavy nebo nabízí různé způsoby plnění úkolů, mohou herní návrháři dát hráčům zastoupení.
- Pokrok ve hře: neboli postup hráče hrou. Získávání bodů, odemykání nových schopností nebo plnění úkolů, to vše může vést k postupu. Uspokojivý herní pokrok by měl hráčům přinést pocit úspěchu a povzbudit je, aby pokračovali ve hře.
- Znovuhratelnost: schopnost hry pro opakované hraní. Dlouho po úvodním rozehrání budou mít hráči pouze zájem o hru, která je znovu hratelná. Přidáním náhodných faktorů, vytvářením četných konců nebo generováním úkolů, které lze vyřešit mnoha způsoby, mohou herní návrháři učinit své hry znovu hratelnějšími.
- Průběh hry: tempo a rytmus hry. Zatímco hra se špatným průběhem se může zdát nudná, hra s dobrým průběhem může být vzrušující a poutavá. Pečlivý design úrovní, výzvy a vyvážení obtížnosti mohou pomoci vytvořit plynulou hru.

Pro vytváření her, které jsou pro hráče zajímavé, obtížné a zábavné, jsou herní principy zásadní. Herní designéři mohou vytvářet hry, které jsou poutavé, obohacující a zapamatovatelné tím, že vezmou v úvahu pravidla herní mechaniky, rovnováhy, uživatelské zkušenosti, příběhu, hráčského zastoupení, pokroku, znovuhratelnosti a průběhu hry [32].

5.4 Evoluční přístup ke hře dáma

Evoluční přístup ke hře dáma uvádí metody genetické evoluce lineární a nelineární pro vyhodnocovací funkce ve hře dáma.

Výběr evolučního přístupu ke hře dáma demonstruje univerzálnost EA v kontextu různých herních scénářů. Dáma jako komplexní strategická hra nabízí mnoho možností pro aplikaci evolučního přístupu a představuje výzvu, kterou je možné řešit různými metodami, včetně hledání optimální strategie nebo vývoje hráčů UI.

Mezi přístupy použité k EA ve hře dáma náleží [33]:

- Komponenty hodnotících funkcí
- Typy heuristiky
- Evoluční proces a výsledky
- Porovnání heuristik

5.4.1 Komponenty hodnotících funkcí

Osm základních charakteristik. Počty kamenů (1) a dam (2). Počet bezpečných figurek nebo figurek, které jsou blízko okraje šachovnice, včetně kamenů (3) a dam (4). Počty pohyblivých nebo schopných provést jiný tah než zajetí kamenů (5), nebo dam (6). Celková vzdálenost pěšců k postupové čáře (7) a počet volných polí na povýšení (8).

Jedenáct prvků návrhu. Celkový počet figurek obránce (9), což jsou kameny a dámy dohromady ve dvou nejnižších řadách. Celkový počet útočících kamenů umístěných ve třech nejvyšších řadách (10). Počet kamenů (11) a dam (12), které jsou umístěny uprostřed na osmi středových polích herního plánu. Kameny (13) a dámy (14) leží na hlavní diagonále a (15), (16) na dvojité diagonále. Množství osamocených kamenů (17) a dam (18) (osamocená figurka je ta, která kolem sebe nemá žádnou další figurku). Množství prázdných polí (19) (prázdná políčka sousedící s alespoň třemi figurkami stejné barvy).

Šest charakteristik vzoru. Každý vzor je vysvětlen z pohledu bílého hráče. Funkce (20) až (25) mohou přijímat pouze hodnoty boolean, protože každý hráč na herním plánu může mít vždy pouze jednu instanci každého vzoru [33].

5.4.2 Typy heuristiky

Pro zjištění, zda hodnoty příslušných parametrů pro oba hráče v lineární heuristice vykazují symetrii, byly provedeny testy. Bylo zjištěno, že váhy spojené s počty kamenů a dam jsou nejvíce nevyvážené. Použití variací v příslušných hodnotách parametrů symetrie by pomohlo snížit počet genů. Pokud jde o dva asymetrické parametry, bylo navrženo, že asymetrie je způsobena vzájemnými závislostmi mezi parametry. Tato závislost může negativně ovlivnit kvalitu heuristiky. Lineární a nelineární heuristika byla definována [33].

Lineární heuristika

Osm faktorů (8 F). Tato heuristika brala v potaz rozdíly mezi prvními osmi charakteristikami zmíněné v kapitole 5.4.1.

Deset faktorů (10 F). Tato heuristika vzala stejné parametry jako 8 F. Původní testy symetrie ukázaly, že by měl být brán v potaz hrubý počet kamenů a dam pro každého hráče nezávisle.

Patnáct faktorů (15 F). Všechny faktory této heuristiky byly specifikovány jako rozdíly mezi vypočtenými hodnotami pro každý parametr na každé straně. V úvahu byly brány následující funkce: (1)-(4), (9)-(12), (19)-(25).
Devatenáct faktorů (19 F). Všechny složky této heuristiky byly ve formě rozdílů mezi příslušnými hodnotami vypočtenými pro oba hráče. Zohledňuje všechny parametry (1)-(19).

Dvacet pět faktorů (25 F). Tato heuristika zohledňovala variace hodnot všech relevantních parametrů [34].

Nelineární heuristika

Základní zásadou nelineární heuristiky je, že se ukázalo jako výhodné rozdělit hru do fází a použít různé heuristiky pro každou fázi. Bylo zde několik klíčových bodů, které vyžadovaly úpravu funkce hodnocení. Jedním z nich byla přítomnost dam, která bohatě demonstrovala, jak sofistikovaná hra je. V pozicích na konci hry může být nutné použít i jiné heuristiky. GA byl podle některých provedených testů významně ztížen zavedením nelineárních komponent s podmínkami, které nebyly disjunktní. Bylo to proto, že vícenásobné překrývající

se podmínky umožňovaly dosažení velmi podobných výsledků mnoha způsoby, pokaždé s velmi odlišnými hodnotami parametru. Proto byla hra rozdělena do samostatných fází.

Tři fáze (3Ph). Pro tuto heuristiku byla hra rozdělená do tří fází. První fáze začátek: Na herním plánu nejsou žádné dámy a každý hráče má více než tři kameny. Druhá fáze dámy: Oba hráči mají alespoň jednu dámu a více než tři figurky. Třetí fáze finální: Jeden nebo oba hráči mají tři a méně figurek. Lineární heuristika v každé fázi zohledňovala variace ve stejných osmi parametrech jako heuristika 8 F.

Expertní tři fáze (E3Ph). Hra byla rozdělena do fází pomocí této heuristiky, stejně jako tomu bylo u (3Ph). Každá ze tří fází lineární heuristiky zohledňovala variace (bílá vs. černá) v následujících deseti vlastnostech: (3), (10)-(12), (16), (20)-(23) a (25). Na základě výsledků počátečních běhů generátoru byla vybrána výše zmíněná sada parametrů, přičemž ty nejdůležitější byly upřednostněny.

V 3Ph i E3Ph byly parametry týkající se dam brány v úvahu pouze v případě, že byly na šachovnici dámy přítomné [34].

5.4.3 Evoluční proces a výsledky

Pro generování koeficientů a1, …, aj. výše uvedených lineárních kombinací byly použity GA. Každý heuristický koeficient byl reprezentován jako vektor skutečných celých čísel, kde každé číslo představovalo konkrétní gen. V případě nelineární heuristiky evoluce nezměnila okolnosti, za kterých byly nelineární komponenty tvořeny. Populaci tvořilo 300-400 jedinců. Každá fáze zaznamenala 75 % - 80 % regeneraci nejslabších členů populace.

Pro výběrové řízení byly použity turnaje. Pro každý turnaj byl náhodně vylosován určitý počet exemplářů z populace. Aby se rozhodlo, kdo soutěž vyhraje, byla porovnána jejich hodnocení zdatnosti. Vítězové dvou takových soutěží byli spojeni a poté se zapojili do křížení. Velikost turnaje mezi vzorky byla stanovena mezi dvěma a pěti, v závislosti na tom, kolik genů má každý vzorek.

Každá nelineární složka a základní heuristická funkce byly zkříženy samostatně pro každou dvojici odpovídajících lineárních kombinací v heuristice. Genotyp každé lineární kombinace byl náhodně rozdělen do dvou skupin. Potomci dostávají zděděné hodnoty každé části genotypu od odpovídajícího rodiče. Interval

určený hodnotami tohoto genu v rodičovských vzorcích byl použit k náhodnému výběru hodnoty genu, na který bylo dělení umístěno. Nejslabší vzorek v populaci byl nahrazen pouze v případě, že by hodnocení zdatnosti nově vygenerovaného potomka bylo vyšší než u vzorku, který má být nahrazen. Poměr efektivních křížení k potencionálním byl v průběhu procesu uveden mezi 80 % a 90 %. Efektivní křížení bylo to, ve kterém byl výsledný vzorek skutečně přidán do populace. Geny každého nově vytvořeného vzorku podstoupily jednu ze tří typů mutací: násobení dvěma, dělení dvěma nebo změně znaku genu.

Pomocí generátoru heuristiky byla hra rozdělena do několika samostatných úrovní. Během první fáze algoritmu byla získána heuristika, která by mohla přesně vyhodnotit scénáře blízko konce hry. Aby toho bylo dosaženo, bylo vyhodnoceno několik náhodně generovaných pozic pomocí alfa-beta přístupu bez použití jakékoli heuristiky. Místa, která byla hlubší než hloubka alfa-beta algoritmu, byla považována za remízu. Správnost každého vzorku byla určena pomocí vzorce

$$n/\sum(h_i - a_i)^2,$$

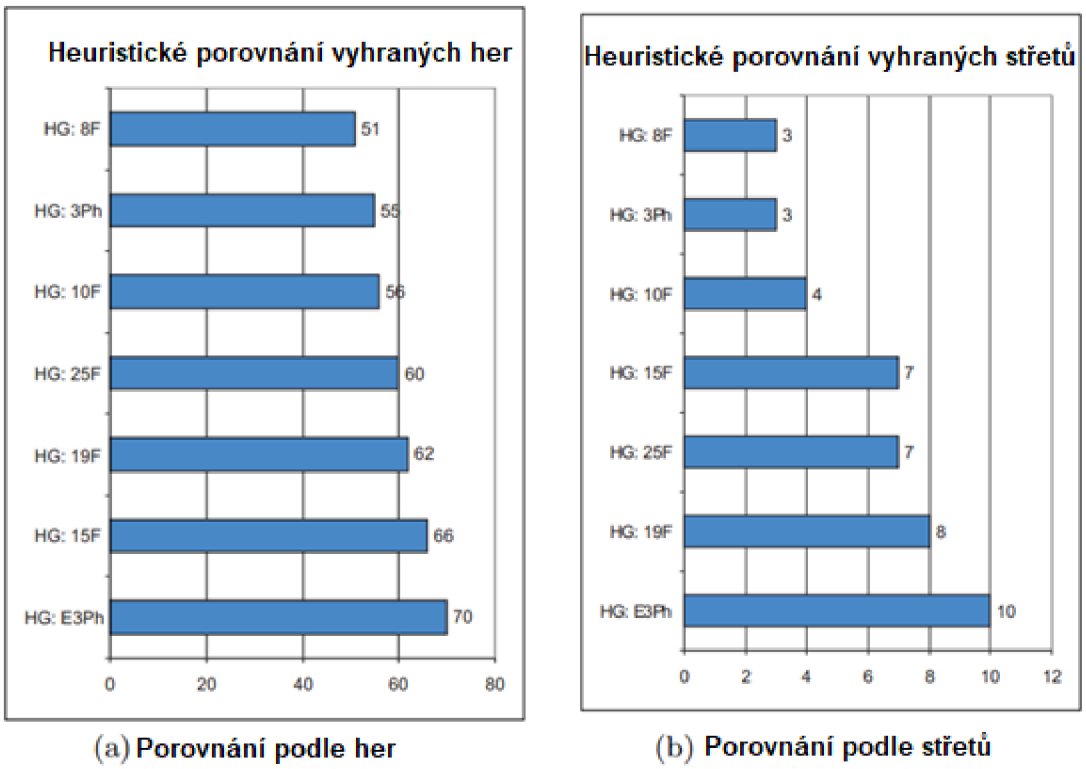
kde n představuje počet testovacích okolností, h_i představuje vyhodnocení i-té testovací situace heuristickým vzorkem a a_i představuje alfa-beta algoritmus.

Po skončení počáteční fáze byly přidány nové náhodné vzorky, které nahradily nejhůře vyhovující podíl populace. Nejvhodnější vzorek z předchozí fáze sloužil jako heuristická funkce algoritmu alfa-beta, protože generoval nové scénáře a vyhodnocoval je. Tento postup trval, dokud nebylo dosaženo základní pozice. Pozice dosažené v první fázi byly brány v úvahu po 82 až 86 krocích. Tento interval byl stanoven pomocí experimentů, které vypočítaly typický počet tahů potřebných k vítězství ve hře pomocí náhodných tahů. Počet pozic v dalších fázích byl brán v úvahu 3000 až 4500 pozic [35].

5.4.4 Porovnání heuristik

Pro posouzení relativních předností heuristik byla vytvořena soutěž. Každá heuristika soutěžila v 10 hrách proti ostatním, přičemž strany se po každé hře vyměnily. Hry hrané mezi libovolnými dvěma algoritmy byly párově odlišné, protože alfa-beta implementace zahrnovala určitou náhodnost. Byly použity dvě kategorie. První kategorie byla založena na celkovém počtu her, které každá

heuristika vyhrála, remizovala nebo prohrála. Druhá kategorie brala v úvahu výsledky série 10 her. Heuristika získala 2 body za vítězství a 1 bod za remízu.



(a) Porovnání podle her (b) Porovnání podle střetů

Obrázek 10 Porovnání heuristik
Zdroj: [33]

Bez ohledu na to, která srovnávací kritéria byla vzata v úvahu, heuristika E3Ph prokázala, že funguje lépe než kterákoli jiná. Očekávalo se, že propracovanější heuristika si povede lépe v konkurenci při zohlednění složitějších kritérií [33].

5.5 Zlepšování her používáním evolučních algoritmů

Videohry jsou složité systémy se spoustou vzájemně propojených částí. EA se ve videohrách používají k optimalizaci systémů, aby hráči měli celkově lepší herní zážitek. EA lze za tímto účelem použít v různých metodách. Jednou z možností je použití EA ke zlepšení herních mechanismů. Tvůrci her mohou vytvářet zajímavější a vyváženější hry pomocí EA. Tyto algoritmy lze například použít k úpravě úrovně obtížnosti hry. EA mohou určit ideální úroveň obtížnosti pro hru, která je náročná, testováním různých úrovní obtížnosti a sledováním reakce hráčů.

EA lze také použít ve videohrách ke zlepšení chování UI. Akty a volby provedené počítačem řízenými postavami se ve hře označují jako chování UI. Pomocí EA k optimalizaci chování UI mohou vývojáři ztížit hráčům protivníky. Například EA mohou být použity ve střílečkách z pohledu první osoby k optimalizaci chování UI protivníka. Mohou určit ideální chování pro protivníky, které je těžké porazit, zkoušením různých vzorců chování a pozorováním, jak hráči reagují. Optimalizace herních prostředků je třetí způsob, jak lze EA použít pro videohry.

Vizuální a zvukové složky hry, jako jsou modely postav, textury a zvukové efekty, jsou známé jako herní aktiva. Tvůrci her mohou vytvářet poutavější a esteticky příjemnější hry pomocí EA k optimalizaci herních materiálů. EA lze použít například ke zlepšení osvětlení hry. EA mohou určit ideální osvětlení pro hru, která vytváří požadovanou atmosféru, testováním různých konfigurací osvětlení a sledováním reakce hráčů. Tvůrci videoher musí nejprve identifikovat problém, který se snaží vyřešit a teprve potom mohou použít EA k jeho vyřešení. To znamená specifikaci cílů hry nebo určení potenciálních oblastí pro zlepšení.

