Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Umělá inteligence v počítačových hrách

Bakalářská práce

Autor: Michal Horák
Studijní obor: ai3-p

Vedoucí práce: Ing. Jakub Beneš

Hradec Králové Květen 2023

Prohlášení:

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

vlastnoruční podpis

V Hradci Králové dne *27.4.*2023 Michal Horák

Poděkování:

Děkuji vedoucímu bakalářské práce Ing. Jakubovi Benešovi za metodické vedení práce a podporu.

Anotace

Práce obsahuje průzkum jednotlivých metod Umělé inteligence v počítačových hrách. Hodnotí jejich využití v počítačových hrách a herním průmyslu. Vysvětluje použité technologie pro implementaci demonstračního kódu. Demonstruje získané poznatky pomocí hry Minecraft.

Annotation

This thesis includes the research of individual methods of artificial intelligence in games. Evaluates their use in computer games and the gaming industry. Explains technologies used for the demo code. Demonstrates gained knowledge using the game Minecraft.

Obsah

1 Úvod

Umělá inteligence (AI) zažila velký progres v posledních letech. Je to jak prosperující výzkumná oblast s rostoucím počtem důležitých výzkumných oblastí, tak základní technologie pro stále větší počet aplikačních oblastí [1]. Cílem AI je usnadnit, vylepšit nebo v některých případech i nahradit určitou úlohu vykonávanou člověkem. Je založená na vzdělávacích algoritmech a statistických modelech, které ji umožňují identifikovat souvislosti, ty poté využívá pro splnění svých úkolů. Je mnoho informací o tom, jak AI funguje, co obnáší apod.

Již od zrodu myšlenky umělé inteligence pomáhají hry pokroku ve výzkumu AI. Hry nepředstavují jen zajímavé a složité problémy, aby umělá inteligence vyřešila – např. dobře hrát hru; nabízejí také plátno pro kreativitu a výraz, který uživatelé (lidé, nebo dokonce stroje!) zažívají. Tím pádem, hry jsou pravděpodobně vzácnou doménou, kde se věda (řešení problémů) setkává s uměním a interakce: tyto ingredience udělaly z her jedinečnou a oblíbenou doménu studium AI [1].

2 Cíl práce

Práce prozkoumá oblast umělé inteligence a popíše její jednotlivé charakteristiky. Vezme v potaz využití AI v herním průmyslu a případně je zmíní. Následovně získané informace demonstruje pomocí zvolené hry. Popíše využité technologie potřebné k vytvoření demonstrace. Nakonec představí vytvořenou demonstraci a její návaznost na získané poznatky o umělé inteligenci.

3 Umělá inteligence počítačových her

Umělá inteligence uvnitř počítačových her umožňuje Non Player Character (NPC) se rozhodovat, pohybovat se uvnitř herního světa, nebo interagovat se svým okolím, hráčem nebo dalšími NPC. Pro dosažení věrohodné umělé inteligence se také využívají metody strojového učení pro dosažení více realistického rozhodování, interakce atd.

Mezi nejčastější metody vytváření umělé inteligence patří:

1. **Finite State Machines (FSM):** jde o definování stavů, kterých může umělá inteligence nabývat, například "nečinnost", "útok" nebo "útěk", a podmínek při kterých se tyto stavy změní.
2. **Behavior Trees (BT)**: jde o typ hierarchické definice stavů, které umožňují komplexnější chování. BT rozděluje jednotlivé akce do uzlů, a NPC se rozhoduje jakou akci zvolí na základě podmínek dětských a rodičovských uzlů.
3. **Decision Trees (DT)**: používají se pro vytváření rozhodovacího procesu NPC, kterému jsou představeny jednotlivé možnosti a DT rozhodne jakou vybrat na základě určitých podmínek či atributech.
4. **Pathfinding**: proces hledání nejlepší možné cesty NPC pro dosažení cíle.
5. **Machine learning**: metody na základě podpůrného učení a neuronových sítí se využívají pro napodobení více věrohodnotné umělé inteligence.
6. **Rule-based systems**: jasně definovaná pravidla, které NPC musí dodržet, s účelem rychlého rozhodování a efektivity.

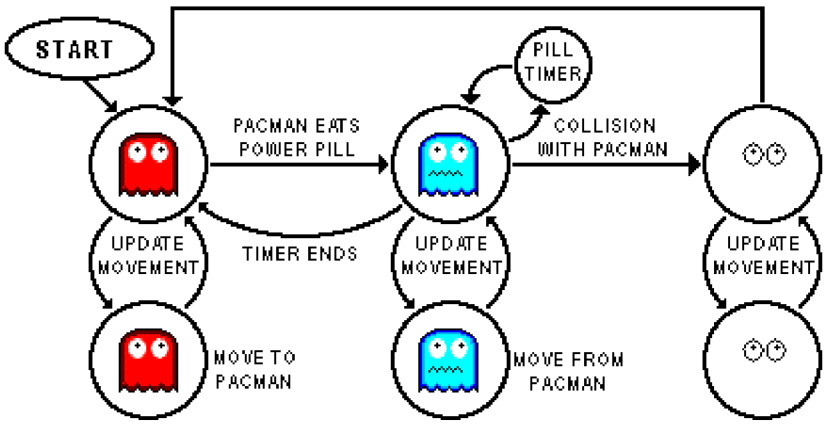
3.1 Finite state machines

Konečné automaty (nebo-li FSM) je výpočetní model definující skupinu stavů, jejich přechody a akce, které jsou potřeba pro vstup nebo výstup z těchto stavů. FSM se mohou využít jak pro hardware, tak i software. Pro herní průmysl se často využívají ve tvorbě AI, kde se vytváří různá chování a vlastnosti, kterého může nabýt.

Formálně jsou konečné automaty definované jako uspořádaná pětice:

- **S** je konečná neprázdná množina stavů
- **Σ** je konečná neprázdná množina vstupních symbolů
- **σ** je přechodová tabulka, popisující pravidla přechodů mezi stavy
- **s** je počáteční stav
- **F** je množina finálních stavů

Konečný automat může být Deterministický nebo Nedeterministický [2], kdy Nedeterministický má množinu stavů v přechodové tabulce namísto jednoho cílového stavu. Zároveň je možné, aby vstupní symbol byl prázdný (nebo-li nedefinovaný). Tento automat současně vstupuje do všech stavů najednou a přijme takový vstup, který se nachází nebo vede alespoň k jednomu z konečných stavů.



Obrázek 1 – vizualizace Finite state machines, Zdroj: http://oddwiring.com/archive/websites/mndev/MSB/GD100/fsm.htm

3.2 Behavior trees

Strom chování (BT) je typ hierarchického konečného automatu. BT popisuje různé akce jako uzly (či větve), kde se hiearchicky pohybujeme na základě podmínek, nadřazených (parent) a podřazených (child) větví. Začátek stromu se nazývá kořen (root) a představuje akci nebo cíl, který se snažíme dosáhnout. Podřazené uzly jsou různé způsoby nebo podmínky, jak cíle dosáhnout.

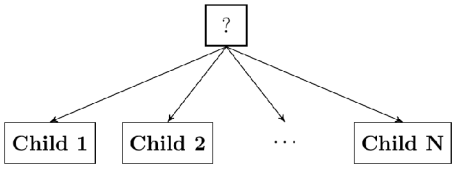
Používáme specifické typy uzlů pro tvorbu stromu chování:

3.2.1 Uzel řízení toku

Uzel řízení toku se používá pro správu uzlů jemu podřazeným. Podřazené uzly se provádějí postupně a vrací stav úspěšný nebo neúspěšný. Uzel řízení toku následně rozhoduje, zda vykoná následující podřazený uzel. Rozděluje se na Selector (Fallback), Sequence a nebo Parallel uzel.

Selector (Fallback) uzel

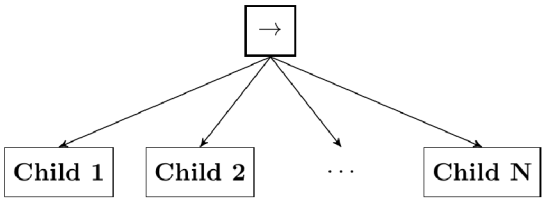
Fallback uzel vyhledává ten podřazený uzel, který neselže. Vrací stav úspěšný nebo spuštěný, pokud jeden z podřazených uzlů vrací stav úspěchu nebo spuštění, pokud všechny vrátí stav neúspěchu, tak vrací neúspěch. Podřazené uzly jsou seřazené zleva do prava podle významnosti.



Obrázek 2 – Fallback uzel, Zdroj: [3]

Sequence uzel

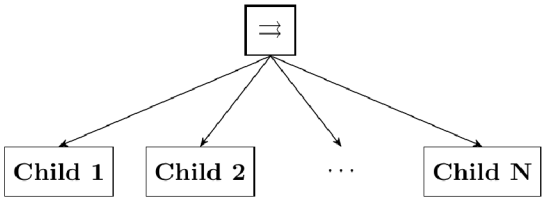
Sequence uzel vyhledává ten podřazený uzel, který ještě neuspěl. Vrací stav neúspěšný nebo spuštěný, pokud jeden z podřazených uzlů vrací stav neúspěchu nebo spuštění, pokud všechny vrátí stav úspěchu, tak vrací úspěch [3]. Podřazené uzly jsou seřazené zleva do prava podle významnosti.



