UNIVERZITA PARDUBICE

Fakulta elektrotechniky a informatiky

IMPLEMENTACE AI MODELU PRO ZÁVODNÍ POČÍTAČOVOU HRU

Michal Bielák

Bakalářská práce

2024

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Michal Bielák**
Osobní číslo:	**I19066**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Téma práce:	**Implementace AI modelu pro závodní počítačovou hru**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem bakalářské práce je navržení závodní hry a její zpracování v herním enginu Godot. Hra bude nabízet několik předdefinovaných okruhů, uživatelský editor okruhů a vlastní závody. Hra bude zpracována ve 2D prostoru s pohledem shora. Jako protivníci pro závody bude navržena a zpracována vlastní implementace AI.
AI bude zajišťovat pohyb vlastní pohyb po trati, předcházet kolizím s překážkami a závodníky na trati a snažit se o bezproblémové projetí trati. Pro úkony bude využita metoda ray-castingu. AI bude se hrou komunikovat pomocí protokolu WebSocket. Za správný výsledek bude považována AI zajišťující hráči kompetitivní závod.

Rozsah pracovní zprávy: **min. 30 stran**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

YANNAKAKIS, Georgios N a Julian TOGELIUS. Artificial Intelligence and Games. Cham: Springer International Publishing, 2018. ISBN 9783319635187. Dostupné z: doi:10.1007/978-3-319-63519-4

BERNHAUPT, Regina. Game User Experience Evaluation. Cham: Springer International Publishing, 2015. ISBN 3319159844. Dostupné z: doi:10.1007/978-3-319-15985-0

KUBAT, Miroslav. An Introduction to Machine Learning. 2nd ed. 2017. Cham: Springer International Publishing, 2017. ISBN 9783319639123. Dostupné z: doi:10.1007/978-3-319-63913-0

Vedoucí bakalářské práce: **Ing. Jan Merta, Ph.D.**
Katedra softwarových technologií

Datum zadání bakalářské práce: **16. prosince 2022**
Termín odevzdání bakalářské práce: **12. května 2023**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

Ing. Jan Panuš, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 28. února 2023

Prohlašuji:

Práci s názvem Implementace AI modelu pro závodní počítačovou hru jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 9. 5. 2024

Michal Bielák

PODĚKOVÁNÍ

Rád bych touto cestou poděkoval vedoucímu práce, panu Ing. Janu Mertovi, Ph.D., za odborné vedení, rady, dozor a časovou flexibilitu při konzultacích při psaní této práce. Dále děkuji také své rodině za trpělivost a podporu při studiu a práci na této bakalářské práci.

ANOTACE

Bakalářská práce se zabývá tvorbou závodní počítačové hry spolu s klientem, umožňujícím ovládání této hry. Práce popisuje výběr technologií pro vývoj, rozebírá jednotlivé typy umělé inteligence, jejich vývoj, rozšíření a dělení. Dále se práce věnuje popisu vývoje samotné hry a klienta, kde jsou tyto části rozděleny do jednotlivých fází, dle problémů, které jsou v jednotlivých fázích vývoje řešeny.

KLÍČOVÁ SLOVA

Umělá inteligence, Godot, GDScript, herní engine, skriptování, Python

TITLE

AI model implementation for racing videogame

ANNOTATION

The bachelor's thesis deals with the creation of a racing videogame together with the AI client, enabling the control of this game. The thesis describes the selection of technologies for development, analyzes individual types of artificial intelligence, their development and division according to different criteria. Furthermore, the work is dedicated to the development description of the game itself and the client, where these parts are divided into individual phases, according to the problems that the individual parts focus on.

KEYWORDS

Artificial intelligence, Godot, GDScript, game engine, scripting, Python

OBSAH

SEZNAM ZKRATEK

NN / ANN – Neural netowork / Artificial neural network

RL – Reinforcement learning

AI – Artificial intelligence – Umělá inteligence

DQN – Deep Q network – hluboká neuronová síť s podporou Q-Leariningu

MSE – Mean square error – střední křížová chyba

LTS – Long term support

SEZNAM OBRÁZKŮ

1 ÚVOD

Cílem této bakalářské práce je ověření aplikace externího klienta pro závodní počítačovou hru, umožňujícího automatizované řízení auta jezdícího po předdefinované trati, spolu se samotnou videohrou. V dnešním světě jsou autonomní technologie zahrnuty ve všech oblastech našeho života, a to včetně dopravy. Autonomní řízení dopravních prostředků se ukazuje být stále důležitějším tématem, proto je problematika řešená v práci zajímavá nejen pro hráče, ale má relevanci v reálném světě, zejména v oblasti vývoje autonomního řízení osobních automobilů. Je vhodné této problematice porozumět a řešit ji, alespoň se virtuálním prostředí her.