Funkce fitness hodnotí, jak dobře systém funguje. Měla by být vytvořena tak, aby podporovala pozitivní chování a trestala negativní chování. Například funkce fitness může hráče odměnit za sbírání mincí ve hře, kde je cílem sbírat mince a penalizovat hráče za ztrátu životů. Vývojáři mohou využít EA k optimalizaci systému, jakmile je identifikována funkce fitness a problém. Funkce fitness se používá k vyhodnocení populace potenciálních řešení. Za účelem vytvoření nové populace kandidátských řešení jsou poté použita ta nejlepší řešení. K nalezení nejlepší funkční odpovědi se tento proces znovu opakuje. GA, evoluční metody a genetické programování jsou některé z EA, které lze použít ve videohrách [36].

5.6 Umělá inteligence ve hrách

Herní byznys prošel proměnou díky UI, která zlepšila uživatelské zkušenosti a oživila virtuální světy. EA se staly silným nástrojem pro vytváření inteligentních a adaptivních her v kontextu UI ve videohrách. EA optimalizují a rozvíjejí řešení komplikovaných problémů tím, že čerpají inspiraci ze základních principů přirozené evoluce. Tyto algoritmy umožňují vytvářet a zlepšovat herní chování a strategie, což vede k dynamickému a obtížnému hraní. Vytváření inteligentních protivníků je

jedním z hlavních použití EA ve hrách. EA umožňují NHP rozvíjet své rozhodovací schopnosti v průběhu času, spíše než se spoléhat pouze na předem stanovená pravidla nebo vlastní chování.

Nejprve se vytvoří populace NHP s různými vlastnostmi nebo taktikou. Nejvhodnější NHP jsou vybrány tak, aby vytvořily následující generaci prostřednictvím soutěžení mezi nimi. Nové chování se objevuje jako výsledek procesů, jako je mutace a křížení. Populace se mění, aby se stala konkurenceschopnější a přizpůsobivější. EA byly použity v herní UI k vytvoření protivníků, kteří se mohou učit od akcí hráčů a přizpůsobovat se jim, což umožňuje zajímavé hraní, které přináší nové výzvy. Pocit nepředvídatelnosti a vzrušení vytváří schopnost těchto protivníků používat inovativní a nápadité techniky, které by lidští hráči nepředpokládali.

EA lze také použít ke zlepšení herních mechanismů a nastavení. Herní designéři se setkávají s problémem vyvážení herních komponent, jako je obtížnost, schopnosti postav nebo distribuce zdrojů. EA poskytují automatizovaný a iterativní způsob úpravy těchto parametrů. Algoritmy mohou hledat nejlepší nastavení parametrů, která maximalizují zábavu nebo férovost tím, že problém zformulují jako optimalizační úlohu a navrhnou vhodné fitness funkce.

Procedurální generování obsahu je další oblastí, ve které EA vynikají. V algoritmech generování procedurálního obsahu se herní materiál, jako jsou úrovně, mapy, hádanky nebo dokonce postavy automaticky generují pomocí přístupů UI. EA mohou vyvíjet obsah, který splňuje konkrétní kritéria návrhu tím, že specifikují sadu pravidel, omezení a cílů. Díky závislosti na ručně vyráběném materiálu a schopnosti vytvářet rozsáhlé a rozmanité herní světy, nabízí tato metoda hráčům jedinečné, znovu hratelné zážitky. Potenciál herní UI se dále zvyšuje kombinací EA s jinými metodami UI, jako jsou neuronové sítě. Tvůrci her mohou vyvíjet neuronové sítě, které řídí chování NHP kombinací EA se strukturami neuronových sítí. Tato metoda, známá jako neuroevoluce, umožňuje NHP plnit náročné úkoly nebo se učit z hráčských interakcí tím, že jim umožňuje adaptovat a měnit jejich techniky prostřednictvím evolučních procesů.

V oblasti UI ve videohrách se objevily EA jako účinný nástroj. Umožňují navrhovat adaptivní a inteligentní hry, ve kterých protivníci v průběhu času mění

svou taktiku a chování. EA také usnadňují optimalizaci nastavení hry a vytváření generování obsahu, což hráčům poskytuje řadu zajímavých zážitků. Potenciál herní UI lze prozkoumat novými způsoby kombinací EA s dalšími metodami UI, což dále posouvá hranice inteligentního hraní [37].

6 Návrh a realizace ukázkového řešení

EA je založen na principu genetického programování. Hlavním cílem algoritmu je evolvovat populace hráčů, kteří se učí navigovat ve stanoveném prostředí a překonávat překážky za účelem dosažení cíle.

V algoritmu se vytváří populace hráčů, přičemž každý jedinec je reprezentován genetickým kódem ve formě seznamu genů. Genetický kód určuje chování hráčů v prostředí a zahrnuje informace o jejich pohybu a rozhodování.

Algoritmus pracuje ve více generacích, přičemž každá generace reprezentuje jednu epochu evoluce. V každé generaci probíhá proces selekce, křížení a mutace, které jsou základem evolučního procesu. Během selekce jsou vybíráni jedinci s vyšší fitness hodnotou, což jsou hráči, kteří dosahují lepších výsledků a jsou nejblíže k cíli. Tito vybraní jedinci mají větší šanci být zahrnuti do procesu křížení, kde dochází ke kombinaci jejich genetického materiálu. Křížení spočívá v kombinaci genetického kódu dvou rodičů s cílem vytvořit nového potomka, který zdědí některé charakteristiky obou rodičů. To přispívá k diverzifikaci populace a hledání nových a lepších řešení. Mutace je dalším důležitým prvkem evolučního procesu, který náhodně modifikuje genetický kód jedinců. Tímto způsobem se do populace zavádějí nové varianty genetického kódu, což umožňuje průzkum nových oblastí prostoru řešení a vyhledávání optimálních kombinací genů.

Fitness funkce je klíčovým prvkem algoritmu, který hodnotí výkon jedinců na základě jejich schopnosti dosahovat cíle a překonávat překážky. Fitness funkce poskytuje zpětnou vazbu pro selekci a výběr lepších jedinců do další generace.

V průběhu evolučního procesu hráči procházejí postupným zlepšováním a učením se, jak se efektivněji přibližovat k cíli a vyhýbat se překážkám. Cílem algoritmu je nalézt optimální kombinaci genů, která umožní hráčům úspěšně a efektivně navigovat ve stanoveném prostředí.

Výsledkem EA je populace hráčů, kteří jsou schopni dosáhnout cíle. Algoritmus je schopen řešit problémy navigace a překonávání překážek v různých prostředích.

6.1 Hardware počítače využitého k experimentům

Pro provádění experimentů a implementaci EA je využit počítačový systém s následujícím hardwarovým vybavením:

- Procesor: Intel Core i7-10700KF, 8 jader / 16 vláken, frekvence 3,8 GHz s maximální frekvencí 5,1 GHz
- Operační paměť: 32 GB DDR4, frekvence 3000 MHz
- Grafická karta: NVIDIA GeForce GTX 1060, 6 GB GDDR5
- Úložiště: SSD Samsung 970 EVO Plus, kapacita 2 TB, rychlost čtení/zápisu až 3500/3300 MB/s
- Operační systém: Windows 10 Pro, verze 21H2

Tento hardware poskytuje dostatečný výpočetní výkon a grafickou akceleraci pro provedení experimentů s EA v herních situacích. Grafická karta umožňuje vizualizaci dat a tvorbu teplotních map pro analýzu chování virtuálních hráčů v různých scénářích.

Kromě toho je pro vývoj a testování kódu využíváno vývojářské prostředí PyCharm Community Edition 2023.1.3 spolu s programovacím jazykem Python, což umožňuje rychlý vývoj a experimentování s různými variantami EA.

6.2 Python a Java

Python je známý svou jednoduchou a čitelnou syntaxí, což umožňuje rychlejší a efektivnější vývoj kódu. Tato vlastnost je vhodná pro implementaci složitých algoritmů, jako jsou EA, které mohou mít komplexní struktury. Díky snadné čitelnosti kódu je také jednodušší sledovat a upravovat implementaci. Java má striktnější syntaxi a může vyžadovat více kódu pro dosažení stejného cíle.

Dalším důležitým faktorem je široká podpora. Existuje mnoho knihoven, nástrojů a online zdrojů, které usnadňují vývoj a analýzu algoritmů v Pythonu. Java má silnou a rozsáhlou komunitu, což znamená dostupnost knihoven a nástrojů pro vývoj algoritmů.

Rychlost prototypování je dalším klíčovým aspektem Pythonu. Jazyk je interaktivní a podporuje dynamické typování, což umožňuje rychlé vytváření

a testování částí algoritmu. Java může vyžadovat více práce pro vytvoření prototypů algoritmů z důvodu její složitosti.

Python nabízí bohaté knihovny pro vědecké výpočty, jako jsou NumPy a Matplotlib. Tyto knihovny jsou zásadní při implementaci EA, které často pracují s komplexními datovými strukturami a vyžadují vizualizaci výsledků.

Flexibilita Pythonu je dalším klíčovým faktorem. Python je univerzálním jazykem, který lze využít v různých oblastech, včetně vývoje her a UI. Má rozsáhlé nástroje pro vizualizaci dat, které nabízí i Java, ale Python má v tomto ohledu větší rozmanitost a možnosti.

Knihovny použité pro tvorbu simulací

Pro tvorbu algoritmu bylo využito několik knihoven pro implementaci a analýzu EA v herních situacích. Následující knihovny hrály klíčovou roli ve vývoji a experimentech:

- NumPy je základní knihovnou pro vědecké výpočty v Pythonu. Poskytuje efektivní nástroje pro práci s daty, lineární algebrou a numerickými výpočty. Umožňuje rychlý a optimalizovaný způsob manipulace s maticemi a vektory.
- Pygame je knihovna pro vytváření her a multimediálních aplikací v Pythonu. Umožňuje vytvářet interaktivní grafiku, zvukové efekty a animace. V rámci práce je využitá pro tvorbu vizuálních reprezentací herních scénářů a simulací, což umožňuje zobrazit a sledovat průběh experimentů.
- PyQt5 je knihovna pro tvorbu grafického uživatelského rozhraní v Pythonu. Pomocí této knihovny je vytvořené okno, na kterém lze sledovat statistiky v každé generaci. Důvodem použití této knihovny je, že Pygame neumí zobrazit více než jedno okno.
- Matplotlib je knihovna pro vizualizaci dat v Pythonu. Poskytuje nástroje pro tvorbu různých druhů grafů, diagramů a vizualizací. V algoritmu je využita pro zobrazení teplotních map.

- Tabulate je knihovna pro formátování a prezentaci tabulkových dat. Tato knihovna umožnila vytvoření tabulek pro každý experiment a usnadnila analýzu a porovnání různých aspektů.

Tato sada knihoven poskytuje základ pro implementaci, vizualizaci a analýzu EA v rámci herních simulací.

6.3 Analýza implementovaných částí kódu

Pohyb hráčů

```
  def move(self):
        if not self.is_alive or self.is_finished: # Kontrola,
zda je hráč stále živý nebo již nedosáhl cíle
            return
        angle = self.genes[
            self.step]  # Z genetického kódu hráče se na
základě aktuálního kroku získá úhel, pod kterým se bude hráč
pohybovat
        new_x = self.x + self.velocity * math.cos(angle)
        new_y = self.y + self.velocity * math.sin(angle)
        # Zamezení hráče pro vystoupení z okna
        if 0 <= new_x <= window_width - 25:
            self.x = new_x
        if 0 <= new_y <= window_height - 25:
            self.y = new_y
        self.positions_history.append((self.x, self.y)) #
Přidání pozice do seznamu pro vykreslování heatmapy
        self.step += 1
        if self.step >= len(
                self.genes):  # Pokud hráč provedl všechny
kroky daného genetického kódu, prochází svůj genetický kód od
začátku
            self.step = 0
```

Ukázka kódu 2 Pohyb hráčů
Zdroj: [autor]

V rámci implementace EA ve hrách má zásadní význam metoda nazvaná `move()`. Tato metoda je zodpovědná za řízený pohyb hráčů na herní ploše na základě jeho genetického kódu, který nese. Cílem této metody je umožnit hráčům provádět pohyby podle geneticky určených instrukcí a zároveň sledovat jeho polohu a historii pohybu.

V průběhu pohybu hráčů tato metoda následuje několik klíčových kroků:

- Kontrola životnosti a dosažení cíle: Nejprve metoda ověřuje, zda je hráč stále naživu a zda již nesplnil svůj cíl. V případě, že hráč není živý nebo již dokončil úkol, pohyb se neprovádí a metoda se ukončuje.
- Získání úhlu z genetického kódu: Z genetického kódu hráče `genes` je na základě aktuálního kroku `step` získán úhel, pod kterým by se hráč měl pohybovat.
- Výpočet nové pozice: Pomocí trigonometrických funkcí `math.cos` a `math.sin` je vypočtena nová pozice hráče na herní ploše na základě jeho aktuální pozice a rychlosti. To umožňuje simulovat pohyb hráče.
- Kontrola, zda je hráč uvnitř herní plochy: Je kontrolováno, zda se nová pozice hráče nachází uvnitř herní plochy. Tím se zabrání hráči opustit herní prostor.
- Zaznamenání pohybu: Aktuální pozice hráčů je přidána do historie jeho pohybu, což umožňuje pozdější vizualizaci a analýzu chování hráče.
- Inkrementace kroku: Po každém provedeném pohybu se počítadlo kroků `step` inkrementuje, aby bylo možné postupně procházet genetický kód hráče a provádět odpovídající pohyby.
- Cyklický pohyb genetického kódu: Jakmile hráč provede všechny kroky definované v genetickém kódu, počítadlo kroků se resetuje na počátek, čímž se umožní cyklický pohyb hráče podle stejného genetického kódu.

Dosažení cíle

```
# Výpočet vzdálenosti hráče od cíle
    def calculate_distance_to_target(self):
        target_x, target_y = target_position
        distance = math.sqrt((self.x - target_x) ** 2 +
(self.y - target_y) ** 2)
        return distance

    # Ověření zda se hráč nachází v cíli
    def is_colliding_with_target(self, target_position,
target_size):
        target_x, target_y = target_position
        distance = math.sqrt((self.x - target_x) ** 2 +
(self.y - target_y) ** 2)
        return distance < (self.size + target_size) / 2

    # Ověření, zda hráč dosáhl cíle a volání konce hry
    def check_target_reached(self, target_position,
target_size):
        if self.is_colliding_with_target(target_position,
target_size):
            output_file = "heatmap.png"
            create_heatmap_and_save_to_png(
                population.players, window_width,
window_height, output_file,
                target_position=target_position,
obstacles=obstacles)
            self.game_over()
        else:
            return False
```

Ukázka kódu 3 Dosažení cíle
Zdroj: [autor]

Tato část kódu `calculate_distance_to_target()` obsahuje tři důležité metody, které se týkají hráčova dosažení cíle na herní ploše:

- `calculate_distance_to_target()`: Tato metoda slouží k výpočtu vzdálenosti mezi hráčem a cílem na herní ploše. Využívá Pythagorovy věty pro výpočet Eukleidovské vzdálenosti mezi dvěma body. Souřadnice cíle jsou získány z proměnné `target_position` a výsledná vzdálenost je poté vrácena.
- `is_colliding_with_target(...)`: Tato metoda ověřuje, zda se hráč nachází v oblasti cíle. Opět využívá výpočtu Eukleidovské vzdálenosti mezi hráčem a cílem. Porovnává tuto vzdálenost s polovinou součtu velikostí hráče a cíle. Pokud je vzdálenost menší než tato hodnota, znamená to, že hráč se nachází v oblasti cíle a dochází k překryvu.
- `check_target_reached(...)`: Tato metoda ověřuje, zda hráč dosáhl cíle. Nejprve zkontroluje, zda hráč koliduje s cílem pomocí metody `is_colliding_with_target`. Pokud ano, vytvoří teplotní mapu a uloží ji do souboru pro následnou vizualizaci. Poté je volána metoda `game_over()`, která signalizuje, že hráč dosáhl cíle a hra byla ukončena.