Obrázek 3 – Sequence uzel, Zdroj [3]

Parallel uzel

Parallel uzel vyhledává určitý počet úspěšných uzlů, který přesahuje hranici M. Vrací stav úspěchu, pokud více než počet M uzlů vrací stav úspěch, stav neúspěchu, pokud více, než N - M (všechny podřazené uzly - hraniční počet) uzlů vrací stav, neúspěch a stav spuštění v jiném případě. Podřazené uzly jsou seřazené zleva do prava podle významnosti.



Obrázek 4 – Parallel uzel, Zdroj: [3]

3.2.2 Dekoratér

Dekoratér je uzel, který upravuje nebo opakuje výstup podřazeného uzlu, který může mít přiřazen pouze jeden. Dekoratér slouží pro rozšíření možností a nepředvídatelnosti umělé inteligence. Rozděluje se na různé typy podle jejich specifického zaměření.

Mezi typy dekorátoru patří:

- **Inverter** jednoduše převrací vrácený stav jemu podřazenému uzlu. Nejčastěji je využit pro podmínkové sekvence [4].
- **Succeeder** vždy vrací stav úspěchu nezávisle na vráceném stavu podřazeného uzlu. Využívá se pokud očekáváme vrácení chyby, ale chceme udržet proces aktivní.
- **Repeater** opakovaně vyvolává svůj podřazený uzel dokud vrací jakoukoliv odpověď. Je možné také určit počet opakování, kdy po skončení Repeater vrací stav svému nadřazenému uzlu.
- **Repeat until fail** je forma repeateru s rozdílem, že podřazený uzel je opakovaně vyvoláván dokud nevrátí neúspěšný stav. Po obdržení neúspěchu vrátí úspěšný stav svému nadřazenému uzlu.

3.2.3 Uzel akce

Uzel akce je nejnižší typ uzlu, v hierarchii stromu představuje list. Používá se pouze jako podřazený uzel a nemůže mít žádné vlastní podřazené uzly. Představuje konkrétní akci v hierarchii a nabývá stavů úspěch, neúspěch a spuštěn.

3.2.4 Uzel podmínky

Uzel podmínky je stejně jako Uzel akce nejnižší typ uzlu. Jde o podřazený uzel a také nemůže mít podřazené uzly. Představuje podmínku určitého stavu, akce nebo vstupu. Narozdíl od Uzlu akce, ale nabývá pouze stavů úspěch a neúspěch.

3.3 Decision trees

Strom rozhodnutí (DT) je model definující rozhodovací proces umělé inteligence nebo algoritmus využívaný pro machine learning [5]. Představuje jednoduchý vývojový diagram, který umožňuje rychlé a nenáročné rozhodování umělé inteligence.

Jednotlivé části stromu rozhodnutí se rozdělují na:

- **Kořen** je počáteční uzel popisující cíl nebo konkrétní akci.
- **Větev** představuje celou jednu část stromu rozhodnutí.
- **Rozdělení** představuje jednoduchou podmínku, která má převážně pravdivou a nepravdivou část.
- **Uzel rozhodnutí** je navazující část stromu, která rozděluje strom na další uzly. Odkazuje na listy a nebo další uzly rozhodnutí.
- **List** je konečná část stromu, která obsahuje odpověď, akci nebo hodnotu.

Implementace Stromu rozhodnutí pro strojové učení umožňuje růstu, kdy umělá inteligence si vytváří čím dál tím větší mapu rozhodnutí pro svůj konkrétní účel. Díky jednoduchosti tohoto modelu je poté jednoduché, aby přerostl. Pro kontrolu nečistoty stromu se používají metody Entropy a Gini, které počítají nečistotu a náhodnost uzlů a výsledků. Pokud je strom příliš velký nebo nepřesný, tak je možné využít metod Pruning nebo Boosting.

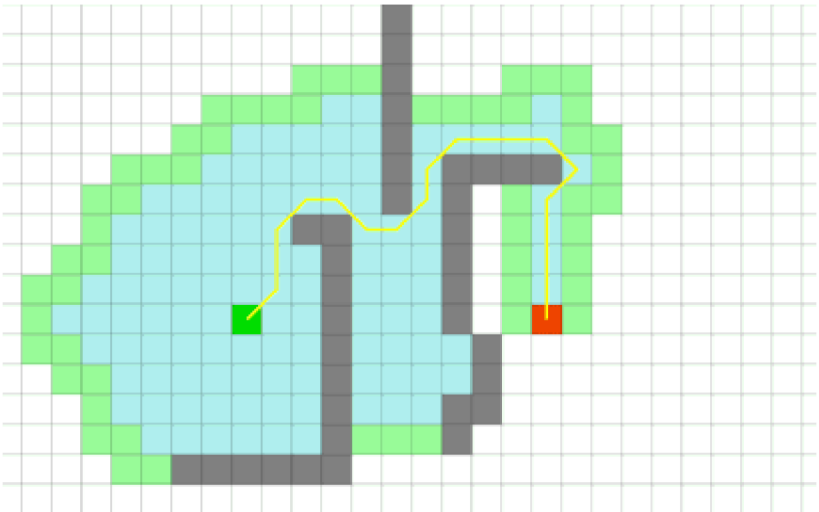
- **Pruning** je proces odstraňování nečistých nebo nevyužívaných větví stromu. Výsledkem je více soustředěný strom.
- **Boosting** je proces kombinování více nečistých nebo slabých stromů v jeden silný. Výsledkem je přesnější strom.

3.4 Pathfinding

Pathfinding je proces nalezení nejkratší cesty k cíli. Pro NPC se jedná o průchod z počátečního bodu do cílového skrz herní svět nebo plochu. Jde o nejpodstatnější vlastnost umělé inteligence ve hrách a umožňuje o navázání dalších komplexních vlastností, které mohou vytvořit více uvěřitelné chování. Proces hledání nejkratší cesty pracuje s mnoho informacemi, ale nejpodstatnější rozdělení je, zda algoritmy jsou informované nebo neinformované [6]. Tedy jestli agent má informaci o cíli (tím pádem o prostoru/mapě) nebo pouze přístup k informacím, které jsou „před ním“.

Pathfinding algoritmy se rozdělují na:

- **Neinformované algoritmy** hledají cíl v neznámém prostoru a postupně své prostředí prohledávají.
- **Informované algoritmy** hledají cíl v známém prostoru a často mají přístup k přesné poloze svého cíle. Rovnou tedy hledají optimální cestu k určenému cíli.



Obrázek 5 - vizualizace Pathfinding algoritmu, Zdroj: https://www.codingame.com/learn/pathfinding

3.4.1 Neinformované algoritmy

Neinformovaný algoritmus hledání nejkratší cesty nemá přístup k datům prostoru, ale pouze k jemu viditelnému okolí. Z tohoto důvodu si vybírá pouze viditelné body a postupuje dále podle nově dostupných bodů. Výhodou tohoto přístupu je v tom, že nevyžaduje žádné předprocesování, což přináší nenáročnost a rychlé zpracování. Jasnou nevýhodou je, že například dosažení cíle v bludišti bude trvat velmi dlouho a v některých případech to může být nemožné [6]. V kontextu s AI se tyto algoritmy mohou využít pro rychlé nastartování pohybu NPC, zatímco více sofistikovaný algoritmus vypočítá více uvěřitelnou cestu.

3.4.1.1 Breadth First Search

Prohledání do šířky (BFS), je typ neinformovaného algoritmu, který se převážně využívá pro prohledávání stromů a grafů. Způsob prohledávání je na principu hledání sousedícího uzlu od toho předchozího. Pro vytvoření BFS algoritmu se využívá datový typ FIFO (First In - First Out) pro ukládání dat a pouze využívá funkce vložení na konec fronty a odstranění ze začátku. Výhodou tohoto řešení je, že garantujeme nalezení optimálního řešení a neuvízne v alternativní špatné cestě. S tím ale přichází větší náročnost paměti a při větších prostorech i delší operační čas.

3.4.1.2 Depth First Search

Prohledání do hloubky (DFS), je typ neinformovaného algoritmu, který se převážně používá pro prohledávání grafů. Princip hledání do hloubky spočívá v prohledávání zvolené větve do jejího konce a následnému navrácení, pokud se ve větvi cíl nenachází. Pro vytvoření DFS algoritmu se využívá datový typ LIFO (Last In - First Out) pro ukládání dat a využívá funkcí pro vložení a odstranění z vrcholu zásobníku. Výhodou algoritmu je, že je méně náročný a vyžaduje menší operační čas při větších prostorech. Nevýhodou je, že negarantuje nalezení cesty a může některé uzly prohledat vícekrát.