Teoretická část se na začátku věnuje stručnému popisu videoher a historii jejich vývoje. Je také věnována podrobnějšímu popisu umělé inteligence, rozebírá její různé typy, algoritmy a oblasti jejího využití, spolu s vysvětlením pojmů s umělou inteligencí souvisejících. Jsou zahrnuty také kapitoly o souvislosti umělé inteligence s videohrami a srovnání aktuálních možností vývoje.

Praktická část je zaměřena na dokumentaci vývoje závodní videohry v herním enginu Godot, od úvodní analýzy, přes implementaci a postup nalezení řešení jednotlivých problémů, po zhodnocení zvoleného postupu práce.

Dále je praktická část zaměřena na návrh a vývoj externího klienta, umožňujícího sběr dat a ovládání hry bez nutnosti uživatelského zásahu. Od návrhu a základní implementace přes testování k optimalizaci. Každá fáze představuje klíčový krok k dosažení cíle: vytvoření efektivního a inteligentního klienta schopného úspěšného zvládnutí projetí závodní dráhy.

2 VÝVOJ VIDEOHER

2.1 Historie videoher

V posledních letech se s videohrami setkáváme stále častěji. Dá se říct, že poslední 2–3 generace na videohrách vyrostly, ale videohry jsou starší, než si většina lidí myslí. Jejich existence se datuje už do 40. let minulého století. Vývoj počítačových her můžeme dělit do generací.

První počítačové hry se datují před polovinu minulého století, kde se za první „počítačovou“ hru dá považovat hra Nim, běžící na elektro-mechanickém zařízení zvaném Nimatron.

Za následující generaci her lze považovat hry jako je OXO a Tennis for Two. Kde první zmíněná hra využívala CRT obrazovku, původně sloužící ke zobrazení stavu paměti pro zobrazení koleček a křížků, jako ve známé hře piškvorky, a klasického vytáčecího číselníku pro uživatelský vstup. Druhá zmíněná hra byla zajímavá využitím osciloskopu pro zobrazení letícího míče. Obě hry ale často nebývají označovány za skutečné počítačové hry.

Další počítačovou hrou je hra Spacewar! vytvořena v roce 1962. Bývá označována za první skutečnou počítačovou hru. Jednalo se o vesmírnou střílečku, jejíž základní princip byl a je opakován při vývoji her velmi často.

Důležitým milníkem je také rok 1968, ve kterém společnost Magnavox začala vyvíjet první domácí herní konzoli – Odyssey. Konzole byla vydána v roce 1972.

Po tomto milníku nastoupily na trh arkádové hry, v tomto případě myšleno hry pevně spjaté s videoherními automaty. Tento typ her postupně ztrácel na popularitě, protože jejich domácí používání bylo dost obtížné. Arkády byly nedlouho poté vystřídány populárními herními konzolemi, které se v jisté míře udržely dodnes.

Okolo roku 1992 s příchodem her jako byl Unreal nebo Doom začaly trh ovládat počítače a herní konzole a domácí hraní se stalo populárním.

Zdroje[1], [2]

2.2 Herní platformy

Jak je zmíněno v předchozí kapitole, platformy pro hraní počítačových v minulosti nabíraly nebo ztrácely na popularitě. Dnes se mezi nejpopulárnější řadí mobilní telefony a tablety, jejichž si vývoj prošel podobným vývojem jako u her na PC, domácí i přenosné herní konzole a počítače.

2.3 Trendy a budoucí vývoj

Trendem ve videoherním průmyslu je dnes virtuální realita, online hry a mobilní hraní. Dá se tedy předpokládat další vývoj ubírající se právě směrem virtuální a rozšířené reality. Co se vývoje herních platforem týká, zdá se, že s rostoucí rychlostí internetu, se bude více směřovat k online streamovacím službám, a hraní bude umožněno odkudkoli.