Tyto metody tvoří důležitou součást algoritmu a zajišťují, že hráč bude správně sledován vzhledem k cíli na herní ploše a že hra bude ukončena v případě, že hráč dosáhne svého cíle.

Výpočet fitness

```
    def calculate_fitness(self):
        if not self.is_alive:  # Kontrola zda je hráč naživu
            self.fitness = 0
            return
        target_x, target_y = target_position
        distance = math.sqrt((self.x - target_x) ** 2 +
(self.y - target_y) ** 2)
```

```
# Normalizace vzdálenost pro hodnoty 0 až 100
        if self.min_distance is not None and
self.max_distance is not None:
            normalized_distance = (distance -
self.min_distance) / (self.max_distance - self.min_distance)
            self.fitness = 100 - (normalized_distance * 100)
        else:
            self.fitness = 1 / distance
```

Ukázka kódu 4 Výpočet fitness
Zdroj: [autor]

Tato metoda `calculate fitness()` slouží k výpočtu hodnoty fitness pro konkrétního hráče. Funkce fitness je v tomto případě úspěšnost hráče, jak daleko se nachází od cíle. Výpočet hodnoty fitness zahrnuje několik kroků:

- Nejprve se kontroluje, zda je hráč naživu proměnnou `is_alive`. Pokud hráč není naživu, jeho fitness se nastaví na 0 a výpočet se ukončí.
- Vypočítá se Eukleidovská vzdálenost mezi současnou pozicí hráče a cílem souřadnice `target_position`.
- Vzdálenost se normalizuje v rozmezí 0 až 100, což umožňuje porovnání různých fitness hodnot. Normalizace se provádí na základě minimální a maximální vzdálenosti `min_distance` a `max_distance`, které jsou uloženy pro každého hráče. Hodnota vzdálenosti je normalizována na číslo mezi 0 a 1, a poté převedena na rozmezí 0 až 100. Čím je hráč blíže cíli, tím má vyšší fitness.
- Nakonec se vypočítá samotná hodnota fitness. Pokud jsou k dispozici `min_distance` a `max_distance`, je fitness nastavena na hodnotu 100 mínus normalizovaná vzdálenost (čím blíže cíli, tím vyšší fitness). Pokud `min_distance` a `max_distance` nejsou k dispozici (což může nastat v některých situacích), je fitness nastavena na inverzní hodnotu vzdálenosti.

Ruletový výběr

```
    def roulette_selection(self):
        total_fitness = sum(player.fitness for player in
self.players) # Celková fitness populace
        pick = random.uniform(0, total_fitness) # vybrání
náhodného čísla od 0 do celkové fitness
        current = 0
        for player in self.players:
            current += player.fitness
            if current > pick and player.is_alive:  # Výběr
pouze z živých hráčů
                return player
```

Ukázka kódu 5 Ruletový výběr
Zdroj: [autor]

Tato metoda `roulette_selection()` provádí selekci jedince z populace pomocí ruletového výběru v rámci EA. Metoda provádí několik kroků:

- `total_fitness`: Nejprve se spočítá celková fitness populace. To se provádí tak, že se projde každý jedinec v populaci a jeho fitness se přičítá k celkové sumě.
- `pick`: Následně se náhodně vybere číslo v rozsahu od 0 do celkové fitness populace. Toto číslo určuje pozici na ruletě, kde hledá jedince.
- `current`: Tato proměnná slouží jako ukazatel pozice na ruletě. Postupně se přesouvá po ruletě a hledá místo, kde se nachází vybrané číslo.
- Cyklus pro výběr jedince prochází jednotlivé jedince v populaci. V každém kroku přidá k proměnné `current` fitness aktuálního jedince. Pokud se `current` dostane nad hodnotu `pick`, znamená to, že se našlo místo na ruletě, kde je náhodně vybraný bod. Dále kontroluje, zda je daný hráč naživu, pokud ano, tak ho vrátíme jako vybraného jedince.

Turnajový výběr

```
  def tournament_selection(self, tournament_size):
        alive_players = [player for player in self.players if
player.is_alive]  # Vytvoření seznamu živých hráčů
        tournament_players = random.sample(alive_players,
tournament_size)  # Náhodný výběr hráčů do turnaje
        tournament_players.sort(key=lambda player:
player.fitness, reverse=True)  # Seřazení hráčů podle fitness
        return tournament_players[0]  # Vrátíme hráče
s nejvyšší fitness
```

Ukázka kódu 6 Turnajový výběr
Zdroj: [autor]

Tato metoda `tournament_selection(…)` provádí selekci jedince z populace pomocí turnajového výběru. Následující kroky vysvětlují, jak tato metoda funguje:

- `alive_players`: Nejprve se vytvoří seznam živých hráčů. To se provádí filtrováním populace tak, aby zůstali pouze ti hráči, kteří jsou stále naživu.
- `tournament_players`: Následně se provede náhodný výběr hráčů do turnaje. Zde se používá funkce `random.sample`, která vybere `tournament_size` náhodných hráčů ze seznamu živých hráčů. Tím je vytvořena turnajová skupina hráčů.
- `tournament_players.sort(...)`: Hráči v turnaji jsou seřazeni podle jejich fitness od nejvyšší po nejnižší. Tím se zajišťuje, že první hráč v seznamu má nejvyšší fitness.
- `return tournament_players[0]`: Na závěr funkce je vrácen hráč s nejvyšší fitness, tedy vítěz turnaje. Tímto způsobem je získán jedinec pro další evoluční operace křížení a mutace.

Jednobodové křížení

```
    def single_point_crossover(parent1, parent2):
        crossover_point = random.randint(0,
len(parent1.genes))
        genes = parent1.genes[:crossover_point] +
parent2.genes[crossover_point:]
        new_child = Player(genes=genes)
        return new_child
```

Ukázka kódu 7 Jednobodové křížení
Zdroj: [autor]

Tato `metoda single_point_crossover(…)` provádí operaci jednobodového křížení mezi dvěma rodiči, kteří jsou reprezentováni genetickými kódy. Následující etapy popisují fungování této metody:

- `crossover_point`: Nejprve se náhodně vybere bod, ve kterém dojde ke křížení genetických kódů rodičů. Tento bod je vybrán jako náhodné celé číslo od 0 do délky genetického kódu prvního rodiče (parametr `len(parent1.genes)`).
- `genes`: Geny potomka se vytvoří spojením první části genetického kódu prvního rodiče a druhé části genetického kódu druhého rodiče. Tímto se vytvoří nová sada genů pro potomka.
- `new_child`: Nový potomek se vytvoří pomocí spojených genů. To zahrnuje vytvoření instance třídy `Player` a předání nových genů jako parametru.
- `return new_child`: Na závěr metoda vrací nově vytvořeného potomka, který vznikl křížením genetických kódů rodičů.

Dvoubodové křížení

```
      def two_point_crossover(parent1, parent2):
        # Zajištění, že první rodič má menší délku
genetického kódu, pokud tomu tak není, rodiče jsou prohozeni
        if len(parent1.genes) > len(parent2.genes):
            parent1, parent2 = parent2, parent1
        size = min(len(parent1.genes), len(parent2.genes))  #
Určení délky kódu pro křížení
        # Dva náhodné body křížení
        crossover_point1 = random.randint(0, size)
        crossover_point2 = random.randint(0, size)
        # Body křížení seřazené vzestupně
        crossover_start = min(crossover_point1,
crossover_point2)
        crossover_end = max(crossover_point1,
crossover_point2)
        child = Player(genes=[])
        # Geny před prvním bodem křížení
        child.genes[:crossover_start] =
parent1.genes[:crossover_start]
        # Geny mezi 2 body křížení
        child.genes[crossover_start:crossover_end] =
parent2.genes[crossover_start:crossover_end]
        # Geny po 2. bodu křížení
        child.genes[crossover_end:] =
parent1.genes[crossover_end:]
        return child
```

Ukázka kódu 8 Dvoubodové křížení
Zdroj: [autor]

Tato metoda `two_point_crossover(...)` implementuje dvoubodové křížení mezi dvěma rodiči, kteří mají genetické kódy reprezentované sekvencí genů. Následující kroky popisují fungování této metody:

- Nejprve se zajišťuje, že první rodič má menší nebo stejnou délku genetického kódu než druhý rodič. Pokud tomu tak není, rodiče jsou prohozeni. To je důležité pro správné provedení dvoubodového křížení.
- Je určena délka kódu, který bude sloužit pro křížení. Tato délka je menší než délka kódů obou rodičů.
- Náhodně se vybírají dva body křížení, `crossover_point1` a `crossover_point2`, které označují indexy v genetickém kódu. Tyto body určují místo, kde budou geny rodičů vyměněny.
- Body křížení jsou seřazeny vzestupně, aby bylo zajištěno správné vymezení oblasti pro křížení.
- Vytvoří se nový potomek, který bude mít prázdný genetický kód.
- Geny před prvním bodem křížení jsou převzaty z prvního rodiče.
- Geny mezi dvěma body křížení jsou převzaty z druhého rodiče.
- Geny po druhém bodu křížení jsou opět převzaty z prvního rodiče.
- Nový potomek je vrácen jako výsledek křížení.

Uniformní křížení

```
    def uniform_crossover(parent1, parent2):
        child_genes = []
        for gene1, gene2 in zip(parent1.genes,
parent2.genes):
            # Pro každý gen vybíráme náhodně z rodičů
            child_gene = random.choice([gene1, gene2])
            child_genes.append(child_gene)
        child = Player(child_genes)
        return child
```

Ukázka kódu 9 Uniformní křížení
Zdroj: [autor]

Metoda `uniform_crossover(...)` provádí operaci uniformního křížení mezi dvěma rodiči, kteří jsou reprezentováni genetickými kódy. Při uniformním křížení je každý gen potomka volen z jednoho z rodičů s určitou pravděpodobností. Konkrétně metoda pracuje tímto způsobem:

- Metoda prochází postupně geny obou rodičů paralelně pomocí funkce `zip`.
- Pro každý gen je náhodně vybrán jeden z rodičů. To se provádí pomocí funkce `random.choice`, která náhodně vybere jednu z možností.
- Vybrané geny jsou postupně přidávány do seznamu `child_genes`, který reprezentuje genetický kód nového potomka.
- Jakmile jsou všechny geny potomka vybrány, je vytvořen nový hráč s tímto genetickým kódem, a tím je provedeno uniformní křížení.

Mutace

```
  def mutation(self, player):
        if self.average_fitness >
self.previous_average_fitness:  # Kontrola pokud je průměrné
fitness větší než předchozí fitness
            self.mutation_rate = self.low_mutation_rate
        else:
            self.mutation_rate = self.high_mutation_rate
        for i in range(len(player.genes)):
            if random.random() < self.mutation_rate:
                player.genes[i] += random.uniform(-1.5, 1.5)
                player.genes[i] %= 2 * math.pi
```

Ukázka kódu 10 Mutace
Zdroj: [autor]

Tato metoda `mutation(...)` implementuje adaptivní mutaci genetického kódu hráče. Mutace je důležitou součástí evolučního procesu, protože umožňuje diverzifikaci genetického materiálu a hledání nových řešení. Metoda pracuje tímto způsobem:

- Nejprve se provádí kontrola, zda je průměrná fitness populace (reprezentovaná proměnnou `self.average_fitness` větší než předchozí průměrná fitness `self.previous_average_fitness`. Tato kontrola slouží k určení, zda má dojít k úpravě mutační rychlosti. Pokud je průměrná fitness vyšší, mutační rychlost je snížena na nižší hodnotu

`self.low_mutation_rate`, což zvyšuje pravděpodobnost menších změn v genetickém kódu. Pokud je průměrná fitness nižší nebo se nezměnila, mutační rychlost se nastaví na vyšší hodnotu `self.high_mutation_rate`.

- Procházejí se všechny geny genetického kódu hráče `player.genes`. Pro každý gen se provádí následující kroky:
    - Generuje se náhodné číslo mezi 0 a 1 pomocí `random.random()`. Pokud je toto náhodné číslo menší než aktuální mutační rychlost, dojde k mutaci tohoto genu.
    - K hodnotě genu se přičte náhodná hodnota z intervalu (-1.5, 1.5). Tím dochází ke změně genu o náhodnou malou hodnotu.
    - Pro zajištění, že hodnota genu zůstává v povoleném rozsahu (0 až 2π, což odpovídá kruhovému rozsahu úhlů), se použije operace % (modulo) s hodnotou `2 * math.pi`. To zajistí, že i po přidání náhodné hodnoty se hodnota genu pohybuje v daném rozsahu.

Nová generace

```
  def next_generation(self):
        self.update_fitness()
        if any(player.fitness is None for player in
self.players):
            return
        self.players.sort(key=lambda player: player.fitness,
reverse=True)
        new_players = []
        elite_count = int(
            self.elite_percentage * len(self.players))
```

```
# Elitismus zachování top 10% hráčů z předchozí generace
        new_players.extend(self.players[:elite_count])
        remaining_count = len(self.players) - elite_count
for _ in range(remaining_count):
            parent1 = self.selection()
            parent2 = self.selection()
            child = self.crossover(parent1, parent2)
            self.mutation(child)
            new_players.append(child)
        self.players = new_players
        self.remaining_players = len(self.players)
        # Přidání hodnot do tabulky pro aktuální generaci
        population.update_table(self.generation)
        population.display_table()
        self.generation += 1
        for player in self.players:
            player.reset()
            player.is_alive = True
            player.fitness = None
            player.min_distance = self.min_distance
            player.max_distance = self.max_distance
```

Ukázka kódu 11 Nová generace
Zdroj: [autor]

Tato metoda `next_generation()` slouží k vytváření nové generace hráčů. Následující kroky vysvětlují, jak tato metoda pracuje:

- Nejprve jsou aktualizovány hodnoty fitness pro všechny hráče aktuální generace.
- Kontrola existence fitness hodnot. Je ověřeno, zda jsou všem hráčům přiřazeny platné hodnoty fitness. Pokud ne, generace zůstává beze změny.
- Hráči jsou seřazeni sestupně podle jejich fitness hodnot, čímž jsou umístěni nejlepší hráči na začátku, z důvodu implementace elitismu.

- Část nejlepších hráčů (určená procentem) je zachována pro další generaci beze změn (elitismus). Tím je zajištěno, že nejlepší jedinci se přenášejí do následující generace.
- Zbývající hráči jsou vytvořeni pomocí operací křížení a mutace. Párování rodičů je prováděno pomocí selekce, kde jsou vybráni dva rodiče na základě jejich fitness hodnot. Poté je mezi nimi provedeno křížení a aplikována mutace na výsledného potomka.
- Seznam hráčů je nahrazen novými hráči vytvořenými v této generaci. Rovněž jsou resetovány atributy hráčů, aby byli připraveni pro další kolo evoluce.
- Data o aktuální generaci jsou přidána do tabulky pro sledování průběhu evoluce a generace jsou posunuty o jedna.

6.4 Experimenty evolučních algoritmů

V rámci experimentů a analýzy využití EA v herním prostředí byly provedeny testy s cílem zkoumat vliv různých selekčních metod a metod křížení. Experimenty byly prováděny na simulované populaci hráčů o velikosti 1000 jedinců, kde každý hráč byl reprezentován genotypem představujícím jeho pohyb. V každém z těchto testů byl implementován elitismus a adaptivní mutace.

Velikost 1000 hráčů pro každou generaci byla zvolena z důvodu dosažení dostatečného pokrytí různých strategií a možností v rámci evolučního procesu. Tisícová populace umožňuje algoritmům efektivně prozkoumat různé varianty chování a taktik a nalézt optimální řešení či strategii. Zároveň zvolená velikost populace umožňuje algoritmům zachovat rozmanitost a diverzitu, což je klíčové pro nalezení řešení.