3.4.2 Informované algoritmy

Informovaný algoritmus hledání nejkratší cesty má narozdíl od neinformovaného přístup k informacím prostoru, s kterým má pracovat. Z toho vyplývá, že vypočítává nejlepší možnou cestu ze všech možností. Výběr nejlepší možnosti spočívá v určení nejméně náročného výsledku nebo-li nalezení cesty, která má nejmenší cenu [6]. Celková cena cesty se určuje podle celkové vzdálenosti zvolené cesty. Tedy náš algoritmus se snaží najít co nejlevnější možnost. Většinou tento přístup zaručuje nalezení cíle, ale ne vždy nalezení té nejlepší možné cesty. Nevýhodou tohoto způsobu je, že vyžaduje nějaký procesní čas, který může přispět na nereálnosti AI.

Nejvíce využívané strategie pro informované algoritmy jsou:

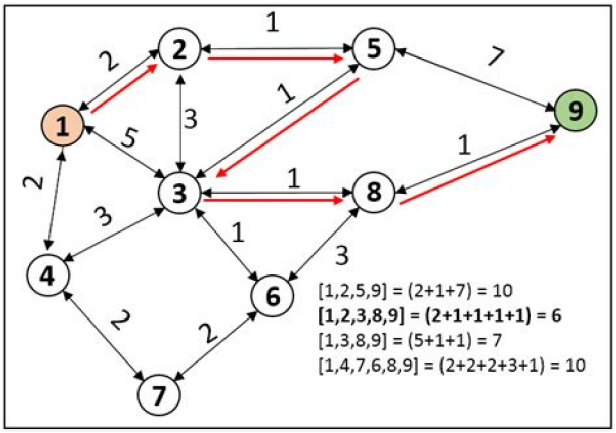
- **Stejnoměrné hledání cesty** je přístup vždy zvolení toho nejlevnějšího následujícího uzlu. Tím je zaručena optimální a kompletní nalezení té nejlevnější cesty za cenu nízké efektivity.

- **Heuristické hledání cesty** je přístup estimování ceny následujících uzlů. Tím snížíme procesní čas, ale nejde o optimální a někdy ani o kompletní nalezení cesty.

Tyto strategie umožňují pro variaci různých přístupů k hledání nejkratší cesty, kdy každý přístup využívá jinou strategii určitým způsobem. Dva nejvíce používané informované algoritmy jsou Djikstra a A* algoritmus. Djikstra využívá strategii Stejnoměrného hledání cesty, zatímco A* algoritmus využívá obou strategii. V praxi je A* algoritmus více efektivní než Djikstra a z tohoto důvodu je nejpoužívanější v herním průmyslu. Díky tomu existují také jiné formy A* algoritmu s jejich specifickým zaměřením, jako třeba IDA* a SMA*.

3.4.2.1 Djikstra algoritmus

Djikstra algoritmus je typ informovaného algoritmu, který se soustředí na strategii Stejnoměrného hledání cesty, kdy se snaží najít tu nejlevnější cestu. Tato strategie zaručuje, že vždy nalezne optimální cestu, která má nejmenší cenu k cíli, pokud cíl existuje. Způsob prohledávání cesty spočívá v tom, že se spustí cyklus s maximálním počtem iterací rovnému počtu zkoumaných uzlů a následně se při každém cyklu prověřuje právě jeden uzel a jeho cena. Djikstra algoritmus využívá podobu prioritní fronty (FIFO), kde prioritnější uzly jsou ty, které mají nejmenší cenu. Jelikož si pamatuje ceny všech uzlů, tak následně může prioritizovat právě ty nejlevnější. Právě z důvodu náročnosti výpočtu všech možností, ale vzniká velká nevýhoda dlouhého procesního času a vysoké využití procesní síly. Z těchto důvodů se využívá v herním průmyslu velmi málo, protože je vždy velké zaměření na efektivitu kódu a náročnost programu.



Obrázek 6 – Djikstra algoritmus, Zdroj: http://toughcoder.net/blog/2022/09/12/understanding-dijkstra-algorithm/

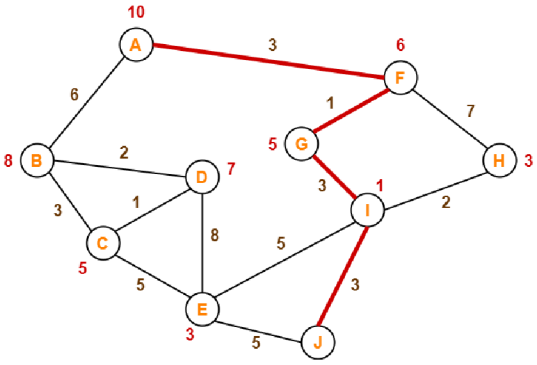
3.4.2.2 A* algoritmus

A* algoritmus je typ informovaného algoritmu, který se soustředí na strategii Heuristického hledání s využitím principů Stejnoměrného hledání. Toto zaměření bere větší ohled na rychlejší nalezení cíle za cenu méně optimalizované cesty (cesty s větší cenou). Výsledek se ale často blíží k nejvíce optimální cestě.

Heuristická vzdálenost má různá řešení:

- **Euklidovská vzdálenost** je výpočet vzdálenosti od počátečního bodu do zvoleného, kdy počítáme rozdíly souřadnic x, y počátečního a zvoleného bodu a následně je sečteme.
- **Manhattanská metrika** je výpočet vzdáleností, kde počítáme rozdíly souřadnice x, y počátečního bodu a sečteme je se stejným rozdílem zvoleného bodu

A* algoritmus iteračně prohledává ty body, které jsou pomocí předchozího výpočtu nejblíže k opravdovému cíli. Využívá formy prioritní fronty (FIFO), kde prioritnější uzly jsou ty, které mají nejmenší heuristickou vzdálenost. Nevýhodou A* algoritmu je pouze, že ne vždy dokáže najít tu nejlevnější cestu, ovšem za cenu mnohem menšího času, který je potřeba pro získání výsledku, je zanedbatelná. V určitých případech může být paměťově náročný, pokud se cíl nachází poměrně daleko.



Obrázek 7 – A algoritmus, Zdroj: https://www.gatevidyalay.com/tag/a-algorithm-example-ppt/*

3.4.2.3 IDA* algoritmus

IDA* algoritmus je typ informovaného algoritmu, který využívá principu A* s tím rozdílem, že se soustředí na omezení využití paměti. Navíc od stejného principu A* algoritmu využívá také princip DFS algoritmu a využívá prahové hodnoty, která omezuje prostor, ve kterém algoritmus prohledává v určitou iteraci. Průběh tohoto hledání funguje tak, že pokud zvolená cesta v dané iteraci přesahuje prahovou hodnotu, tak se tato prohledaná větev zahodí. Prahová hodnota představuje heuristickou vzdálenost, která se navyšuje po každém neúspěšném nalezení cíle. Nevýhodou tohoto řešení je, že při každé nově zvolené iteraci prohledáváme již dříve prohledané větve, což v určitých případech může silně navýšit procesní čas a vést k zásekům. Ovšem nižší využití paměti může být vhodnější pro hry, které se soustředí na co největší optimalizaci, třeba při vytváření co nejmenšího programu pro zařízení s nízkou pamětí, např. staré konzole.

3.4.2.4 SMA* algoritmus

SMA* algoritmus je typ informovaného algoritmu, který využívá principu A* algoritmu a snaží se omezit využití paměti. Dosahuje toho tím, že má nastavené určité omezení využitelné paměti, a pokud by přesáhl této hranice, tak spustí proces vyklízení. Průběh algoritmu je tedy normální, dokud nenarazí na limit zvolené paměti. V tuto chvíli spustí optimalizační proces, při kterém zmenšuje ty největší větve takovým způsobem, že odstraní ty nejvzdálenější listy a hodnotu heuristické vzdálenosti těchto listů připíše jim nadřazeným. Tímto způsobem má stále informaci o celkové ceně průchodu větve, ale nepotřebuje si pamatovat informaci jednotlivých uzlů. Nevýhodou tohoto řešení vzniká v případě, kdy zvolíme příliš nízkou paměť, v tu chvíli se nám navýší procesní čas, protože strávíme více času optimalizováním paměti než hledáním optimální cesty. Musíme tedy zvolit optimální velikost paměti pro cíl algoritmu. Velkou výhodou je ovšem silné optimalizování využití paměti. To může být ideální pro stejná využití jako u IDA* algoritmu.

3.5 Machine learning

Machine learning je metoda učení umělé inteligence s využitím informací nebo dat [7]. Jde o komplexní obor, který zasahuje do mnoha odvětví s rozdílnými účely. V případě využití strojového učení pro počítačové hry je nejpopulárnější trénování umělé inteligence, aby byla více inteligentní, obtížná porazit a více napodobila reálné chování hráče.