3 HERNÍ ENGINE

Termín herní engine (česky někdy také herní motor) vznikl v 90. letech díky FPS hrám (first-person shooter – česky střílečka z pohledu první osoby), především dík hře Doom studia *id Software* Hra Doom byla první hrou s architekturou dobře odlišující funkční komponenty (vykreslování grafiky, kolize atd.) a herní svět (grafiku, herní světy, a herní pravidla). Tento způsob návrhu hry umožňoval dalším vývojářům hru upravovat (módovat) a dokonce na jejím základu i vydávat další hry. Hra Doom se tak vlastně stala prvním herním enginem.

Herní engine je tedy základ, na kterém hra stojí. Co všechno musí herní engine zajišťovat, aby byl považován za herní engine, není přesně zavedeno, obvykle ale zajišťuje tvorbu grafického rozhraní, správu fyziky ve hře, kolize, zvuky, síťové rozhraní, a také správu prostředků. Nabízí vývojáři nástroje a funkce pro zjednodušení práce a tím umožnuje se více soustředit na řešení logiky a dalších problémů. Tato kapitola se dále zaměřuje na podrobnější představení 2 herních enginů, Unity3D a Godot.

Zdroje **Error! Reference source not found.**[4]

3.1 Unity (3D)

Jedná se o herní engine vzniklý v roce 2005, s cílem poskytnout vývojářům ucelený přístup k nástrojům pro vývoj her. Nabízí multiplatformní vývoj her s možností skriptování v jazyce C#. I přes svůj název Unity3D není unity jen enginem pro tvorbu 3D her. Nabízí nástroje pro tvorbu her ve 2D a 3D prostoru i virtuální a rozšířenou realitu. Z Unity je možné exportovat na všechny rozšířené platformy, z nichž mezi hlavní patří PC, herní konzole (PlayStation, Xbox, Nintendo) a v neposlední řadě mobilní zařízení s OS Android a iOS.

Zdroje [3], [4]

3.2 Godot

Godot je open-source herní engine vyvíjený komunitou. Nabízí možnosti práce se 2D a 3D grafikou. Nabízí export pro desktopové platformy, Android, iOS a webový export do HTML5.

Jednou z hlavních výhod enginu Godot jsou rozšířené možnosti skriptování. Základním skriptovacím jazykem pro engine je proprietární skriptovací jazy GDScript, vyvinutý přímo pro Godot. Tento jazyk je (přes svůj základ v C++) syntaxí velmi podobný Pythonu. Výhodou skriptování v Godotu je i možnost skriptování ve Visual Scriptu, díky kterému je možné tvořit své první hry bez znalosti programování. Godot však podporuje i jazyky C# a C++, pro které

je nutné mít speciální verzi editoru – Godot Mono. Dále je možné pro Godot psát i v jiných jazycích, pro ty však neexistuje oficiální podpora (jsou vyvíjeny komunitou), a nelze tak zaručit jejich kompatibilita do budoucna. Jako ostatní herní enginy má i Godot vlastní nástroje pro správu fyziky, animací, zvuků, a ostatních nástrojů pro zjednodušení práce. Zajímavostí u enginu je editor, který využívá vlastní „vykreslovací systém“, ve kterém je sám napsán. Godot editor je tedy produktem vývoje v Godot Enginu. Editor je možné spustit mimo PC platformy i na platformu Android, pro kterou byl v roce 2023 vydán editor oficiálně na obchod Play.

Zdroje [8]

3.3 Srovnání a volba enginu pro projekt

Z popisu obou herních enginů vyplývá, že ani jeden z enginů se nedá označit za lepší. Volba je opravdu spíše osobní preferencí.

Za hlavní výhody Unity se dá považovat především rozsáhlejší uživatelská komunita, ze které plyne lepší podpora, větší počet kvalitních rozšiřujících knihoven a větší množství tutoriálů a jiných materiálů pro jednodušší začátek s enginem. Vzhledem ke svému dlouholetému rozšíření také podporuje více platforem.

Godot, jakožto engine vyvíjený komunitou až od roku 2014 nemá uživatelskou komunitu na úrovní větších herních enginů, jakým je i například Unity. Postupně však roste jeho popularita, a to především mezi vývojáři indie her. S rostoucí komunitou tedy roste i množství dostupných tutoriálů a dalších materiálů, které jsou i v době psaní práce velmi omezené. Hlavním zdrojem informací pro vývoj je především oficiální dokumentace k enginu. Naopak za výhodu lze považovat diverzitu skriptovacích jazyků, které tvorbu her zpřístupní i začátečníkům, a v neposlední řadě také šíření pod MIT licencí.