Experimenty jsou následující kombinace selekčních metod a metod křížení:

- Turnajový výběr a jednobodové křížení
- Turnajový výběr a dvoubodové křížení
- Turnajový výběr a uniformní křížení
- Ruletový výběr a jednobodové křížení

- Ruletový výběr a dvoubodové křížení
- Ruletový výběr a uniformní křížení

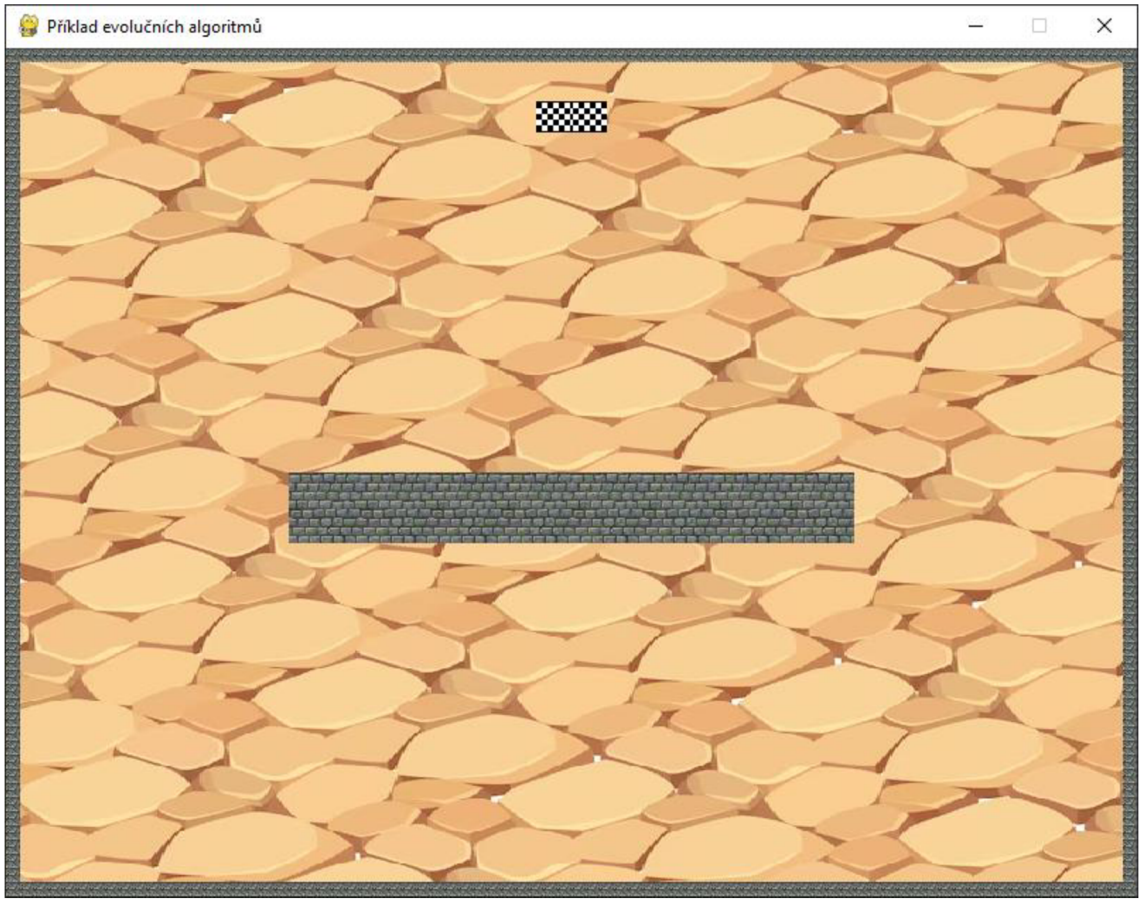
Každý test v experimentu byl opakován desetkrát, aby bylo možné sledovat konzistenci a variabilitu výsledků. Během každého testu byla populace hráčů v každé generaci ohodnocena pomocí fitness funkce, která vypočítávala úspěšnost daného hráče v herním prostředí, v těchto případech vzdálenost od cíle. Následně byla provedena selekce rodičů a aplikace metod křížení na vytvoření potomků. Potomci byli podrobeni mutacím a výsledná populace byla připravena pro následující generaci.

V průběhu experimentů bylo důležité mít jasné kritérium pro ukončení každé generace. Každý experiment má dvě ukončovací podmínky.

- Všichni hráči zemřou. Tato podmínka ukončuje generaci, když všichni hráči v generaci zemřou. To může nastat, pokud všichni hráči narazí do překážek. Generace se ukončuje, protože žádný hráč nemá možnost provádět další akce.
- Časový limit 30 sekund. Tato podmínka ukončuje generaci po uplynutí časového limitu. Bez ohledu na to, zda hráči stále žijí nebo ne, pokud uplyne tato časová lhůta, generace je ukončena. Tím se zajišťuje, že celý experiment zůstává v přijatelném časovém rámci a je možné provádět evoluční metody.

6.4.1 Experimenty jednoduché překážky

Pro účely experimentů jednoduché překážky byla do herního prostředí přidána specifická překážka. Hlavním cílem této modifikace je umožnit hráčům naučit se efektivně obcházet tuto překážku prostřednictvím využití evolučních metod. Tímto způsobem se zkoumá, jakým způsobem se algoritmy adaptují na nové podmínky a vyvinou strategie pro úspěšné překonání překážky a dosažení stanoveného cíle.



Obrázek 11 Jednoduchá překážka
Zdroj: [autor]

6.4.1.1 Turnajový výběr a jednobodové křížení pro jednoduchou překážku

V rámci experimentu byly provedeny testy s využitím kombinace turnajového výběru a jednobodového křížení, které měly za cíl analyzovat, jak tyto dvě evoluční metody spolupracují a jakým způsobem ovlivňují vývoj strategií UI v herním prostředí. Tato kombinace byla zvolena s ohledem na možnost dosažení synergie mezi výběrem nejsilnějších hráčů turnajem a následným genetickým křížením, konkrétně formou jednobodového křížení.

Cílem tohoto experimentu bylo zjistit, jak se kombinace těchto dvou evolučních metod projeví v kontextu hledání optimální strategie v herním prostředí. Analýza výsledných dat z tohoto experimentu poskytuje informace o tom, zda turnajový výběr spolu s jednobodovým křížením přispívá k rychlejší konvergenci algoritmu k efektivní strategii a jakým způsobem ovlivňuje diverzitu v populaci.

Po provedení tohoto experimentu následují tabulky a teplotní mapa, které poskytnou detailní pohled na vývoj a chování UI v rámci této kombinace evolučních metod.

Průměrný počet generací	3,9
Střední hodnota fitness	43,52
Průměr nejvyšší fitness	88,91
Průměr zbývajících hráčů	689,79
Průměr kroků do cíle	1558

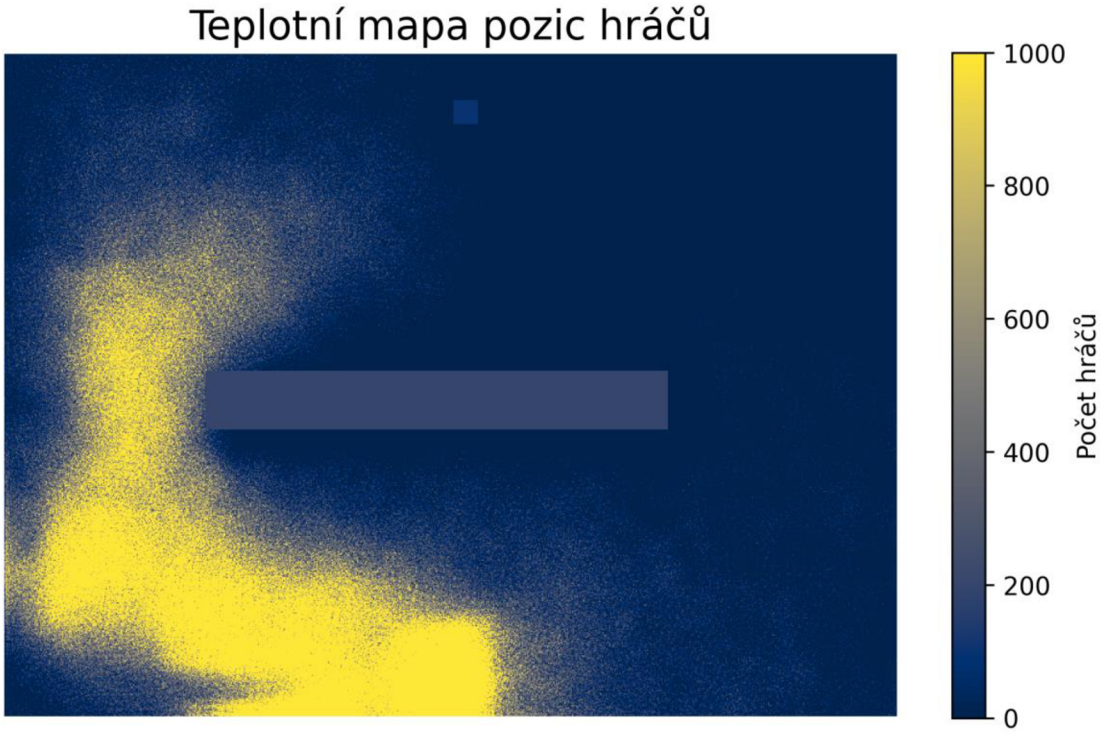
Tabulka 2 Průměry turnajového výběru a jednobodového křížení pro jednoduchou překážku
Zdroj: [autor]

Nejvyšší počet živých hráčů	829
Nejnižší počet živých hráčů	481
Nejvyšší počet generací	6
Nejnižší počet generací	1
Nejvyšší počet kroků	1686
Nejnižší počet kroků	1415

Tabulka 3 Extrémy turnajového výběru a jednobodového křížení pro jednoduchou překážku
Zdroj: [autor]

Generace	Průměrné fitness	Nejlepší fitness	Zbývající hráči
1	25,83	75,8	491
2	45,85	85,23	784
3	46,79	91,39	707
4	55,27	93,93	736
5	57,31	100	773

Tabulka 4 Příklad 1. testu turnajového výběru a jednobodového křížení pro jednoduchou překážku
Zdroj: [autor]



Obrázek 12 Teplotní mapa 1. testu turnajového výběru a jednobodového křížení pro jednoduchou překážku
Zdroj: [autor]

V rámci provedeného experimentu s využitím turnajového výběru a jednobodového křížení byla provedena analýza vývoje strategií UI v herním prostředí. Výsledky tohoto experimentu prezentují průměrné hodnoty a charakteristiky generací během evoluce.

Průměrný počet generací byl 3,9. Střední hodnota fitness dosáhla hodnoty 43,52, zatímco průměrná nejvyšší fitness byla 88,91.

V každém testu zůstalo průměrně 689,79 hráčů. Průměrný počet kroků do cíle činil 1558.

Během experimentu byly zaznamenány také extrémní hodnoty. Nejvyšší počet živých hráčů dosáhl hodnoty 829, zatímco nejnižší počet živých hráčů byl 481. Nejvyšší počet generací, který byl dosažen, byl 6, zatímco nejnižší byl 1. Nejvyšší počet kroků dosáhl hodnoty 1686, zatímco nejnižší počet kroků byl 1415.

Konečně, analýza průběhu 1. testu ukazuje vývoj generací od 1. do 5. postupným růstem průměrné fitness od 25,83 až po 57,31. Nejlepší dosažená fitness se pohybovala od 75,8 do 100.

6.4.1.2 Turnajový výběr a dvoubodové křížení pro jednoduchou překážku

V rámci zkoumaného experimentu byl uskutečněn test, kde byl kombinován turnajový výběr a dvoubodové křížení. Tento komplexní test měl za úkol analyzovat vzájemnou interakci a vliv těchto dvou evolučních metod na vývoj strategií UI v herním prostředí. Tato kombinace byla zvolena s úmyslem nalézt synergii mezi výběrem nejsilnějších hráčů pomocí turnajového výběru a následným genetickým křížením, konkrétně formou dvoubodového křížení.

Cílem tohoto experimentu bylo odhalit, jak se tato kombinace evolučních metod projeví v rámci hledání optimální strategie v herním prostředí. Díky analýze získaných dat z tohoto experimentu byla možnost zhodnotit, zda použití turnajového výběru společně s jednobodovým křížením přispívá k rychlejší konvergenci EA k efektivní strategii a jak ovlivňuje diverzitu v populaci hráčů.

Po dokončení samotného experimentu následují prezentace výsledků pomocí tabulek a teplotní mapy.

Průměrný počet generací	3,9
Střední hodnota fitness	40,25
Průměr nejvyšší fitness	88,93
Průměr zbývajících hráčů	674,59
Průměr kroků do cíle	1453,9

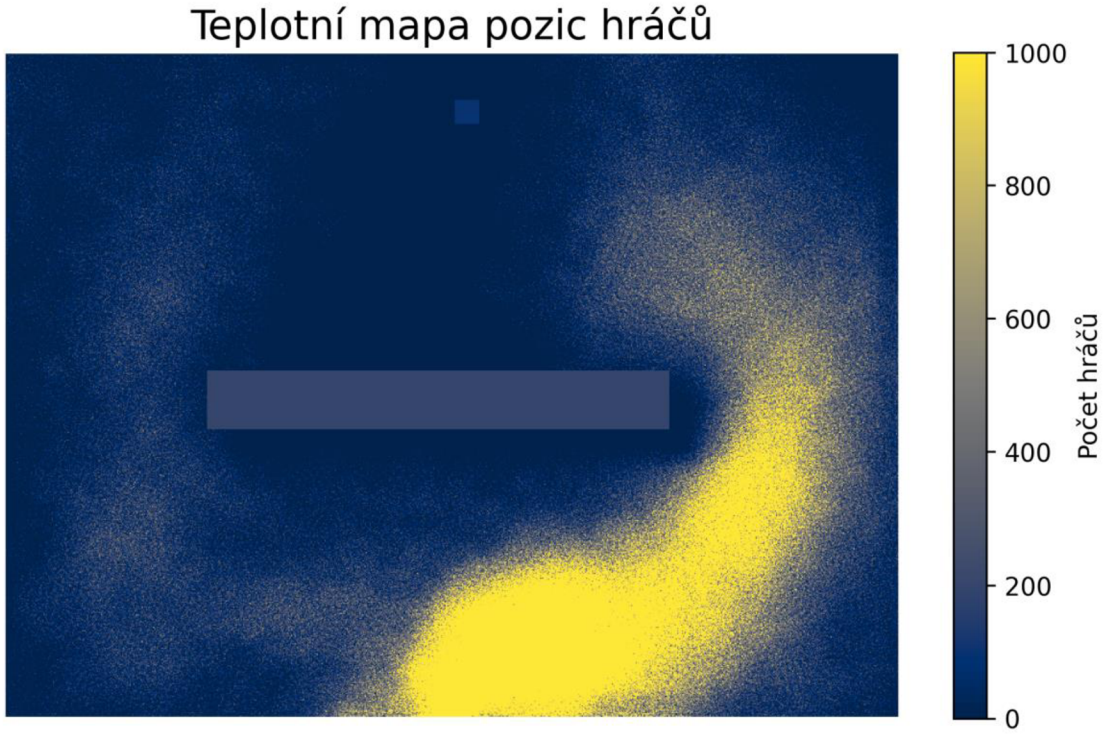
Tabulka 5 Průměry turnajového výběru a dvoubodového křížení pro jednoduchou překážku
Zdroj: [autor]

Nejvyšší počet živých hráčů	925
Nejnižší počet živých hráčů	478
Nejvyšší počet generací	6
Nejnižší počet generací	3
Nejvyšší počet kroků	1706
Nejnižší počet kroků	1020

Tabulka 6 Extrémy turnajového výběru a dvoubodového křížení pro jednoduchou překážku
Zdroj: [autor]

Generace	Průměrné fitness	Nejlepší fitness	Zbývající hráči
1	25,86	76,27	493
2	45,70	80,05	791
3	45,78	84,27	740
4	42,45	88,67	642
5	43,88	92,67	616
6	47,63	100	616

Tabulka 7 Příklad 9. testu turnajového výběru a dvoubodového křížení pro jednoduchou překážku
Zdroj: [autor]



Obrázek 13 Teplotní mapa 9. testu turnajového výběru a dvoubodového křížení pro jednoduchou překážku
Zdroj: [autor]

V rámci provedeného experimentu s využitím kombinace turnajového výběru a dvoubodového křížení byla analyzována evoluce strategií UI v herním prostředí. Výsledky této zkoušky prezentují průměrné charakteristiky jednotlivých generací a odhalují, jakým způsobem se algoritmus vyvíjel směrem k dosažení optimálních strategií.