Způsobů jak dosáhnout chtěných výsledků je mnoho, ale nejpodstatnější jsou tyto hlavní přístupy:

- **Učení pod dozorem (Supervised learning)**: je metoda strojového učení, která využívá značených datasetů pro vytvoření algoritmů, které správně zpracují přijatou informaci na základě použitého zadání.
- **Učení bez dozoru (Unsupervised learning)**: je metoda strojového učení, která využívá neznačených datasetů pro vytvoření algoritmů, které se snaží nalézt souvislosti a skupiny podobných informací bez lidského zásahu.
- **Zpětnovazební učení (Reinforcement learning)**: je metoda strojového učení, která se učí za běhu podle získaného výsledku. Nevyužívá předem získaného datasetu, ale posiluje ty výsledky, které jsou nejblíže k žádaným.
- **Neuronové sítě (Neural networks)**: je podmnožina strojového učení, která se snaží napodobit funkci lidského mozku. Využívá vrstev uzlů, které se rozdělují na vstupní, skryté a výstupní. Každý uzel je spojen s následujícím uzlem a postupně zpracovává informaci.

3.5.1 Učení pod dozorem

Učení pod dozorem využívá trénovacích dat pro učení modelů, které vyhodnotí informace s žádaným výsledkem. Tento dataset obsahuje vstupní hodnoty a jejich správné zpracování [8]. Přesnost výsledného algoritmu je měřena na základě ztrátovosti, která se postupně snižuje dokud není dostatečně nízká. V herním průmyslu se učení pod dozorem využívá v mnoha aspektech. Nejpodstatnější využití je při tvorbě agentů, kteří vykonávají úlohy jako jsou pathfinding, rozhodování nebo napodobování hráčského chování. Dále se učení pod dozorem využívá pro vytváření animací, analytiku nebo anticheat (odhalení podvodného chování hráčů) při vývoji her.

Učení pod dozorem se může dále rozdělit na:

- **Klasifikační algoritmy**: soustředí se na správné rozdělení vstupních dat do korespondujících kategorií. Rozdělovací proces spočívá ve vytváření odhadů správného označení specifických entit. Klasifikované algoritmy se využívají při vytváření agentů do her.

- **Regresní algoritmy**: snaží se rozlišit vztahy mezi závislými a nezávislými hodnotami. Nejčastěji se využívají pro vytváření projekcí nebo odhadů dat pro analýzu.

3.5.1.1 Naive Bayes

Naivní Bayes je klasifikační přístup využívající Bayesovy zásady, která popisuje pravděpodobnost na základě předešlých podmínek. Nadále předpokládá, že vstupní hodnoty jsou nezávislé. I když je výpočet velmi rychlý, tak nemusí vždy být přesný. Nejčastější využití je pro algoritmy doporučení (reklama, obsah).

3.5.1.2 Support vector machines

Support vector machines je model, který využívá klasifikace i regrese dat. Princip rozdělení dat spočívá ve vytvoření hyperplochy v takové vzdálenosti, aby porovnávaná data měla co největší vzdálenost. Tato hyperplocha slouží jako rozhodující hranice pro data [9]. Výhodou je rychlost výpočtu a menší potřeba vzorků pro učení.

3.5.1.3 K-nearest neighbor

K-nearest neighbor je klasifikační algoritmus, který označuje data na základě jejich vzájemné vzdálenosti a asociaci. Využívá Eukleidovy věty pro výpočet vzdálenosti mezi daty a následovně jim přiřadí nejčastější nebo průměrnou kategorii. Princip algoritmu spočívá v předpokladu, že podobná data se vyskytují blízko sebe. Tento přístup je jednoduchý na implementaci, ale má problém s narůstajícím datasetem.

3.5.1.4 Random forest

Random forest je algoritmus, který využívá Stromu rozhodnutí (DT) [10]. Průběh výpočtu začíná náhodným rozdělením dat na skupiny, které tvoří jednotlivé DT. Následně vybere vektory dat, které projdou jednotlivými stromy a vybere se ten, který má největší četnost. Tento přístup zvládne zpracovat velká kvanta dat, ale je náročný.

3.5.1.5 Linear regression

Lineární regrese se používá pro určení vztahů mezi závislou a jednou nebo více nezávislou hodnotou. Pro každou regresi je hledané nejlepší spojení, které se vypočítává pomocí metody nejméně čtverců. Využívá se pro tvoření předpovědí pro nové výskyty.

3.5.1.6 Logistic regression

Logistická regrese se má stejný princip jako Lineární regrese, s tím rozdílem, že se soustředí na taková data, jejichž výsledek má logickou hodnotu (např. ano, ne nebo pravda). Využívá se v případech, kdy očekáváme stejný výsledek (např. spam nebo biologie).

3.5.2 Učení bez dozoru

Učení bez dozoru narozdíl od Učení pod dozorem nevyužívá trénovacích dat, ale naopak prozkoumává zvolené datasety a vyhledává podobnosti a vzory tzv. „naslepo“. Tyto souvislosti nachází bez jakékoliv lidské intervence [11], proto se tento model nazývá Učení bez dozoru. Využití je převážně pro průzkumy, analýzu, rozeznávání a rozdělování dat. Jedním z nejznámějších použití jsou algoritmy doporučení obsahu (YouTube, Google).

Jsou tři různá zaměření při využívání Učení bez dozoru:

- **Clustering (Seskupování)**

  Je způsob seskupování dat na základě jejich podobností nebo odlišností. Využívá se pro zprocesování neupravených a neoznačených dat. Mezi seskupovací algoritmy patří Exkluzivní, Překrývající, Hierarchický a Pravděpodobnostní.

- **Association rules (Pravidla asociace)**

  Je metoda hledání vztahů mezi proměnnými ve zvoleném datasetu. Využívá se např. pro analýzu prodeje produktů. Využívané algoritmy jsou Eclat, FP-Growth a Apriori, který je nejvíce rozšířený.

- **Dimensionality reduction (Redukce rozměrů)**

  Je technika pro redukci dat ve velkém datasetu s co nejmenším dopadem na výsledek. Jde o optimalizaci, která se soustředí na snížení procesního času výpočetních algoritmů. Využívá se při předzpracování dat pro jejich využití. Mezi používané metody patří Analýza hlavních komponent, Dekompozice singulární hodnoty a Autoenkóder.

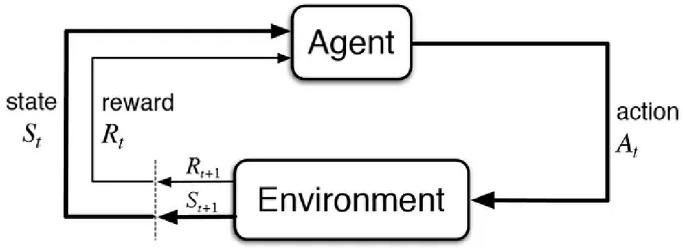
3.5.3 Zpětnovazební učení

Reinforcement learning (RL) je typ strojového učení, které využívá odměn a trestů pro označení správného nebo špatného výsledku. Narozdíl od Učení pod dozorem nemá přístup ke správnému řešení, podle kterého by určilo správný výsledek, ale prochází prostorem nebo prostředím a postupně přichází na chtěný výsledek. Cílem této metody je nalézt takové řešení, které poskytne co největší možnou odměnu [12]. Využití Zpětnovazebního je široké, ale jedním z příkladů je optimalizace robotů ve strojírenství, kde mohou roboti vzájemně sdílet své přístupy. V herním průmyslu je největší využití pro vytváření velmi obtížných oponentů (AI) nebo hledání strategií.

Pro řešení Zpětnovazebního učení se určují proměnné:

- **Prostředí** je určitý prostor ve kterém agent operuje
- **Stav** je aktuální situace agenta
- **Odměna** je zpětná vazba od prostředí
- **Strategie** je metoda rozvhrnutí agentova stavu do činů
- **Hodnota** je následující odměna, kterou agent dostane pokud odvede určitou akci v určitém stavu

Vytvoření optimální strategie přináší komplikace v podobě vyvážení hledání nových řešení a získávání maximální odměny. Tato problematika se nazývá Exploration vs Exploitation trade-off (Kompromis průzkumu a zužitkování). Z toho vyplývá, že nejlepší strategii je vyvážený přístup.



Obrázek 8 – Zpětnovazební učení, Zdroj: [12]

Při vytváření zpětnovazebních algoritmů se využívají metody:

- **Markov Decision Processes**

  Markovův rozhodovací proces (MDP) je matematická soustava pro popis prostředí v RL. Skládá se z konečného počtu stavů prostředí S a sadu možných činů A v každém stavu, skutečnou odměňovací funkci R a přechodový model P. Ovšem pro realistické scénáře je nejvíce vhodné se pohybovat v neznámém prostředí.

- **Q-learning**

  Q-learning je přístup k RL využívající hodnot Q, které pro každý čin v daném stavu určují odměnu pro následující směr stavu [13]. Princip algoritmu spočívá v zjištění možných odměn pro dané prostředí a jednotlivé fáze. Po dosáhnutí cílového stavu se proces spustí znovu, začínající v nové startovní pozici.

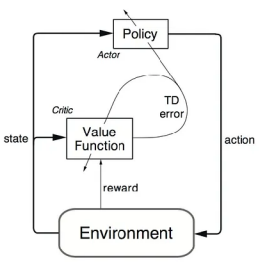
Mezi nejpoužívanější algoritmy zpětnovazebního učení patří:

- **Deep Q-Networks**

  Deep Q-Networks (DQN) je RL algoritmus, který využívá Neuronových sítí pro estimaci Q hodnot. Využívají se ovšem pouze pro nízkorozměrné prostředí.