Co se týká stability a výkonu obou enginů, jsou oba enginy srovnatelné. Dá se však říct, že při nalezení chyby je v Godotu chyba opravena rychleji, protože komunita reaguje v podstatě okamžitě, a navíc vývojáři nic nebrání si nalezenou chybu opravit sám.

Herní engine Godot byl pro využití v projektu zvolen z důvodu open-source licence spolu s aktivní vývojářskou komunitou a rychle se rozvíjejících funkcionalit.

4 UMĚLÁ INTELIGENCE

Umělá inteligence (v práci dále referování jako AI – Artificial intelligence), je vědní obor zabývající se vývojem a tvorbou algoritmů, programů a systému, které umožňují počítačům vykonávat úkoly, na které by bylo třeba lidské inteligence. Mezi tyto úkoly je zahrnuto rozpoznávání vzorů, porozumění mluvenému nebo psanému jazyku, plánování a učení. Hlavním cílem této vědecké disciplíny je pochopení a napodobení principů lidského a zvířecího chování.

```
(write-line "single quote used, it inhibits evaluation")
(write '(* 2 3))
(write-line " ")
(write-line "single quote not used, so expression evaluated")
(write (* 2 3))
```

Obrázek 1: Ukázka kódu programovacího jazyka LISP

Za otce tohoto termínu je považován americký informatik John McCarthy, který ji definoval již v roce 1955 při tvoření programovacího jazyka LISP, jehož ukázku můžeme vidět na obrázku 1.

Umělá inteligence lze rozdělit na několik skupil a podskupin, v této kapitole jsou rozebrány vybrané typy umělé inteligence a vlastnosti, dle kterých lze AI dělit.

Zdroje **Error! Reference source not found.**[7]

4.1 Expertní systémy (Rule-based AI)

Expertní systémy, anglicky též zvané Rule-based AI, jsou nejjednodušší formou umělé inteligence. Používají právě pravidla, podle kterých mají pevně nastavené znalosti. Z názvu vypovídá, že se jedná o systémy řešící problémy, kde je třeba expertních znalostí.

Jedná se vlastně o složité podmínky, ve kterých není třeba počítačem vytvořené inteligence. Staticky je nastaveno, co je pravda a co se má na základě splnění podmínky udělat. V případě expertních systémů je tedy na subjektivním zhodnocení každého, zda bude tyto algoritmy za umělou inteligenci považovat.

Zdroje [9], [10]

4.2 Strojové učení

Strojové učení je s časem stále více se rozvíjející odvětví, zabývající se tvorbou a využíváním výpočetních algoritmů, které se dokážou učit na základě odezvy z prostředí,

ve kterém pracují. Strojové učení je možné aplikovat na snad všechny odvětví, které si umíme představit. Termín „strojové učení“ tedy neoznačuje žádný konkrétní algoritmus, ale jedná se o zastřešující termín pro všechny algoritmy přizpůsobující se na základě vstupů.

Z nejznámějších stojí za zmínku rozpoznávání obličeje a hlasů, doporučovací systémy (reklamy), prediktivní psaní (na telefonu) automobily bez řidičů nebo systémy řízení dopravy.

Strojové učení můžeme dělit dle několika kritérií. Kritéria, podle kterých a jak dělit machine learning jsou ale v různých zdrojích dost nekonzistentní, v této práci je zvoleno rozlišení dle následujících kategorií:

- Typ učení
- Algoritmus
- Model
- Optimalizační metody

Dalšímu rozlišení není v této práci věnována pozornost. Často se totiž kategorie překrývají, kdy například i v probraných možnostech je možné v některých zdrojích narazit na informaci, že rozhodovací strom je modelem i algoritmem strojového učení. V této práci je na rozhodovací strom referováno jako na model. Další rozlišení je možné například podle oblasti aplikace nebo techniky učení.

Zdroje **Error! Reference source not found.**[11]

4.3 Dělení dle typu učení

Dělení dle typu učení rozlišuje strojové učení dle způsobu, jakým se model učí a aktualizuje parametry. Jedná se o nejzákladnější dělení strojového učení, na 4 typy.