Průměrný počet generací v rámci experimentu dosáhl hodnoty 3,9. Střední hodnota fitness dosáhla úrovně 40,25, přičemž průměrná nejvyšší dosažená fitness byla 88,93.

V průběhu experimentu byly zaznamenány také extrémní hodnoty. Nejvyšší počet živých hráčů dosáhl 925, naopak nejnižší počet živých hráčů byl 478. Maximální počet dosažených generací byl 6, zatímco nejnižší počet byl 3. Nejvyšší počet kroků dosáhl hodnoty 1706, zatímco nejnižší počet kroků byl 1020.

Analýza 9. testu vývoje generací od 1. do 6. ukázala postupný nárůst průměrné fitness od 25,86 do 47,63. Nejlepší dosažená fitness se pohybovala od 76,27 do 100.

6.4.1.3 Turnajový výběr a uniformní křížení pro jednoduchou překážku

V rámci prováděného experimentu byl propojen turnajový výběr a uniformní křížení. Tento komplexní test měl za úkol analyzovat vzájemnou interakci a vliv těchto dvou evolučních metod na vývoj strategií UI v herním prostředí. Tato kombinace byla zvolena s cílem prozkoumat možné synergické efekty mezi výběrem nejsilnějších hráčů prostřednictvím turnajového výběru a následným genetickým křížením, a to konkrétně pomocí uniformního křížení.

Účelem této experimentální analýzy bylo odhalit, jak se tato konkrétní kombinace evolučních metod projeví v kontextu hledání optimální strategie v herním prostředí. Skrze analýzu získaných dat z tohoto pokusu bylo možné zkoumat, zda použití turnajového výběru společně s uniformním křížením přináší rychlejší konvergenci EA k efektivní strategii a jakým způsobem ovlivňuje diverzitu v populaci hráčů.

Po dokončení samotného experimentu následuje prezentace výsledků formou tabulek a teplotní mapy.

Průměrný počet generací	6,8
Střední hodnota fitness	34,37
Průměr nejvyšší fitness	84,89
Průměr zbývajících hráčů	577,4
Průměr kroků do cíle	1502,4

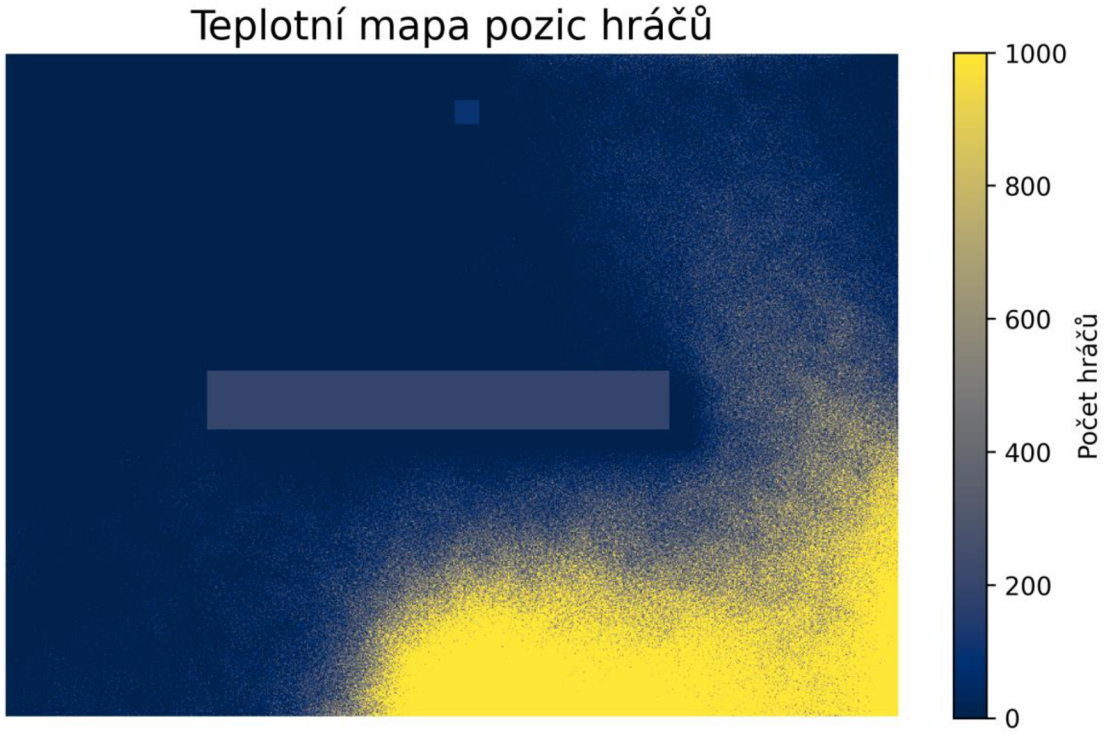
Tabulka 8 Průměry turnajového výběru a uniformního křížení pro jednoduchou překážku
Zdroj: [autor]

Nejvyšší počet živých hráčů	760
Nejnižší počet živých hráčů	399
Nejvyšší počet generací	14
Nejnižší počet generací	4
Nejvyšší počet kroků	1585
Nejnižší počet kroků	1421

Tabulka 9 Extrémy turnajového výběru a uniformního křížení pro jednoduchou překážku
Zdroj: [autor]

Generace	Průměrné fitness	Nejlepší fitness	Zbývající hráči
1	25,25	78,86	480
2	35,2	75,15	644
3	35,43	69,82	614
4	34,62	82,14	585
5	38,28	78,13	650
6	33,69	74,86	562
7	37,48	73,12	627
8	33,05	75,51	542
9	34,34	76,77	561
10	34,96	82,92	566
11	32,75	87,45	518
12	35,47	90,49	501
13	45,59	95,41	592
14	45,89	100	618

Tabulka 10 Příklad 7. testu turnajového výběru a uniformního křížení pro jednoduchou překážku
Zdroj: [autor]



Obrázek 14 Teplotní mapa 7. testu turnajového výběru a uniformního křížení pro jednoduchou překážku
Zdroj: [autor]

V rámci provedeného experimentu, kde byla zkombinována metoda turnajového výběru s uniformním křížením, byla provedena analýza evoluce strategií UI v herním prostředí. Výsledky této studie prezentují průměrné charakteristiky jednotlivých generací a odhalují, jakým způsobem se algoritmus vyvíjel směrem k dosažení optimálních strategií.

Průměrný počet generací v rámci tohoto experimentu činil 6,8. Střední hodnota fitness dosáhla úrovně 34,37, zatímco průměrná nejvyšší dosažená fitness byla 84,89.

V každém testu zůstalo průměrně 577,4 hráčů. Průměrný počet kroků do cíle činil 1502,4.

Během experimentu byly zaznamenány extrémní hodnoty. Nejvyšší počet živých hráčů dosáhl hodnoty 760, naopak nejnižší počet živých hráčů byl 399. Maximální počet dosažených generací byl 14, zatímco minimální počet pouze 4. Maximální počet kroků dosáhl hodnoty 1585, zatímco nejnižší počet kroků byl 1421.

Analýza vývoje 7. testu od 1. do 14. generace ukázala postupný nárůst průměrné fitness od 25,25 do 45,89. Nejlepší dosažená fitness se pohybovala od 69,82 do 100.

6.4.1.4 Ruletový výběr a jednobodové křížení pro jednoduchou překážku

V rámci prováděného experimentu byly provedeny testy, ve kterých byl kombinován ruletový výběr a jednobodové křížení. Tento experiment měl za úkol detailně analyzovat vzájemné působení a dopad těchto dvou evolučních metod na vývoj strategií UI v herním prostředí. Tato kombinace byla zvolena s cílem prozkoumat možnou synergii mezi selekcí nejsilnějších hráčů skrze ruletový výběr a následným genetickým křížením, konkrétně formou jednobodového křížení.

Hlavním cílem této experimentální analýzy bylo odhalit, jak tato specifická kombinace evolučních metod ovlivňuje hledání optimální strategie v herním prostředí. Skrze pečlivou analýzu získaných dat z tohoto testu bylo možné zkoumat, zda integrace ruletového výběru a jednobodového křížení poskytuje rychlejší konvergenci EA k efektivní strategii a jakým způsobem ovlivňuje diverzitu v populaci hráčů.

Po dokončení samotného experimentu následuje prezentace výsledků ve formě strukturovaných tabulek a teplotní mapy z konkrétního testu.

Průměrný počet generací	15,3
Střední hodnota fitness	48,67
Průměr nejvyšší fitness	84,07
Průměr zbývajících hráčů	851,3
Průměr kroků do cíle	1615,8

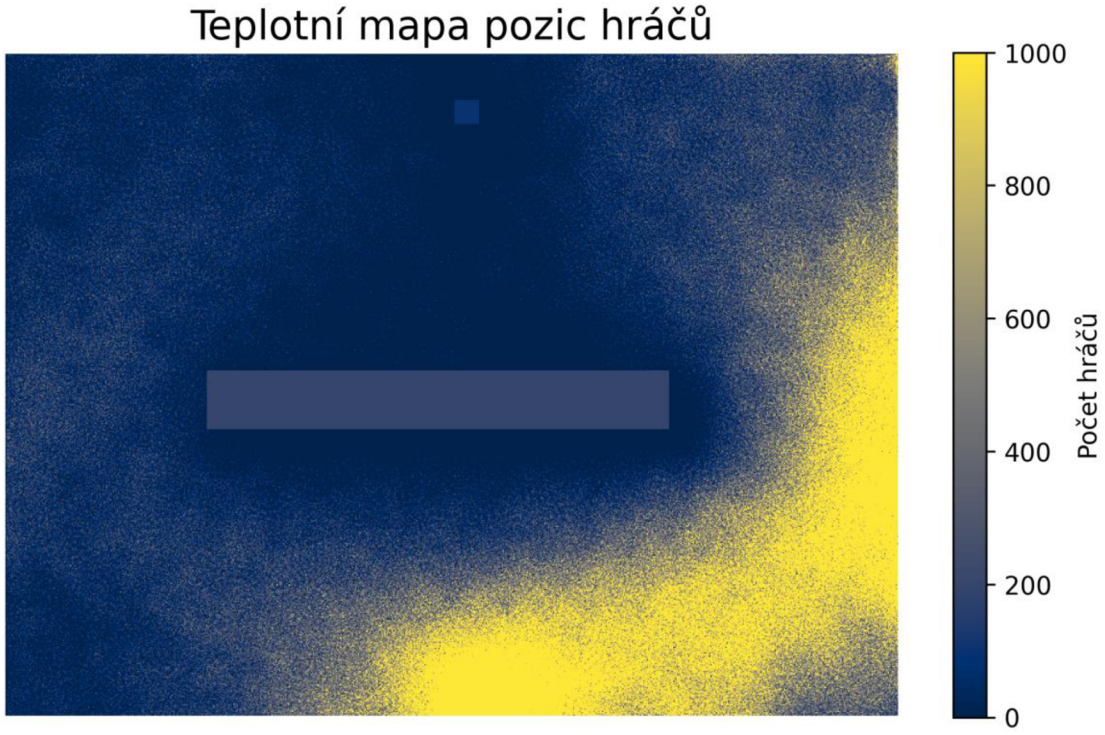
Tabulka 11 Průměry ruletového výběru a jednobodového křížení pro jednoduchou překážku
Zdroj: [autor]

Nejvyšší počet živých hráčů	940
Nejnižší počet živých hráčů	468
Nejvyšší počet generací	36
Nejnižší počet generací	1
Nejvyšší počet kroků	1736
Nejnižší počet kroků	1436

Tabulka 12 Extrémy ruletového výběru a jednobodového křížení pro jednoduchou překážku
Zdroj: [autor]

Generace	Průměrné fitness	Nejlepší fitness	Zbývající hráči
1	25,88	79,74	495
2	44,88	85,27	856
3	45,51	80,87	862
4	47,82	83,92	892
5	47,2	79,18	875
6	46,78	78,55	862
7	48,01	79,34	876
8	50,71	87,63	913
9	50,2	75,09	891
10	49,23	79,04	866
11	48,46	84,87	858
12	48,66	78,27	859
13	50,9	81,52	889
14	50,57	81,49	880
15	49,82	83,33	866
16	50,7	87,43	871
17	51,6	86,29	876
18	51,57	88,93	860
19	51,68	89,79	852
20	53,06	89,91	867
21	54,11	90,76	865
22	54,97	89,96	864
23	54,87	92,84	848
24	53,63	93,12	817
25	54,48	94,33	818
26	57,69	92,41	844
27	59,19	93,86	846
28	60,04	95,4	847
29	60,84	95,34	843
30	63,01	100	870

Tabulka 13 Příklad 9. testu ruletového výběru a jednobodového křížení pro jednoduchou překážku
Zdroj: [autor]



Obrázek 15 Teplotní mapa 9. testu ruletového výběru a jednobodového křížení pro jednoduchou překážku
Zdroj: [autor]

V rámci prováděného experimentu, kde byla kombinována metoda ruletového výběru s jednobodovým křížením, byla realizována analýza evoluce strategií UI v herním prostředí. Výsledky této studie představují průměrné charakteristiky jednotlivých generací a odhalují, jakým způsobem se algoritmus vyvíjel směrem k dosažení optimálních strategií.

Průměrný počet generací v rámci tohoto experimentu činil 15,3. Střední hodnota fitness dosáhla úrovně 48,67, zatímco průměrná nejvyšší dosažená fitness byla 84,07.

Počet zbývajících hráčů v každém testu dosáhl průměrné hodnoty 851,3. Průměrný počet kroků do cíle činil 1615,8.

V průběhu experimentu byly zaznamenány extrémní hodnoty. Nejvyšší počet živých hráčů dosáhl hodnoty 940, zatímco nejnižší počet živých hráčů byl 468. Maximální počet dosažených generací byl 36, zatímco minimální počet jen 1. Nejvyšší počet kroků dosáhl hodnoty 1736, zatímco nejnižší počet kroků byl 1436.

Analýza vývoje 9. testu od 1. do 30. generace ukázala postupný nárůst průměrné fitness od 25,88 do 63,0049. Nejlepší dosažená fitness se pohybovala od 75,09 do 100.

6.4.1.5 Ruletový výběr a dvoubodové křížení pro jednoduchou překážku

V rámci prováděného experimentu byla realizována série testů, kde byla kombinována metoda ruletového výběru s dvoubodovým křížením. Tento komplexní soubor testů měl za cíl důkladně analyzovat vzájemné interakce a dopad těchto dvou evolučních technik na vývoj strategií UI v herním prostředí. Tato kombinace byla vybrána s úmyslem zkoumat potenciální synergii mezi selekcí nejsilnějších hráčů pomocí ruletového výběru a následným genetickým křížením, konkrétně formou dvoubodového křížení.

Hlavním záměrem této experimentální analýzy bylo odhalit, jak tato specifická kombinace evolučních metod ovlivňuje proces hledání optimální strategie v herním prostředí. Skrze pečlivou analýzu dat získaných z těchto testů bylo možné prověřit, zda integrace ruletového výběru a dvoubodového křížení přispívá ke zrychlené konvergenci EA k efektivní strategii a jakým způsobem ovlivňuje diverzitu v populaci hráčů.

Po ukončení samotného experimentálního procesu následovala prezentace výsledků formou systematických tabulek a teplotní mapy konkrétního testování. Tyto grafické reprezentace umožnily detailní pohled na dynamiku a chování UI v rámci této konkrétní kombinace evolučních technik.