- **Deep Deterministic Policy Gradient**

  Deep Deterministic Policy Gradient je RL algoritmus, který využívá Actor-Critic architektury. Oproti DQN si vytváří strategie ve vysokorozměrném prostředí.



Obrázek 9 – Deep Deterministic Policy Gradient, Zdroj: [12]

3.5.4 Neuronové sítě

Neuronové sítě (ANN) jsou typem strojového učení, jejichž princip je založen na neuronech lidského mozku [14]. ANN se skládají z vrstev uzlů, které se rozdělují na vstupní, skryté a výstupní. Každý uzel je spojen s jemu následujícímu uzlu, kde vytvořené spojení má určitou váhu a práh. Pokud výstup jakéhokoliv uzlu přesahuje určitý práh, tak se aktivuje a pošle data do další vrstvy sítě. Stejně jako Učení pod dozorem, jsou ANN závislé na trénovacích datech pro optimalizaci jejich přesnosti. Nejrozšířenější využití ANN jsou rozeznávání hlasu a obrázků. Nejznámějším využitím je vyhledávací algoritmus Google.

Mezi nejpoužívanější typy neuronových sítí patří:

- **Feedforward neural networks (FNN)**

  Jedná se o typický případ ANN, kdy uzly tvoří zmíněné jednotlivé vrstvy. Využívají se pro viditelnost počítače (možnost počítače předstírat vlastnost smyslů jako zrak), porozumění textů (rozeznání kontextu, zabarvení věty, …) a jako základ pro další ANN.

- **Convolutional neural networks (CNN)**

  Jde o podobný přístup jako FNN s rozdílem využití principu z lineární algebry a násobení matic. Využívají se primárně pro rozeznání obrázků, archetypů a viditelnost počítače.

- **Recurrent neural networks (RNN)**

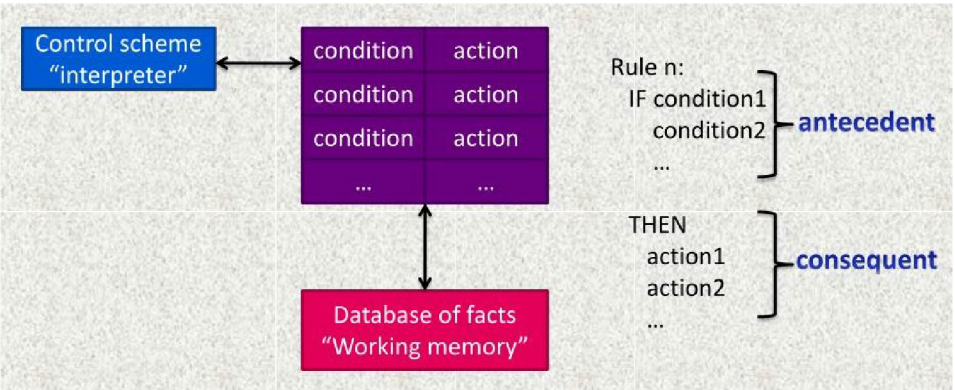
  Hlavním rozdílem RNN jsou jejich zpětnovazební smyčky, které využívají paměti pro ovlivnění výsledku na základě předchozích informací. Využívají se pro účely, kde je potřeba pracovat s časem, konkrétně předpovídání budoucích výsledků (akciový trh, slevy).

3.6 Rule-based systems

Systémy založené na pravidlech (RBS) jsou nejjednodušší formou umělé inteligence [15]. Jde o metodu vytvoření předdefinovaných pravidel pro NPC, které vytváří jeho chování. Tento přístup vytváří jasně definovaná, předvídatelná a rychlá rozhodnutí. Vytváření těchto pravidel využívá buď již zmíněných Stromů rozhodnutí (DT), nebo jednoduše if-then statement, tedy podmínek a jejich následků. Při vývoji her se často využívá jakékoliv formy RBS, kdy už pro samotné vytvoření NPC se nastavují jednoduchá pravidla, aby ve světě „žila“.

RBS se skládají z těchto základních prvků:

- **Soubor faktů** jsou určitá tvrzení, která definují počáteční stav systému.
- **Soubor pravidel** obsahuje veškeré akce, které by měly nastat v rámci obdržené problematiky. Tyto pravidla souvisejí s fakty použitými v podmínce a určité akci v následcích. Systém by měl obsahovat pouze relevantní pravidla, protože počet pravidel ovlivňuje výkonnost vytvořeného systému.
- **Ukončovací kritérium** je podmínka, která potvrzuje ukončení problematiky na základě nalezení výsledku nebo jeho nenalezení. Jde o podstatnou část systému, která zaručuje, aby nebyl spuštěn na vždy.



Obrázek 10 – Rule-based systems, Zdroj: https://www.slideserve.com/aricin/cpts-440-1324085

4 Technologie pro tvorbu pluginu do hry Minecraft

Vytváření úpravy do hry Minecraft vyžaduje primárně pouze pár věcí. První je samotná hra Minecraft, ve které se daná úprava aplikuje, zobrazí a je možné s ní interagovat. Pro použití Minecraftu a následovné vytváření úpravy je potřeba, programovací jazyk Java, alespoň verze 1.8. Dále je nutný nástroj Spigot, který umožňuje úpravu hry bez jakékoliv změny koncového uživatele. Nástroj Spigot se následně spouští jako server, ke kterému je možné se lokálně připojit pomocí hry.

Použité technologie při vytváření pluginu do hry Minecraft:

- **Minecraft** je populární hra naprogramovaná původně v jazyce Java. Její popularita a jednoduchost poskytuje ideální prostředí na zkoušení různých konceptů a nápadů.
- **Java** je jedním z nejrozšířenějších programovacích jazyků. Využívá třídy a objektově orientované programování jako hlavní koncept. Tento přístup přispívá na jednoduchosti a přívětivosti pro mnoho programátorů.
- **Spigot** je software, který rozšiřuje a upravuje základní Minecraft server. Optimalizuje mnoho aspektů hry pro lepší výkon na serveru a zároveň umožňuje programátorům jednodušší způsob jak přistoupit k základním aspektům hry a upravit je podle vlastních představ.

4.1 Minecraft

Minecraft je 3D sandboxová hra vytvořená Švédským developerem „Notch“. Nyní je spravována firmou Mojang Studio, která je součástí Xbox Game Studios, kterou vlastní z části Microsoft [16]. Původní verze hry byla vytvořena pouze pomocí jazyku Java a dostupná na PC pro operační systémy Windows, Linux a macOS. V dnešním stavu je hra napsaná v různých programovacích jazycích a dostupná téměř na všech herních platformách. Termín sandboxová hra znamená, že hráč nemá žádný určitý cíl ve hře, ale pouze mu je představen daný svět, jeho limitace a ovládání, vše ostatní už je na samotném hráči. Vytvořený svět v Minecraftu je dynamicky generovaný podle předdefinovaných pravidel s využitím pevně daných struktur a cílů. To umožňuje pro vytvoření unikátních světů, které ale vždy obsahují podobná prostředí, struktury a NPC. Hlavní půvab hry ale spočívá v její jednoduché krychlové grafice a prostoru, který umožňuje nekonečnou kreativitu.

Java Edice hry je naprogramována pomocí programovacího jazyku Java a nástroje Java Development Kit (JDK). Tento kód je následně zkompilován a zpracován pomocí Java Virtual Machine (JVM). Má vytvořené své vlastní komponenty pro správu jednotlivých částí hry, narozdíl od většiny dnešních populárních her. Komponentami je myšlen herní, grafický, zvukový a síťový engine. Tyto komponenty jsou vytvořeny pomocí knihoven Java (např. LWJGL, OpenAL) [17]. Pro spuštění hry je potřeba, aby zařízení mělo k dispozici Java Runtime Environment (JRE), to znamená nainstalovanou Javu.

4.2 Java

Java je programovací jazyk založený na konceptech tříd a objektově orientované programování za účelem omezení implementačních závislostí [18]. Hlavní slogan Javy je, „Napiš jednou, použij kdekoliv“, který v dnešní době už není úplně platný. Ovšem stále platí, že Java je jedním z nejrozšířenějších programovacích jazyků.

Hlavní pojmy při používání Javy:

- **Java Virtual Machine (JVM)** je proces, který ma na starost vykonání vygenerovaného bytového kódu, který je generován pomocí kompilátoru. Kompilátor JAVAC zpracovává vytvořený program v Javě a překládá ho na bytový kód, který umožňuje počítači vykonat příkazy definované v kódu. Kompilátor je takový překladatel mezi počítačem a programátorem.
- **Java Development Kit (JDK)** je kolekce nástrojů potřebných pro programování a spouštění programů v Javě. Obsahuje nástroje jako je JVM, JRE , kompilátor, debugger, dokumentaci a další.
- **Java Runtime Environment (JRE)** je kolekce nástrojů potřebných pro spouštění programů v Javě. Obsahuje nástroje jako JVM, prohlížeč a potřebné pluginy.
- **Garbage Collector** je proces, který obstarává využití přidělené paměti programu. Při zpracování vytvořeného programu v Javě se vytváří objekty v přidělené paměti, a Garbage Collector má na starost odstranění již nevyužívaných objektů, nebo obnovení znovu potřebných objektů.
- **ClassPath** je cesta souboru, která označuje soubor dostupný pro zpracování kompilátorem. Tyto soubory jsou označené koncovkou “.class”.