Průměrný počet generací	21,5
Střední hodnota fitness	48,33
Průměr nejvyšší fitness	83,43
Průměr zbývajících hráčů	855,69
Průměr kroků do cíle	1631,1

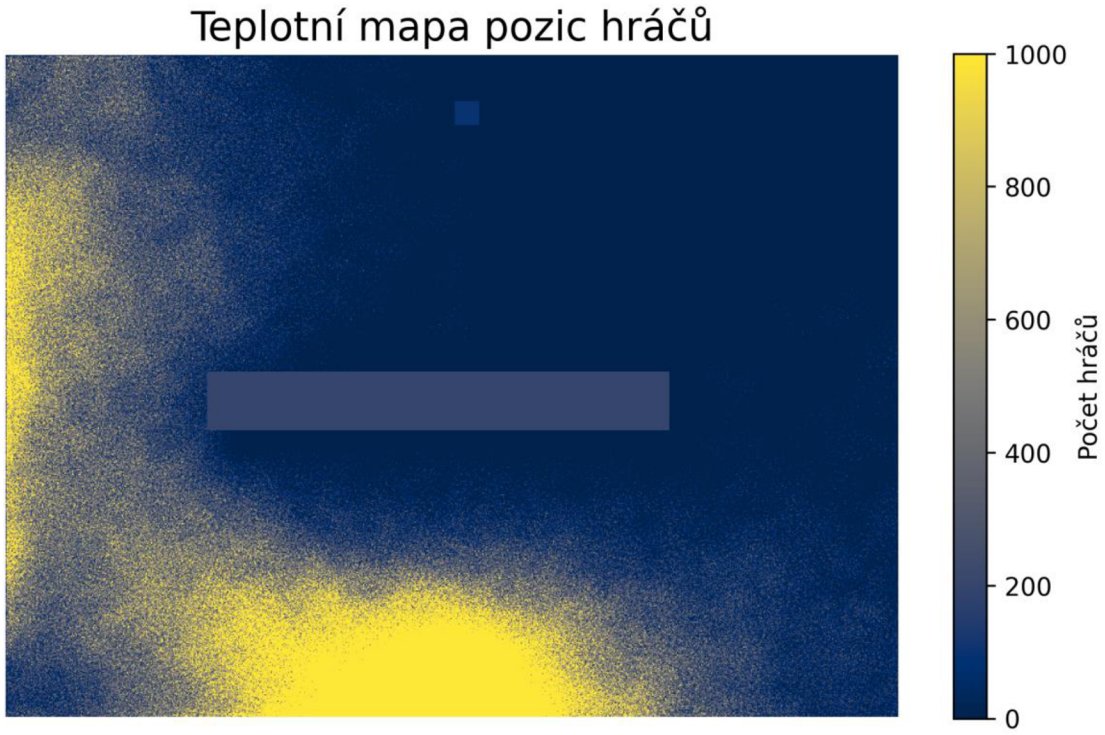
Tabulka 14 Průměry ruletového výběru a dvoubodového křížení pro jednoduchou překážku
Zdroj: [autor]

Nejvyšší počet živých hráčů	920
Nejnižší počet živých hráčů	460
Nejvyšší počet generací	44
Nejnižší počet generací	1
Nejvyšší počet kroků	1706
Nejnižší počet kroků	1514

Tabulka 15 Extrémy ruletového výběru a dvoubodového křížení pro jednoduchou překážku
Zdroj: [autor]

Generace	Průměrné fitness	Nejlepší fitness	Zbývající hráči
1	26,56	82,51	498
2	44,27	80,23	840
3	45,04	74,32	847
4	46,26	82,55	866
5	46,3	71,07	857
6	46,76	71,98	861
7	47,11	73,67	864
8	48,93	74,31	885
9	49,48	76,78	888
10	49,05	83,31	876
11	49,22	79,17	877
12	50,26	79,52	886
13	51,35	82	896
14	50,11	84,35	874
15	49,76	80,97	876
16	49,42	84,69	870
17	48,76	89,43	857
18	49,88	93,94	874
19	51,36	91,8	895
20	50,22	95,42	856
21	50,14	87,32	857
22	48,27	88,76	825
23	49,79	90,59	840
24	50,5	92,76	841
25	52,3	89,93	852
26	53,48	93,99	855
27	53,6	95,85	844
28	52,99	100	880

Tabulka 16 Příklad 7. testu ruletového výběru a dvoubodového křížení pro jednoduchou překážku
Zdroj: [autor]



Obrázek 16 Teplotní mapa 7. testu ruletového výběru a dvoubodového křížení pro jednoduchou překážku
Zdroj: [autor]

V rámci prováděného výzkumu byl proveden soubor testů, kde byla kombinována metoda ruletového výběru s dvoubodovým křížením. Cílem tohoto experimentu bylo provést podrobnou analýzu vývoje strategií UI v herním prostředí pomocí těchto dvou evolučních metod. Získané výsledky této studie prezentují průměrné charakteristiky jednotlivých generací a ukazují, jakým způsobem se EA vyvíjel směrem k dosažení optimálních strategií.

Průměrný počet generací v rámci tohoto experimentu dosáhl hodnoty 21,5. Střední hodnota fitness dosáhla úrovně 48,33, přičemž průměrná nejvyšší dosažená fitness činila 83,43.

V každém testu byla průměrná hodnota hráčů 855,69. Průměrný počet kroků do cíle byl 1631,1.

Během průběhu experimentu byly pozorovány extrémní hodnoty. Nejvyšší počet živých hráčů dosáhl hodnoty 920, zatímco nejnižší počet živých hráčů byl 460.

Maximální počet dosažených generací byl 44, zatímco minimální počet generací byl 1. Nejvyšší počet kroků byl 1706, zatímco nejnižší počet činil 1514.

Analytický pohled na vývoj strategií v rámci 7. testu od 1. do 28. generace ukázal postupný nárůst průměrné fitness od 26,56 do 52,99. Nejlepší dosažená fitness se pohybovala od 71,07do 100.

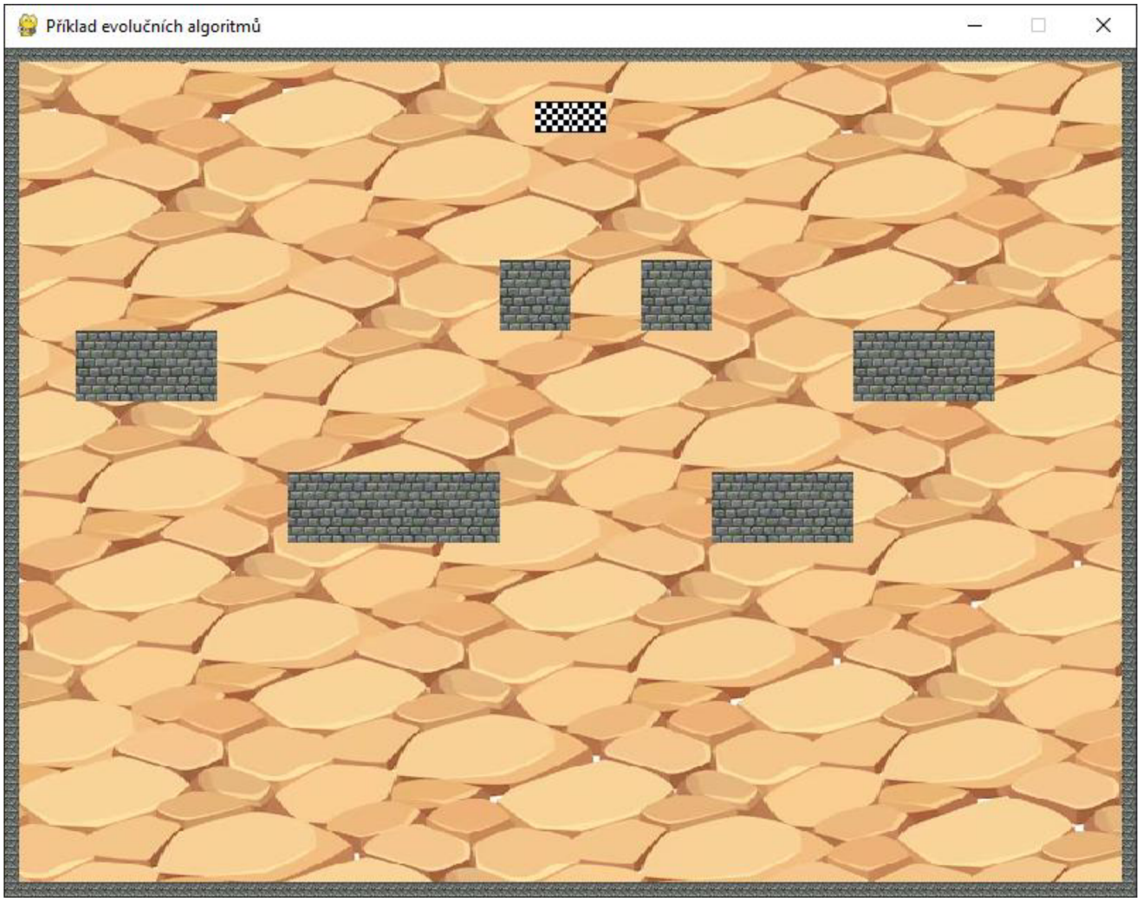
6.4.1.6 Ruletový výběr a uniformní křížení pro jednoduchou překážku

Samotný průběh evolučního experimentu, ve kterém byly kombinovány metody ruletového výběru a uniformního křížení, odhalil několik klíčových faktorů, které stály za nezdařeným výsledkem tohoto testu.

- Prvním faktorem, který přispěl k neúspěchu, je rozdíl v efektivitě mezi ruletovým výběrem a turnajovým výběrem. Na rozdíl od turnajového výběru, který má tendenci lépe selektovat silné jedince a vytvářet vyvážené populace, se ukázalo, že ruletový výběr nedokázal efektivně vybrat nejsilnější jedince. Tato nerovnováha ve výběru mohla způsobit, že kvalitní strategie nebyly dostatečně prosazeny a neměly tak možnost se rozšířit v populaci.
- Druhým faktorem byla nedostatečná účinnost uniformního křížení. Zatímco turnajový výběr byl kombinován s různými druhy křížení, tentokrát se ukázalo, že uniformní křížení v kombinaci s ruletovým výběrem není tak efektivní při vytváření nových potomků. Tím pádem neměly nové generace potřebný genetický materiál k postupnému zdokonalování strategií.
- Třetím klíčovým faktorem byla skutečnost, že evoluční proces uvízl v lokálních optimálních řešeních. Nedostatečná schopnost kombinace ruletového výběru a uniformního křížení průběžně prozkoumávat nové oblasti prostoru řešení vedla k tomu, že populace hráčů se postupně uzavírala kolem suboptimálních strategií a neměla tak možnost objevovat lepší řešení mimo tyto lokální optimální body.

6.4.2 Testy složitých překážek

Pro rozšíření experimentů na složité překážky bylo do herního prostředí zavedeno více specifických překážek. Hlavním záměrem této modifikace je umožnit hráčům naučit se efektivní způsoby, jak překonat tyto náročnější překážky za použití evolučních metod. Cílem je analyzovat, jak se EA podaří adaptovat na nové a obtížnější podmínky a jak budou schopni vyvinout strategie, které umožní úspěšné překonání složité překážky a dosažení stanoveného cíle.



Obrázek 17 Složité překážky
Zdroj: [autor]

6.4.2.1 Turnajový výběr a jednobodové křížení pro složité překážky

Pro tento specifický experiment jsou relevantní a platí stejné výchozí podmínky a předpoklady, jaké byly uvedeny v části týkající se jednoduché překážky viz 6.4.1.1.

Průměrný počet generací	14,5
Střední hodnota fitness	29,57
Průměr nejvyšší fitness	88,02
Průměr zbývajících hráčů	425,61
Průměr kroků do cíle	1692,5

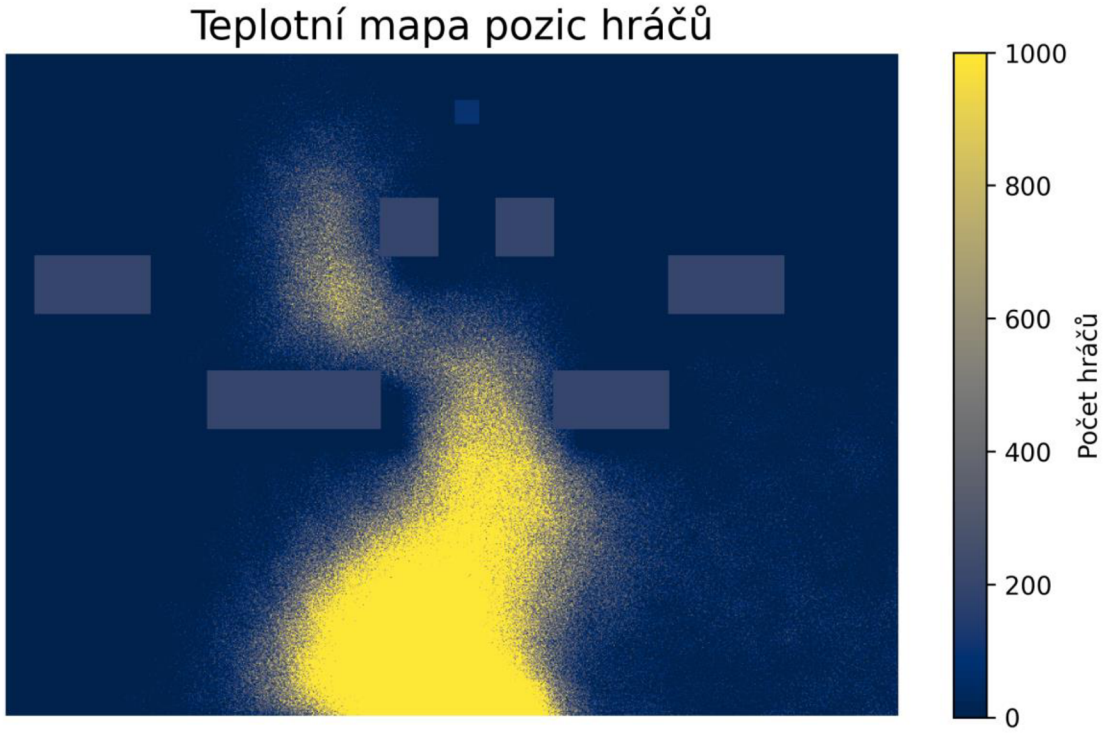
Tabulka 17 Průměry turnajového výběru a jednobodového křížení pro složité překážky
Zdroj: [autor]

Nejvyšší počet živých hráčů	831
Nejnižší počet živých hráčů	149
Nejvyšší počet generací	24
Nejnižší počet generací	9
Nejvyšší počet kroků	1724
Nejnižší počet kroků	1623

Tabulka 18 Extrémy turnajového výběru a jednobodového křížení pro složité překážky
Zdroj: [autor]

Generace	Průměrné fitness	Nejlepší fitness	Zbývající hráči
1	28,12	76,02	528
2	46,96	79,16	795
3	47,29	81,59	736
4	39,7	87,15	579
5	29,92	94,21	420
6	25,36	92,19	346
7	28,35	90,96	378
8	24,34	93,63	313
9	20,59	93,36	266
10	17,75	94,46	226
11	16,85	100	202

Tabulka 19 Příklad 2. testu turnajového výběru a jednobodového křížení pro složité překážky
Zdroj: [autor]



Obrázek 18 Teplotní mapa 2. testu turnajového výběru a jednobodového křížení pro složité překážky
Zdroj: [autor]

V rámci rozšiřujícího výzkumu byla uskutečněna série testů, ve kterých byla zkombinována metoda turnajového výběru s jednobodovým křížením pro řešení složitých překážek v herním prostředí. Hlavním účelem tohoto experimentu bylo pečlivě analyzovat dynamiku vývoje strategií u UI v reakci na tyto dvě evoluční metody. Výsledky této studie nám přináší průměrné charakteristiky jednotlivých generací a odhalují, jakým způsobem se EA vyvíjel směrem k nalezení optimálních strategií pro překonání komplexních překážek.