Výhody použití Javy pro vytváření her

- **Nezávislost platforem**, většina zařízení dokáže pracovat s programy vytvořenými v Javě, stačí aby měly nainstalované JRE.
- **Objektově orientované programování**, nabízí způsob organizování a následně jednodušší práci s vytvořeným kódem.
- **Bezpečnost**, při vytváření kódu v Javě není možné spustit program, který je nefunkční nebo by způsoboval chyby. Programátor je na tyto chyby vždy upozorněn.
- **Velká komunita**, Java je jeden z nejrozšířenějších programovacích jazyků a to s sebou přináší velký obsah již řešených problémů a mnoho expertů, kteří jsou ochotní začínajícím programátorům pomoct.

Nevýhody použití Javy pro vytváření her

- **Správa paměti**, Garbage collector, který má na starost správu přiřazené paměti může významně omezit výkon hry při náročných procesech.
- **Nedostatečná podpora vývoje her**, Java je primárně zaměřená na vytváření firmware nebo jednoduchých aplikací. Nástroje pro vytváření her jsou k dispozici, ale nejsou tolik využívané a následně udržované.
- **Nekompletní nezávislost platforem**, i když Javu je možné spustit na mnoho platformách, jedna z největších herních platforem, konzole (Xbox a Playstation), nepodporuje JVM [19].

4.3 Spigot

Spigot je software, který upravuje základní Minecraft server a zároveň se jedná o API, která umožňuje programátorům upravovat chování hry, které není v základu možné. Je založen na předchůdci Bukkit, který fungoval na stejném principu, ale již není udržován pro nové verze a potřeby hry [20]. Spigot je jedním z nejrozšířenějších nástrojů pro stále aktivní ekosystém serverů. Přináší optimalizace pro chod serveru a zároveň má velkou komunitu, která vytváří mnoho užitečných nástrojů potřebných pro správu herního serveru. Největší výhoda Spigotu je, že koncový uživatel nemusí stahovat žádný dodatečný obsah.

5 Vytvoření pluginu do hry Minecraft

Pro vytváření pluginů do hry Minecraft je potřeba pár nástrojů, které umožní kód vytvořit, zpracovat, exportovat jako program a následně spustit jako funkční úpravu. Prvně je potřeba nainstalovat JDK verze 1. 8, která je nezbytná pro práci s jakoukoli verzí Spigot. Následně pro spuštění testovacího serveru a zároveň jako knihovna pro vytváření pluginu je použitý Spigot verze 1. 16. 5, který musí odpovídat stejné verzi Minecraftu. Nakonec je využit Integrated Development Environment (IDE) Eclipse, který umožní použití knihovny Spigot, napsání kódu pro plugin a jeho export. Exportovaný plugin se následně spouští pomocí serveru Spigot na testovací adrese localhost a aplikuje se ve hře Minecraft, po připojení uživatele na spuštěný testovací server.

Cílem vytvořeného pluginu je vyzkoušení možností úprav ve hře Minecraft a demonstraci zjištěných poznatků ohledně umělé inteligence v počítačových hrách. Pro demonstraci umělé inteligence se bude upravovat již nadefinované NPC ve hře takovým způsobem, aby mělo nové možnosti, cíle a chování.

Téměř všechny objekty ve hře jsou definované jako Entity. Tato třída je hlavní reprezentace všech entit, které mají základní vlastnosti jako pohyb, životy apod. Dále se Entity rozděluje na mnoho podtříd, které mají své vlastní definice, ale v rámci ukázky je využita třída EntityCreature, která je v pluginu využita u konkrétních případů upravených NPC.

5.1 Plugin Goblin thief

Cílem pluginu Goblin thief je vytvoření vlastní entity, která bude rozšiřovat vlastnosti a chování EntityZombie. Entita Zombie ve hře slouží jako nepřítel vůči hráči. Objevuje se, když ve světě je noc a snaží se útočit na hráče. Úprava se bude zabývat událostmi, chování entity, pathfinding, posluchači a podmínkami.

Představa pro vytvoření pluginu Goblin Thief spočívá ve využití malé verze Zombie jako reprezentaci goblina. Následně účel goblina bude spočívat v tom, že je schopný ukrást hráči veškeré věci, pokud se objeví a bude se snažit s ukradenými věcmi utéct. Objevení goblina nastane při zničení již existujícího materiálu ve hře GoldenOre. Pokud se objeví, ukradne všechny věci z hráčského inventáře a začne se hráči vyhýbat. Hráč musí goblina porazit, aby dostal své věci zpět. Při smrti goblina ukradené věci vypadnou na zem.

Obrázek 11 – ukázka GoblinThief ve hře, Zdroj: autor

5.1.1 Hlavní třída

Při vytváření jakéhokoliv pluginu je potřeba definovat hlavní třída, která má na starost iniciaci pluginu.

```
public class Main extends JavaPlugin {
	public List<ItemStack> stolenItems = new ArrayList<>();

	@Override
	public void onEnable() {

		PluginManager pm = this.getServer().getPluginManager();
		pm.registerEvents(new BlockBreak(this), this);
		pm.registerEvents(new EntityDeath(this), this);
		pm.registerEvents(new EntityDamage(), this);
	}

	@Override
	public void onDisable() {

	}

}
```

Obrázek 12 – hlavní třída GoblinThief, Zdroj: autor

Hlavní třída obsahuje funkce onEnable() a onDisable(), které se spouští při spuštění a vypnutí pluginu, což je shodné se spuštěním a vypnutím serveru v tomto případě. Dále je zde vytvořené pole, ve kterém se uloží ukradené věci hráče. Nakonec při spuštění pluginu se inicializují posluchače, které se spustí při zničení bloku, a smrti a poškození entity.

5.1.2 Vlastní entita Goblin

```
public class Goblin extends EntityZombie{
	public Goblin(Location loc) {
		super(EntityTypes.ZOMBIE, ((CraftWorld)
loc.getWorld()).getHandle());

		this.setBaby(true);
		this.setPosition(loc.getX(), loc.getY(), loc.getZ());

		this.setCustomName(new ChatComponentText(ChatColor.GOLD + "" +
ChatColor.BOLD + "Goblin thief"));
		this.setCustomNameVisible(true);
		this.setHealth(50);

		this.goalSelector.a(0, new
PathfinderGoalAvoidTarget<EntityPlayer>(this, EntityPlayer.class, 15, 1.5D,
1.5D));
		this.goalSelector.a(1, new PathfinderGoalPanic(this, 2.0D));
		this.goalSelector.a(2, new
PathfinderGoalRandomStrollLand(this, 0.6D));
		this.goalSelector.a(3, new
PathfinderGoalRandomLookaround(this));
	}

}
```

Obrázek 13 – vlastní entita GoblinThief, Zdroj: autor

Třída Goblin vychází z již definované entity Zombie a dále pomocí konstruktoru ji rozšiřuje a upravuje. Nejdříve se iniciuje EntitaZombie v daném světě, dále se nastaví jako verze Baby a nastaví se ji pozice ve světě podle vstupních hodnot. Následně se entitě nastaví vlastní jméno, které je formátováno takovým způsobem, aby se jméno zobrazilo zlatě a tučně. Nejpodstatnější je změna chování EntityZombie takovým způsobem, aby od hráče utíkala. Zde jsou nastavené cíle pro Entitu podle priority. Nejprioritnější cíl entity je se vyhýbat hráči pomocí metody PathfinderGoalAvoidTarget, ve které je definované, kdo má utíkat, od koho má utíkat, jak daleko a jakou rychlostí. Dále má entita za úkol panikařit, náhodně se pohybovat a rozhlížet pro simulaci samostatného chování.

5.1.3 Událost rozbití bloku

```
public class BlockBreak implements Listener{
	private Main plugin;

	public BlockBreak(Main plugin) {
		this.plugin = plugin;
	}

	@EventHandler
	public void onBreak(BlockBreakEvent event) {
		if(!event.getBlock().getType().equals(Material.GOLD_ORE)) {
			return;
		}

		Random r = new Random();
		if((r.nextInt(1000 + 0) - 0) > 100) {
			return;
		}

		event.getBlock().breakNaturally();
		Goblin goblinThief = new
Goblin(event.getBlock().getLocation());
		WorldServer world = ((CraftWorld)
event.getPlayer().getWorld()).getHandle();
		world.addEntity(goblinThief);

		event.setCancelled(true);

		for (ItemStack item :
event.getPlayer().getInventory().getContents()) {
			plugin.stolenItems.add(item);
		}

		event.getPlayer().getInventory().clear();
	}

}
```

Obrázek 14 – událost rozbití bloku GoblinThief, Zdroj: autor

Třída BlockBreak implementuje posluchač, který má na starost spuštění kódu při vykonání určité akce. Je zde definovaná hlavní třída pro použití pole ukradených věcí, jelikož při této události dojde k předání informací o věcech hráče. Spravovač

událostí spouští funkci onBreak() v případě, že ve hře byl zničen jakýkoliv blok. Na začátku je kontrola, aby se kód spustil pouze, pokud rozbitý blok je typu GOLD_ORE. Následně je definovaná šance objevení goblina při rozbití bloku, která je nastavená na 10%. Po splnění počátečních podmínek je blok zničen a v jeho místě se vytvoří instance vytvořené vlastní entity GoblinThief. Tu je potřeba přidat do stávajícího světa. Nakonec je událost uzavřena, všechny hráčské věci uloženy do pole a obsah inventáře hráče je smazán.

5.1.4 Událost poškození entity

```
public class EntityDamage implements Listener {
	private ItemStack goldNugget = new ItemStack(Material.GOLD_NUGGET);

	@EventHandler
	public void onDamage(EntityDamageEvent event) {
		if (!(event.getEntity() instanceof Zombie)) {
			return;
		}
		if (event.getEntity().getCustomName() == null) {
			return;
		}
		if(!event.getEntity().getCustomName().contains("Goblin
thief")) {
			return;
		}

		Random r = new Random();
		goldNugget.setAmount(r.nextInt(5) + 1);


	event.getEntity().getWorld().dropItemNaturally(event.getEntity().get
Location(), goldNugget);
	}

}
```

Obrázek 15 – událost poškození entity GoblinThief, Zdroj: autor

Třída EntityDamage implementuje posluchač stejným způsobem jako třída BlockBreak. Je zde definovaný předmět GOLD_NUGGET, který hráč obdrží při úderu

do goblina. Stejným způsobem se kód spustí pomocí správce události, pokud jakákoliv entita obdrží poškození. Následovně pomocí podmínek kód pokračuje pouze v případě, že poškozená entita je vlastní entita GoblinThief. Nakonec se určí náhodný počet GOLD_NUGGET, které při úderu z Goblina upadnou, jakožto odměna pro hráče za jeho úsilí získat své věci zpět.

5.1.5 Událost smrti entity

```
public class EntityDeath implements Listener {

	private Main plugin;

	public EntityDeath(Main plugin) {
		this.plugin = plugin;
	}


	@EventHandler
	public void onDeath(EntityDeathEvent event) {
		if (!(event.getEntity() instanceof Zombie)) {
			return;
		}
		if (event.getEntity().getCustomName() == null) {
			return;
		}
		if(!event.getEntity().getCustomName().contains("Goblin
thief")) {
			return;
		}


		for (ItemStack item : plugin.stolenItems) {
			if (item != null) {

	event.getEntity().getWorld().dropItemNaturally(event.getEntity().get
Location(), item);
			}

		}
	}

}
```

Obrázek 16 – událost smrti GoblinThief, Zdroj: autor

Třída EntityDeath také implementuje posluchač jako třída BlockBreak. Stejně tak se zde definuje hlavní třída pro práci s polem ukradených věcí. Pomocí správce událostí se kód spustí vždy, pokud jakákoliv entita zahyne a dále pokračuje pomocí podmínek pouze v případě, že zahubená entita byla vlastní entita GoblinThief. Po poražení prchajícího goblina na jeho místě vypadnou všechny uložené předměty v poli, které byly zanechány při vyvolání události BlockBreak.

5.2 Plugin Angry cow

Cílem pluginu Angry cow je vytvoření entity, která bude přidávat, rozšiřovat vlastnosti a chování EntityCow. Entita Cow ve hře slouží jako pasivní NPC, které hráčovi poskytuje potravu a základní materiály. Objevuje se v určitých prostředích a náhodně se pohybuje. Úprava se bude zabývat událostmi, přidání vlastností, úpravou chování, pathfinding, posluchači a podmínkami.

Představa pro vytvoření pluginu Angry cow spočívá ve vytvoření pomstichtivé verze EntityCow, která může útočit na hráče. Její objevení ve světě bude po usmrcení jakékoliv entity Cow. Po objevení bude pronásledovat hráče a snažit se na něho zaútočit, dokud nezahyne.

Obrázek 17 – ukázka AngryCow ve hře, Zdroj: autor

5.2.1 Hlavní třída

```
public class Main extends JavaPlugin {
	@Override
	public void onEnable() {
		this.getServer().getPluginManager().registerEvents(new
EntityDeath(), this);
	}

	@Override
	public void onDisable() {

	}

}
```

Obrázek 18 – hlavní třída AngryCow, Zdroj: autor

Hlavní třída obsahuje funkce onEnable() a onDisable(), které se spouští při spuštění a vypnutí pluginu. Při spuštění pluginu se inicializuje posluchač události smrti entity.

5.2.2 Vlastní entita AngryCow

```
public class AngryCow extends EntityCow{

   public AngryCow(Location loc) {
            super(EntityTypes.COW, ((CraftWorld)
loc.getWorld()).getHandle());

            this.setPosition(loc.getX(), loc.getY(), loc.getZ());

            this.setCustomName(new ChatComponentText(ChatColor.RED + "" +
ChatColor.BOLD + "Angry cow"));
            this.setCustomNameVisible(true);
            this.setHealth(100);


            try {
                  registerGenericAttribute(this.getBukkitEntity(),
Attribute.GENERIC_ATTACK_DAMAGE);
       } catch (IllegalAccessException e) {
           e.printStackTrace();
       }

            this.setOnFire(5000);
   }
```

Obrázek 19 – vlastní entita AngryCow, Zdroj: autor

Třída AngryCow rozšiřuje EntityCow a pomocí konstruktoru ji přidává nové chování a vlastnosti. Nejdříve se vytvoří entita Cow v daném světě a přiřadí se ji vlastní jméno. Následovně, aby bylo možné zaútočit pomocí pasivní NPC v Minecraftu, také je potřeba přidat vlastnost poškození, která se u EntityCow nenachází. Nakonec se entita zapálí, aby byl omezený čas, při kterém se vyskytuje.

5.2.2.1 funkce initPathfinder

```
@Override
public void initPathfinder() {

	this.getAttributeMap().b().add(new
AttributeModifiable(GenericAttributes.ATTACK_DAMAGE, (a) ->
{a.setValue(10.0);}));
	this.getAttributeMap().b().add(new
AttributeModifiable(GenericAttributes.FOLLOW_RANGE, (a) ->
{a.setValue(1.0);}));


	this.goalSelector.a(0, new PathfinderGoalFloat(this));
		this.goalSelector.a(0, new PathfinderGoalMeleeAttack(this,
1.2D, true));
		this.goalSelector.a(1, new
PathfinderGoalNearestAttackableTarget<EntityHuman>(this, EntityHuman.class,
true));
	this.goalSelector.a(1, new PathfinderGoalRandomLookaround(this));
	this.goalSelector.a(2, new PathfinderGoalLookAtPlayer(this,
EntityHuman.class, 8.0F));

}
```

Obrázek 20 – funkce initPathfinder AngryCow, Zdroj: autor

Funkce initPathfinder definuje chování vlastní entity AngryCow. Nejdříve se přiřadí hodnoty nově přidaným vlastnostem poškození a dosah. Následovně se definuje možnost pohybu a útoku pomocí pathfinding algoritmu, které jsou potřebné pro navazující algoritmus útoku na nejbližší cíl, která je definován hráč. Nakonec jsou přidané algoritmy, které přidávají na iluzi samostatné NPC.

5.2.2.2 funkce registerGenericAttribute

```
static {
       try {
           attributeField = AttributeMapBase.class.getDeclaredField("b");
           attributeField.setAccessible(true);
       } catch (NoSuchFieldException e) {
           e.printStackTrace();
       }
   }
   @SuppressWarnings("unchecked")
      public void registerGenericAttribute(org.bukkit.entity.Entity
entity, Attribute attribute) throws IllegalAccessException {
       AttributeMapBase attributeMapBase =
((CraftLivingEntity)entity).getHandle().getAttributeMap();
       Map<AttributeBase, AttributeModifiable> map = (Map<AttributeBase,
AttributeModifiable>) attributeField.get(attributeMapBase);
       AttributeBase attributeBase =
CraftAttributeMap.toMinecraft(attribute);
       AttributeModifiable attributeModifiable = new
AttributeModifiable(attributeBase, AttributeModifiable::getAttribute);
       map.put(attributeBase, attributeModifiable);
   }
```

Obrázek 21 – funkce registerGenericAttribute AngryCow, Zdroj: autor

Funkce registerGenericAttribute obstarává přidávání vlastností, které nejsou definované pro danou entitu. Vytváří mapu všech stávajících definovaných vlastností a snaží se k nim přidat novou vlastnost, která je ovšem v samotné hře definovaná. Jde o velkou úpravu stavu hry, kdy pasivní NPC normálně nemohou mít vlastnost poškození, jelikož jejich cílem není nikdy útočit. Proto je potřeba odchytit jakékoliv vzniklé chyby, které mohou při takové změně nastat.