Průměrný počet generací v rámci těchto testů dosáhl hodnoty 14,5. Střední hodnota fitness dosáhla úrovně 29,57, zatímco průměrná nejvyšší dosažená fitness činila 88,02.

Zbývající počet hráčů v každém testu dosáhl průměrné hodnoty 425,61, což odráží schopnost EA zachovávat rozmanitost v populaci hráčů při řešení složitých překážek. Průměrný počet kroků do cíle činil 1692,5.

Během průběhu experimentu byly pozorovány extrémní hodnoty. Nejvyšší počet živých hráčů dosáhl hodnoty 831, zatímco nejnižší počet živých hráčů byl 149.

Tato variabilita poukazuje na různorodost strategií v populaci hráčů, kteří se snažili efektivně překonávat náročné překážky. Maximální počet dosažených generací byl 24, zatímco minimální počet generací byl 9.

Nejvyšší počet kroků dosáhl hodnoty 1724, zatímco nejnižší počet kroků byl 1623.

Analýza vývoje 2. testu od 1. do 22. generace ukázala postupný pokles průměrné fitness od 28,12 do 16,85. Průměrná fitness klesá z důvodu zvýšené úmrtnosti hráčů při jejich pokusu dosáhnout cíle. Nejlepší dosažená fitness se pohybovala od 76,02 do maximální hodnoty 100. Tímto způsobem byla potvrzena schopnost EA adaptovat se na náročné podmínky a postupně hledat optimální strategie pro úspěšné překonání složitých překážek v herním prostředí.

6.4.2.2 Turnajový výběr a dvoubodové křížení pro složité překážky

V kontextu tohoto konkrétního experimentu jsou důležité a platí podobné výchozí podmínky a předpoklady jako byly prezentovány v sekci, která se věnovala jednoduché překážce viz 6.4.1.2.

Průměrný počet generací	14,9
Střední hodnota fitness	27,37
Průměr nejvyšší fitness	88,76
Průměr zbývajících hráčů	400,04
Průměr kroků do cíle	1699

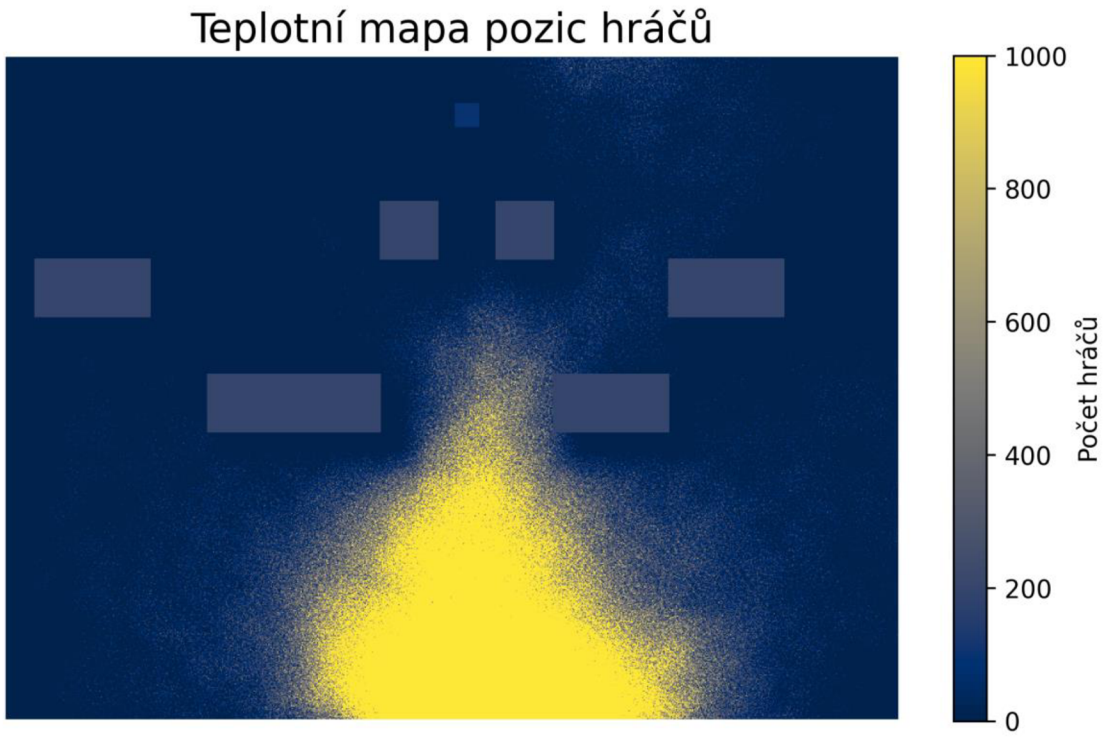
Tabulka 20 Průměry turnajového výběru a dvoubodového křížení pro složité překážky
Zdroj: [autor]

Nejvyšší počet živých hráčů	825
Nejnižší počet živých hráčů	175
Nejvyšší počet generací	22
Nejnižší počet generací	8
Nejvyšší počet kroků	1721
Nejnižší počet kroků	1622

Tabulka 21 Extrémy turnajového výběru a dvoubodového křížení pro složité překážky
Zdroj: [autor]

Generace	Průměrné fitness	Nejlepší fitness	Zbývající hráči
1	29,99	78,78	571
2	44,55	81,08	776
3	45,83	82,11	743
4	39,1	85,05	600
5	29,76	92,1	446
6	24,09	89,8	346
7	21,57	93,67	297
8	28,13	100	402

Tabulka 22 Příklad 8. testu turnajového výběru a dvoubodového křížení pro složité překážky
Zdroj: [autor]



Obrázek 19 Teplotní mapa 8. testu turnajového výběru a dvoubodového křížení pro složité překážky
Zdroj: [autor]

V rámci experimentu byla provedena série testů, ve kterých byla kombinována metoda turnajového výběru s dvoubodovým křížením pro řešení složitých překážek v herním prostředí. Hlavním cílem tohoto experimentu bylo důkladně analyzovat dynamiku vývoje strategií u UI v reakci na tyto dvě evoluční metody. Získané výsledky této studie poskytují průměrné charakteristiky

jednotlivých generací a odhalují, jakým způsobem se EA postupně vyvíjel směrem k nalezení optimálních strategií pro překonání komplexních překážek.

Průměrný počet generací v rámci těchto testů dosáhl hodnoty 14,9. Střední hodnota fitness dosáhla úrovně 27,37, zatímco průměrná nejvyšší dosažená fitness činila 88,76. Průměrný počet zbývajících hráčů v každém testu dosáhl hodnoty 400,04. Průměrný počet kroků do cíle činil 1699.

Během průběhu experimentu byly pozorovány extrémní hodnoty. Nejvyšší počet živých hráčů dosáhl hodnoty 825, zatímco nejnižší počet živých hráčů byl 175. Maximální počet dosažených generací byl 22, minimální počet generací činil 8. Nejvyšší počet kroků dosáhl hodnoty 1721, zatímco nejnižší počet kroků byl 1622.

Detailní analýza vývoje 8. testu od 1. do 8. generace odhalila postupný pokles průměrné fitness od 29,99 až po hodnotu 28,13. Pokles průměrné fitness je způsoben zvýšenou úmrtností hráčů v průběhu jejich snahy dosáhnout cílového bodu. Nejlepší dosažená fitness se pohybovala od 78,78 až po maximální hodnotu 100.

6.4.2.3 Turnajový výběr a uniformní křížení pro složité překážky

V rámci tohoto specifického experimentu jsou relevantní a platí obdobné výchozí podmínky a předpoklady, které byly popsány v části věnované jednoduché překážce viz 6.4.1.3.

Průměrný počet generací	11,7
Střední hodnota fitness	27,93
Průměr nejvyšší fitness	87,23
Průměr zbývajících hráčů	452
Průměr kroků do cíle	1551,9

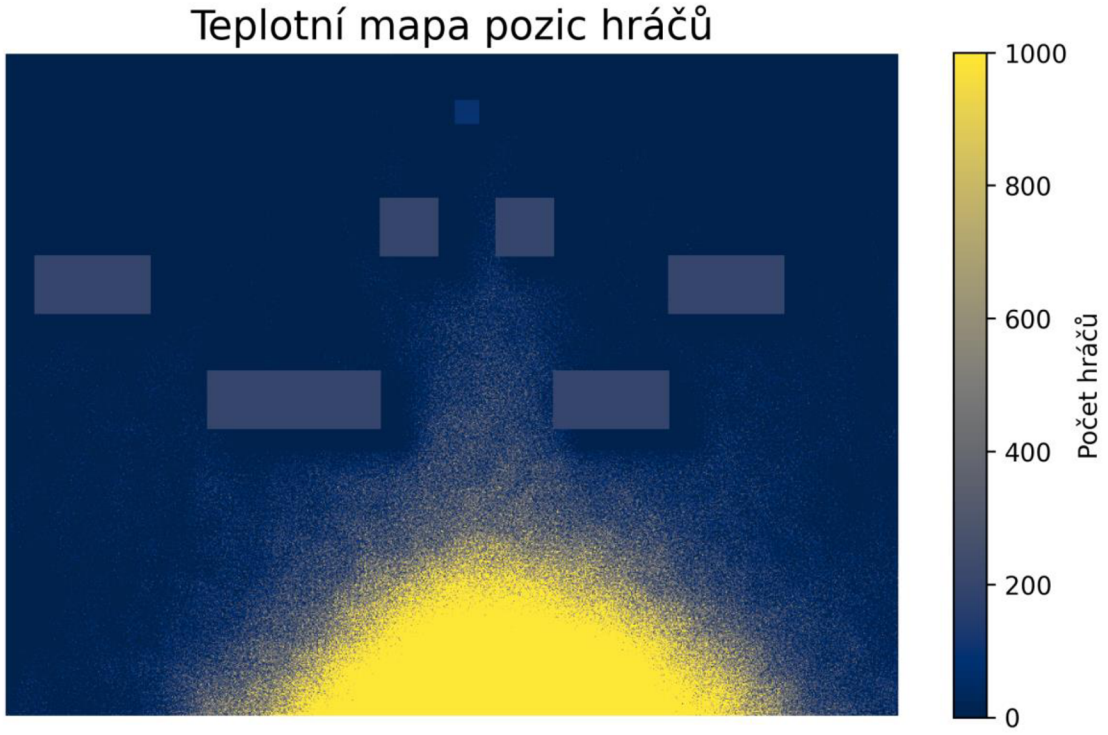
Tabulka 23 Průměry turnajového výběru a uniformního křížení pro složité překážky
Zdroj: [autor]

Nejvyšší počet živých hráčů	753
Nejnižší počet živých hráčů	164
Nejvyšší počet generací	20
Nejnižší počet generací	6
Nejvyšší počet kroků	1586
Nejnižší počet kroků	1486

Tabulka 24 Extrémy turnajového výběru a uniformního křížení pro složité překážky
Zdroj: [autor]

Generace	Průměrné fitness	Nejlepší fitness	Zbývající hráči
1	27,8	81,35	525
2	39,79	85,41	707
3	39,22	84,45	657
4	35,31	87,36	579
5	31,15	86,63	495
6	28,71	86,57	443
7	26,27	86,38	407
8	21,9	86,65	329
9	21,32	92,02	317
10	20,89	89,74	306
11	21,05	90,88	310
12	14,03	92,3	199
13	20,81	90,24	301
14	27,81	87,04	403
15	20,33	92,77	285
16	18,6	90,75	261
17	22,93	92,99	318
18	13,84	91,9	185
19	12,11	94,99	164
20	13,93	100	187

Tabulka 25 Příklad 5. testu turnajového výběru a uniformního křížení pro složité překážky
Zdroj: [autor]



Obrázek 20 Teplotní mapa 5. testu turnajového výběru a uniformního křížení pro složité překážky
Zdroj: [autor]

Během experimentu byla provedena série testů, ve kterých byla kombinována metoda turnajového výběru s uniformním křížením pro řešení složitých překážek v herním prostředí. Hlavním cílem tohoto experimentu bylo pečlivě analyzovat dynamiku vývoje strategií u UI v reakci na tuto kombinaci evolučních metod. Výsledky této studie představují průměrné charakteristiky jednotlivých generací a ukazují, jak se EA postupně vyvíjel směrem k nalezení optimálních strategií pro překonání komplexních překážek.

Průměrný počet generací v rámci těchto testů dosáhl hodnoty 11,7. Střední hodnota fitness činila 27,93, zatímco průměrná nejvyšší dosažená fitness činila 87,23. Průměrný počet zbývajících hráčů v každém testu byl 452. Průměrný počet kroků do cíle byl 1551,9.

Během průběhu experimentu byla zaznamenána rozmanitost výsledků. Nejvyšší počet živých hráčů dosáhl hodnoty 753, zatímco nejnižší počet živých hráčů byl 164. Maximální počet dosažených generací činil 20, zatímco minimální počet

generací byl 6. Nejvyšší počet kroků dosáhl hodnoty 1586, zatímco nejnižší počet kroků byl 1486.

Detailní analýza vývoje v rámci 5. testu od 1. do 20. generace ukázala postupný pokles průměrné fitness od 27,8 až po hodnotu 13,93. Tento pokles průměrné fitness lze přičíst zvýšené úmrtnosti hráčů během jejich pokusů dosáhnout cíle. Nejlepší dosažená fitness se pohybovala od 81,35 až po hodnotu 100.

6.4.2.4 Ruletový výběr a jednobodové křížení pro složité překážky

Experiment s ruletovým výběrem a jednobodovým křížením pro složité překážky nebylo možné úspěšně dokončit z několika důvodů, z nichž jeden klíčový byl, že i přes průběh stovky generací hráči nedosáhli stanoveného cíle. Průběh experimentu ukázal, že tato kombinace evolučních metod nedokázala efektivně vést k vývoji strategií, které by umožnily hráčům úspěšně překonat náročné překážky a dosáhnout cíle v herním prostředí.

Jedním z vysvětlení tohoto neúspěchu může být, že ruletový výběr, který se spoléhá na pravděpodobnostní distribuci fitness hodnot jedinců, nemusel efektivně zohlednit klíčové aspekty prostředí s komplexními překážkami. V kombinaci s jednobodovým křížením, které může mít omezenou schopnost zachovávat a kombinovat klíčové části genetického materiálu, se mohlo stát, že nedostatečně diverzifikovaná populace nedokázala vyvinout strategie pro úspěšné navigování složitým terénem a dosažení cíle.

Dalším faktorem bylo uvíznutí populace v lokálních optimálních řešeních, kdy se EA nedokázal efektivně vymanit z určitého úzkého rozmezí hodnot fitness a nepodařilo se tak dosáhnout diverzifikace a hledání lepších strategií.

Kombinace ruletového výběru a jednobodového křížení nedostatečně odpovídala nárokům složitého herního prostředí s překážkami a nedokázala efektivně vést k nalezení optimálních strategií pro úspěšné překonání těchto překážek.

6.4.2.5 Ruletový výběr a dvoubodové křížení pro složité překážky

Experiment s ruletovým výběrem a dvoubodovým křížením pro složité překážky nebyl úspěšně dokončen z důvodu, že se hráčům nepodařilo dosáhnout

stanoveného cíle. Tento výsledek naznačuje, že zvolená kombinace evolučních metod není pro daný typ problému optimální a nesplňuje požadavky složitého herního prostředí s komplexními překážkami viz 6.4.2.4.

6.4.2.6 Ruletový výběr a uniformní křížení pro složité překážky

Experiment s ruletovým výběrem a uniformním křížením pro složité překážky se nepodařilo dokončit stejně, jak tomu bylo pro jednoduchou překážku, detailní popis viz 6.4.1.6.

7 Shrnutí výsledků

V rámci této diplomové práce byly provedeny rozsáhlé experimenty, které zkoumaly schopnost adaptace UI k překonávání překážek a hledání cíle v herním prostředí.

V případě jednoduchých překážek se osvědčila kombinace turnajového výběru a jednobodového křížení. Tato strategie vedla k významnému zlepšení adaptace UI na prostředí s jednoduchými překážkami. Průměrná fitness se zvyšovala a UI dosahovala vysokých výkonů, což ukazuje na schopnost hráčů vyvíjet efektivní strategie během generací. Podobných výsledků dosáhla také kombinace turnajového výběru a dvoubodového křížení. Naopak, pokus o využití ruletového výběru a uniformního křížení nebyl úspěšný. UI neprojevila schopnost dosáhnout cílových bodů překážek, a to i přes průběh stovek generací.

Při zkoumání adaptace UI na složité překážky byla nejúspěšnější kombinace turnajového výběru a uniformního křížení. Tato strategie umožnila UI lépe se přizpůsobit komplexnímu prostředí složitých překážek. Naopak, ruletový výběr se ukázal jako nevhodný pro efektivní adaptaci na tuto náročnou úroveň překážek, přičemž výsledky byly neuspokojivé v rámci všech testovaných variant křížení.

Turnajový výběr se ukázal jako lepší selekční přístup, zatímco ruletový výběr vyžadoval značně specifické podmínky pro úspěšnou adaptaci.

8 Závěry a doporučení

Výsledky této diplomové práce přinášejí poznatky ohledně adaptace UI na prostředí s různými druhy překážek. Analýzou experimentů lze potvrdit úspěšnost kombinace turnajového výběru a jednobodového křížení při překonávání jednoduchých překážek. Tato strategie se ukázala jako optimální pro efektivní reakci UI na dané prostředí.

Naopak ruletový výběr v kombinaci s uniformním křížením nebyl schopný dosáhnout cíle při adaptaci na jednoduché překážky. Tato situace může být ovlivněna různorodostí prostředí, které ruletový výběr nebyl schopen účinně prozkoumat. Byly identifikovány jisté mezery ve schopnosti ruletového výběru adaptovat se na různé úkoly a tato neúspěšnost může být také spojena s náhodným faktorem této metody.

V případě složitých překážek byla strategie s turnajovým výběrem a uniformním křížením nejúspěšnější. Tento výsledek může signalizovat vyšší flexibilitu turnajového výběru v adaptaci na různorodá prostředí a zdůrazňuje důležitost volby evolučních strategií v závislosti na konkrétní problematice. Ruletový výběr v kombinaci s různými druhy křížení nesplnil očekávání v prostředí složitých překážek. Nedokázal efektivně prozkoumat rozmanité scénáře pro překonání překážek a dosáhnutí cíle.

Doporučení pro budoucí výzkum by mohla směřovat k hlubší analýze mechanismů, které ovlivňují úspěšnost jednotlivých evolučních strategií. Je také vhodné zkoumat vliv parametrů selekce a křížení na adaptaci UI a provést rozsáhlejší srovnání s alternativními metodami optimalizace. Dále je možné rozšířit tuto studii na komplexnější herní scénáře a prostředí, která lépe simulují reálné výzvy.

V závěrečných úvahách lze také diskutovat o alternativních možnostech a technologiích, které by mohly být použity místo jazyka Python a knihovny Pygame pro implementaci herního prostředí. Byly identifikovány limity týkající se výkonu, které mohou omezovat použitelnost této kombinace pro některé složitější úlohy. V případě nároků na vysoký výkon a efektivitu výpočtů by bylo vhodné zvážit

použití jiných programovacích jazyků, jako je C++ nebo Java. Tyto jazyky mohou poskytnout lepší výkon při zpracování komplexních simulací a výpočetních úloh.

Dalším hlediskem je možnost využít specializovaných herních enginů a rámců, které jsou navrženy specificky pro tvorbu her a simulací. Tyto nástroje mohou poskytnout pokročilé funkce pro vizualizace, fyziku, kolize a interakce, což by mohlo vést k realističtějším a sofistikovanějším herním prostředím. Příkladem může být Unreal Engine nebo Unity.

Lze konstatovat, že výzkum adaptace UI v kontextu herních překážek otevřel dveře k lepšímu porozumění mechanismům evolučních strategií a jejich interakcí s prostředím. Práce poskytuje základ pro další pokroky v tomto směru a inspiruje k dalšímu rozvoji technik UI ve světě her.

9 Seznam použité literatury

[1] How Heuristics Help You Make Quick Decisions. *Verywell Mind* [online]. [vid. 2023-01-19]. Dostupné z: https://www.verywellmind.com/what-is-a-heuristic-2795235

[2] HERTWIG, Ralph a Thorsten PACHUR. Heuristics, History of. In: *International Encyclopedia of the Social & Behavioral Sciences* [online]. B.m.: Elsevier, 2015 [vid. 2023-01-19], s. 829–835. ISBN 978-0-08-097087-5. Dostupné z: doi:10.1016/B978-0-08-097086-8.03221-9

[3] FIDORA, Alexander. *Ramon Llull: From the Ars Magna to Artificial Intelligence*. B.m.: Artificial Intelligence Research Inst., 2011. ISBN 978-84-694-5185-4.

[4] HTTPS://WWW.FACEBOOK.COM/48576411181. *In the 17th Century, Leibniz Dreamed of a Machine That Could Calculate Ideas - IEEE Spectrum* [online]. [vid. 2023-08-08]. Dostupné z: https://spectrum.ieee.org/in-the-17th-century-leibniz-dreamed-of-a-machine-that-could-calculate-ideas

[5] JOHNSON, Dale M. Prelude to dimension theory: The geometrical investigations of Bernard Bolzano. *Archive for History of Exact Sciences* [online]. 1977, **17**(3), 261–295. ISSN 1432-0657. Dostupné z: doi:10.1007/BF00499625

[6] GRONER, R., M. GRONER a W. F. BISCHOF. *Methods of Heuristics*. B.m.: Routledge, 2014. ISBN 978-1-317-83849-4.

[7] GAWIEJNOWICZ, Stanisław. Heuristic algorithms. In: Stanisław GAWIEJNOWICZ, ed. *Models and Algorithms of Time-Dependent Scheduling* [online]. Berlin, Heidelberg: Springer, 2020 [vid. 2023-01-20], Monographs in Theoretical Computer Science. An EATCS Series, s. 301–327. ISBN 978-3-662-59362-2. Dostupné z: doi:10.1007/978-3-662-59362-2_16

[8] LIANG, Frank. Optimization Techniques — Simulated Annealing. *Medium* [online]. 21. duben 2020 [vid. 2023-01-20]. Dostupné z: https://towardsdatascience.com/optimization-techniques-simulated-annealing-d6a4785a1de7

[9] LIANG, Frank. Optimization Techniques — Tabu Search. *Medium* [online]. 27. červenec 2020 [vid. 2023-01-20]. Dostupné z: https://towardsdatascience.com/optimization-techniques-tabu-search-36f197ef8e25

[10] *Genetic Programming for Production Scheduling* [online]. nedatováno [vid. 2023-01-21]. Dostupné z: https://link.springer.com/book/10.1007/978-981-16-4859-5

[11] *Aplikované evoluční algoritmy - cvičení 2* [online]. [vid. 2023-01-21]. Dostupné z: http://www.fit.vutbr.cz/~simekv/EVO/cv2/.en

[12] PARDALOS, Panos M., A. MIGDALAS a Leonidas PITSOULIS. *Pareto Optimality, Game Theory and Equilibria*. B.m.: Springer Science & Business Media, 2008. ISBN 978-0-387-77247-9.

[13] OBAYASHI, Shigeru, Kalyanmoy DEB, Carlo POLONI, Tomoyuki HIROYASU a Tadahiko MURATA, ed. *Evolutionary Multi-Criterion Optimization* [online]. Berlin, Heidelberg: Springer, 2007 [vid. 2023-07-26]. Lecture Notes in Computer Science. ISBN 978-3-540-70927-5. Dostupné z: doi:10.1007/978-3-540-70928-2

[14] KRAMER, Oliver. Genetic Algorithms. In: Oliver KRAMER, ed. *Genetic Algorithm Essentials* [online]. Cham: Springer International Publishing, 2017 [vid. 2023-01-24], Studies in Computational Intelligence, s. 11–19. ISBN 978-3-319-52156-5. Dostupné z: doi:10.1007/978-3-319-52156-5_2

[15] JEBARI, Khalid. Selection Methods for Genetic Algorithms. *International Journal of Emerging Sciences*. 2013, **3**, 333–344.

[16] BAECK, Thomas, D. B. FOGEL a Z. MICHALEWICZ. *Evolutionary Computation 1: Basic Algorithms and Operators*. B.m.: CRC Press, 2018. ISBN 978-1-351-98942-8.

[17] PURSHOUSE, Robin C a Peter J FLEMING. Why use Elitism and Sharing in a Multi-Objective Genetic Algorithm? nedatováno.

[18] DAGDIA, Zaineb Chelly a Miroslav MIRCHEV. Chapter 15 - When Evolutionary Computing Meets Astro- and Geoinformatics. In: Petr ŠKODA a Fathalrahman ADAM, ed. *Knowledge Discovery in Big Data from Astronomy and Earth Observation* [online]. B.m.: Elsevier, 2020 [vid. 2023-01-22], s. 283–306. ISBN 978-0-12-819154-5. Dostupné z: doi:10.1016/B978-0-12-819154-5.00026-6

[19] *Genetic Algorithms - Mutation* [online]. [vid. 2023-01-23]. Dostupné z: https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_mutation.htm

[20] BRAMEIER, Markus F. a Wolfgang BANZHAF, ed. A Comparison with Tree-Based Genetic Programming. In: Markus F. BRAMEIER a Wolfgang BANZHAF, ed. *Linear Genetic Programming* [online]. Boston, MA: Springer US, 2007 [vid. 2023-07-26], Genetic and Evolutionary Computation, s. 173–192. ISBN 978-0-387-31030-5. Dostupné z: doi:10.1007/978-0-387-31030-5_8

[21] *Linear Genetic Programming* [online]. Boston, MA: Springer US, 2007 [vid. 2023-07-26]. Genetic and Evolutionary Computation. ISBN 978-0-387-31029-9. Dostupné z: doi:10.1007/978-0-387-31030-5

[22] SAHA, Rajarsi. An insight into the concept of Genetic Algorithm. *Medium* [online]. 19. září 2020 [vid. 2023-01-24]. Dostupné z: https://medium.datadriveninvestor.com/an-insight-into-genetic-algorithms-93428953c098

[23] GREINER, David, Jacques PERIAUX, Jose M. EMPERADOR, Blas GALVÁN a Gabriel WINTER. Game Theory Based Evolutionary Algorithms: A Review with Nash Applications in Structural Engineering Optimization Problems. *Archives of Computational Methods in Engineering* [online]. 2017, **24**(4), 703–750. ISSN 1886-1784. Dostupné z: doi:10.1007/s11831-016-9187-y

[24] LIU, Baoding. Stackelberg-Nash equilibrium for multilevel programming with multiple followers using genetic algorithms. *Computers & Mathematics with Applications* [online]. 1998, **36**(7), 79–89. ISSN 0898-1221. Dostupné z: doi:10.1016/S0898-1221(98)00174-6

[25] LEWONTIN, R. C. Evolution and the theory of games. *Journal of Theoretical Biology* [online]. 1961, **1**(3), 382–403. ISSN 0022-5193. Dostupné z: doi:10.1016/0022-5193(61)90038-8

[26] BACAËR, Nicolas. Game theory and evolution (1973). In: Nicolas BACAËR, ed. *A Short History of Mathematical Population Dynamics* [online]. London: Springer, 2011 [vid. 2023-08-08], s. 127–131. ISBN 978-0-85729-115-8. Dostupné z: doi:10.1007/978-0-85729-115-8_23

[27] RIECHMANN, Thomas. Genetic algorithm learning and evolutionary games. *Journal of Economic Dynamics and Control* [online]. 2001, **25**(6), Computing, economic dynamics, and finance, 1019–1037. ISSN 0165-1889. Dostupné z: doi:10.1016/S0165-1889(00)00066-X

[28] LEWIS, Kemper a Farrokh MISTREE. Modeling Interactions in Multidisciplinary Design: A Game Theoretic Approach. *AIAA Journal* [online]. 1997, **35**(8), 1387–1392. ISSN 0001-1452. Dostupné z: doi:10.2514/2.248

[29] FISHER, R. A. The evolution of sexual preference. *The Eugenics Review*. 1915, **7**(3), 184–192.

[30] ALEXANDER, J. McKenzie. Evolutionary Game Theory. In: Edward N. ZALTA, ed. *The Stanford Encyclopedia of Philosophy* [online]. Summer 2021. B.m.: Metaphysics Research Lab, Stanford University, 2021 [vid. 2023-02-25]. Dostupné z: https://plato.stanford.edu/archives/sum2021/entries/game-evolutionary/

[31] DANIEL. Seven Principles of Game Design and Five Innovation Games that work. *The Design Gym* [online]. 20. říjen 2014 [vid. 2023-04-01]. Dostupné z: https://www.thedesigngym.com/seven-principles-of-game-design-and-five-innovation-games-that-work/

[32] LAINE, Teemu H. a Renny S. N. LINDBERG. Designing Engaging Games for Education: A Systematic Literature Review on Game Motivators and Design Principles. *IEEE Transactions on Learning Technologies* [online]. 2020, **13**(4), 804–821. ISSN 1939-1382. Dostupné z: doi:10.1109/TLT.2020.3018503

[33] KUSIAK, Magdalena, Karol WALĘDZIK a Jacek MAŃDZIUK. Evolutionary Approach to the Game of Checkers. In: [online]. 2007, s. 432–440. ISBN 978-3-540-71589-4. Dostupné z: doi:10.1007/978-3-540-71618-1_48

[34] KUSIAK, Magdalena, Karol WALĘDZIK a Jacek MAŃDZIUK. Evolution of Heuristics for Give-Away Checkers. *Lecture Notes in Computer Science*. 2005, 981. ISSN 0302-9743.

[35] MAŃDZIUK, Jacek a Daniel OSMAN. Temporal Difference Approach to Playing Give-Away Checkers. In: Leszek RUTKOWSKI, Jörg H. SIEKMANN, Ryszard TADEUSIEWICZ a Lotfi A. ZADEH, ed. *Artificial Intelligence and Soft Computing - ICAISC 2004* [online]. Berlin, Heidelberg: Springer, 2004, s. 909–914. Lecture Notes in Computer Science. ISBN 978-3-540-24844-6. Dostupné z: doi:10.1007/978-3-540-24844-6_141

[36] *ec_and_games_lucas.pdf* [online]. [vid. 2023-04-03]. Dostupné z: https://www.cs.montana.edu/courses/spring2007/536/materials/ec_and_games_lucas.pdf

[37] MILLINGTON, Ian a John FUNGE. *Artificial Intelligence for Games*. B.m.: CRC Press, 2009. ISBN 978-0-08-088503-2.

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání diplomové práce

Autor:	**Bc. David Vondráček**
Studium:	I2100850
Studijní program:	N0688A140019 Datová věda
Studijní obor:	Datová věda
Název diplomové práce:	**Evoluční algoritmy a herní situace**
Název diplomové práce AJ:	Evolutionary algorithms and game situations

Cíl, metody, literatura, předpoklady:

Cíl: Analýza dané herní situace a následné implementování nejvhodnějšího evolučního algoritmu

1. Úvod
2. Heuristika
3. Evoluční a Genetické algoritmy
4. Návrh a realizace ukázkového řešení
5. Závěr

Zelinka, Ivan. Evoluční výpočetní techniky: principy a aplikace. Praha: BEN, 2009. ISBN 80-7300-218-3.
Hynek, Josef. Genetické algoritmy a genetické programování. Praha: Grada, 2008. ISBN 8024726955.

Deng, W., Shang, S., Cai, X. et al. An improved differential evolution algorithm and its application in optimization problem. Soft Comput 25, 5277–5298 (2021).

Zadávající pracoviště:	Katedra informačních technologií, Fakulta informatiky a managementu
Vedoucí práce:	Ing. Karel Mls, Ph.D.
Oponent:	Ing. Martina Husáková, Ph.D.
Datum zadání závěrečné práce:	15.10.2021