5.2.3 Událost smrti entity

```
public class EntityDeath implements Listener {
	@EventHandler
	public void onEntityDeath(EntityDeathEvent event) {

		if(!(event.getEntity() instanceof Cow)) {
			return;
		}

		Random r = new Random();
		if((r.nextInt(1000 + 0) - 0) > 50) {
			return;
		}

		AngryCow angryCow = new
AngryCow(event.getEntity().getLocation());
		WorldServer world = ((CraftWorld)
event.getEntity().getKiller().getWorld()).getHandle();
		world.addEntity(angryCow);
	}

}
```

Obrázek 22 – událost smrti AngryCow, Zdroj: autor

Třída EntityDeath implementuje posluchač, který má na starost spuštění kódu v případě, že jakákoliv entita zahyne. Následovně se kód spustí pouze, pokud zemřelá entita je instance Cow a nastane pravděpodobnost objevení 5%. V tu chvíli se vytvoří instance vlastní entity AngryCow, která se objeví na místě zemřelé entity. Vlastní entitu AngryCow je potřeba přidat do světa, ve kterém byl posluchač inicializován hráčem.

5.3 Závěr pluginů

Vytvořením vlastní entity ve světě Minecraft byly demonstrovány základní přístupy a vlastnosti NPC v počítačových hrách. NPC je definováno jeho vlastnostmi a chováním, které při jejich změně vytváří nové funkce a možnosti. Při vytváření entity byly upraveny základní vlastnosti jako počet životů a pojmenování. Přidání neexistujících vlastností přineslo určitá rizika, ale umožnilo přidání neočekávaných interakcí NPC. Následovně byla přidaná nová chování ve formě jiných pathfinding algoritmů s přiřazenými prioritami. Tento přístup vytváření cílů v určitém pořadí je forma Uzle řízení toku, konkrétně sekvenční uzel. Jednotlivé definované akce se postupně spouští pouze v případě, že ty předchozí byly splněny. Dále se využívá informovaných pathfinding algoritmů, které využívají různých informací pro vykonání jejich konkrétně definovaného cíle. Většina použitých pathfinding algoritmů používá formu jednoduchého A* algoritmu, který hledá tu nejméně časově náročnou cestu.

6 Závěr

Průzkum, který měl za úkol představit jednotlivé metody a přístupy umělé inteligence byl úspěšně demonstrován pomocí vytvořeného pluginu ve hře Minecraft. Demonstrace neobsahovala veškeré příklady všech typů umělé inteligence, ale pouze ty případy, s kterými hra už pracuje.

Začátek práce se zabýval širokým průzkumem jednotlivých metod vytváření umělé inteligence. Jednotlivé metody byly rozděleny do náležitých kategorií a vysvětleny. Dále byly hodnocené a zmíněné případné využití v herním průmyslu nebo při vývoji her.

Po průzkumu umělé inteligence byly vysvětleny a popsány technologie, které byly využity pro demonstraci získaných poznatků. Byly zhodnoceny v kontextu herního průmyslu a jejich zamýšleného využití.

Nakonec byla vytvořena demonstrace pomocí pluginu do hry Minecraft ve které bylo upraveno chování, vlastnosti a stav již existujícího NPC. Byla vysvětlena implementace jednotlivých částí programu a demonstrována. Následně byly vysvětleny použité terminologie umělé inteligence získané z průzkumové části práce.

Vytváření pluginů a upravování již existujících her může být skvělý způsob demonstrace různých konceptů. Jde o spojení známé a zábavné s neznámou a zajímavou věcí s originálním přístupem.

7 Seznam použité literatury

[1] G. N. Yannakakis a J. Togelius, *Artificial Intelligence and Games*. Cham: Springer International Publishing, 2018. doi: 10.1007/978-3-319-63519-4.

[2] T. Agarwal, „Finite State Machine (FSM) : Types, Properties, Design and Applications", *ElProCus - Electronic Projects for Engineering Students*, 23. březen 2019. https://www.elprocus.com/finite-state-machine-mealy-state-machine-and-moore-state-machine/ (viděno 28. duben 2023).

[3] „Paper Insight: Learning of Behavior Trees for Autonomous Agents". https://engineering.nordeus.com/learning-of-behavior-trees-for-autonomous-agents/ (viděno 28. duben 2023).

[4] https://www.gamedeveloper.com/author/chris-simpson, „Behavior trees for AI: How they work", *Game Developer*, 18. červenec 2014. https://www.gamedeveloper.com/programming/behavior-trees-for-ai-how-they-work (viděno 28. duben 2023).

[5] T. A. E. Team, „Decision Trees in Machine Learning (ML) with Python Tutorial", *Medium*, 2. duben 2021. https://pub.towardsai.net/decision-trees-in-machine-learning-ml-with-python-tutorial-3bfb457bce67 (viděno 28. duben 2023).

[6] Graham, Ross; McCabe, Hugh And Sheridan, Stephen, „Pathfinding in Computer Games", 2003, doi: 10.21427/D7ZQ9J.

[7] „What is Machine Learning? | IBM". https://www.ibm.com/topics/machine-learning (viděno 28. duben 2023).

[8] „What is Supervised Learning? | IBM". https://www.ibm.com/topics/supervised-learning (viděno 28. duben 2023).

[9] „Support Vector Machines (SVM) Algorithm Explained", *MonkeyLearn Blog*, 22. červen 2017. https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/ (viděno 28. duben 2023).

[10] R. Plesník, „Poker game with AI opponents", 2022.

[11] „What is Unsupervised Learning? | IBM". https://www.ibm.com/topics/unsupervised-learning (viděno 28. duben 2023).

[12] S. Bhatt, „Reinforcement Learning 101", *Medium*, 19. duben 2019. https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292 (viděno 28. duben 2023).

[13] M. T. J. P. 4 prosince 2017, „IBM Developer", *IBM Developer*. https://developer.ibm.com/articles/cc-models-machine-learning/developer.ibm.com/articles/cc-models-machine-learning (viděno 28. duben 2023).

[14] „What are Neural Networks? | IBM". https://www.ibm.com/topics/neural-networks (viděno 28. duben 2023).

[15] C. Grosan a A. Abraham, „Rule-Based Expert Systems", in *Intelligent Systems: A Modern Approach*, C. Grosan a A. Abraham, Ed., in Intelligent Systems Reference Library. Berlin, Heidelberg: Springer, 2011, s. 149–185. doi: 10.1007/978-3-642-21004-4_7.

[16] „Minecraft", *Minecraft Wiki*. https://minecraft.fandom.com/wiki/Minecraft

(viděno 27. duben 2023).
[17] „Java Edition", *Minecraft Wiki*. https://minecraft.fandom.com/wiki/Java_Edition (viděno 27. duben 2023).
[18] „Introduction to Java", *GeeksforGeeks*, 7. listopad 2020. https://www.geeksforgeeks.org/introduction-to-java/ (viděno 27. duben 2023).
[19] „Is Java Good For Game Development? – Explain Game Dev", 15. červenec 2022. https://explaingamedev.com/is-java-good-for-game-development/ (viděno 27. duben 2023).
[20] „Frequently Asked Questions | SpigotMC - High Performance Minecraft". https://www.spigotmc.org/wiki/faq/#what-is-spigot (viděno 27. duben 2023).
[21] A. Hunt, *Learn to Program with Minecraft Plugins: Create Flying Creepers and Flaming Cows in Java*, 1st edition. Dallas, TX: Pragmatic Bookshelf, 2014.
[22] C. Xu, *Learning Java with Games*. Cham: Springer International Publishing, 2018. doi: 10.1007/978-3-319-72886-5.

8 Seznam obrázků

9 Seznam zkratek

10 Zadání

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor:	**Michal Horák**
Studium:	I1900190
Studijní program:	B1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název bakalářské práce:	**Umělá inteligence v počítačových hrách**
Název bakalářské práce AJ:	Artificial intelligence in computer games

Cíl, metody, literatura, předpoklady:

Cílem bakalářské práce je seznámit čtenáře s problematikou umělé inteligence v počítačových hrách. Dále pak pomocí vybraných technologií vytvořit plugin do již existující hry Minecraft, na kterém budou demonstrovány získané poznatky z teoretické části práce.

Osnova:

- Úvod
- Umělá inteligence v počítačových hrách
- Technologie pro tvorbu pluginu do hry Minecraft
- Vytvoření pluginu do hry Minecraft
- Shrnutí, výsledky, závěr, literatura

- Artificial Intelligence and Games, Georgios N. Yannakakis, Julian Togelius
- Learning Java with Games, Chong-wei Xu
- Learn to Program with Minecraft Plugins (Create Flying Creepers and Flaming Cows in Java), Andrew Hunt

Zadávající pracoviště:	Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu
Vedoucí práce:	Ing. Jakub Beneš
Datum zadání závěrečné práce:	26.1.2021