Učení s učitelem (Supervised learning)

Jedná se o nejčastější typ strojového učení. Modelu jsou poskytnuta tréninková (označená/popsaná) data v párech vstup – odpovídající výstup. Model se právě na těchto označených datech naučí vzorce, které následně využívá na datech neoznačených pro odhadnutí výsledků. Za „učitele“ je považován právě seznam kombinací vstupů a odpovídajících výstupů dat.

Příkladem učení s učitelem mohou být:

- obrázky čísel s výslednými čísly (viz obrázek 2)
- seznam vybavení domu a odhadovaná cena domu
- obrázky zvířat a určené zvíře

Zdroje [12], [13]

Obrázek 2 – Příklad označených dat

Učení bez učitele (Unsupervised learning)

Úkolem tohoto typu strojového učení je učit se na datech bez předem určených výstupních hodnot, absence těchto hodnot však výrazně zesložiťuje práci modelu. Úkolem modelu je najít v poskytnutých datech podobnost. Výstupem modelu s učením bez učitele jsou tedy shluky podobných dat, nebo odlišení výrazně vybočujících vstupů. Příkladem učení bez učitele je například detekce podvodů nebo segmentace zákazníků e-shopu.

Zdroje [12], [13], [15]

Učení s částečným učitelem (Semi supervised learning)

Jedná se o kombinaci obou již zmíněných typů učení, tedy učení s učitelem a učení bez učitele. Model využívá označená í neoznačená data, snaží se v nich hledat podobnosti a na základě nalezených podobností předvídá výsledné hodnoty. V praxi je učení bez učitele často implementováno dvěma způsoby.

V případě prvního způsobu, zvaného „sebetrénování“, jsou modelu poskytnuta data, z nichž jen malá část jsou data označená. Na malé části označených dat (např. obrázky koček a psů) naučí základní rozpoznávání. Poté jsou modelu poskytnuta data neoznačená, a model dle vzoru určí výsledný výstup (obrázek psa nebo kočky) a pravděpodobnost, s jakou je tento výstup správný. Data, u kterých existuje nejvyšší pravděpodobnost správnosti jsou označena za správná a označená a jsou poskytnuta modelu pro další učení. Tento postup je prováděn vícekrát.

U druhého způsobu (zvaného „co-training“), jsou vytvořeny dva modely (klasifikátory), oba trénovány na stejné skupině označených dat, na základě jiných parametrů. Oba modely jsou následně využity k predikci neoznačených dat. Následně jsou oba modely využity na neoznačených datech naráz, kde se v případě shody data označí za validní a přidá je k označeným datům.

Příkladem použití toho typu trénování může být klasifikace textu, kde máme například články týkající se různých témat, kde jeden klasifikátor bude řadit rada dle frekvence obsažených slov (slovní pytel), a druhý na základně posloupnosti slov (n-gram). Oba modely následně předpoví zařazená neoznačených dat, a v případě shody jsou data označena za validní a použita k dalšímu trénování modelu.

Zdroje [21], [12], [20]

Zpětnovazební učení (Reinforcement learning)

Zpětnovazebné učení je oproti dalším typům učení výrazně odlišné. Modelu nejsou poskytnuta označená data, ale jeho učení probíhá na základě okamžité zpětné vazby od okolního prostředí. Model využívá agenta, který automaticky prostředí zkoumá a na základě odezvy od prostředí (odměny nebo penalizace) upravuje svou další reakci na podobnou akci. Cílem agenta je potom maximalizovat odměnu, kterou získá za akci pro aktuální stav prostředí.

Zpětnovazebné učení se nejčastěji využívá v „automatickém chování“, ať už programů, robotů (učení se chodit), řídících systémů (energetika, dopravní systémy), nebo videoher.

Zdroje [15][19]

4.4 Dělení dle modelu

Dělení dle modelu znamená dělení dle reprezentace a zpracování dat v rámci určitého algoritmu. Ve strojovém učení to znamená rozdíl ve způsobu reprezentace a zpracování dat. Může se zdát, že se dělení dle modelu překrývá s dělením dle algoritmu, neboť některé modely jsou specifické pro algoritmus, která je používá. Modely bývají specifické pro jednotlivé typy učení.

Prvním zmíněným typem modelu je regrese, z nichž nejjednodušší je **lineární regrese**, používající se pro **učení s učitelem**. Výsledkem lineární regrese je lineární rovnice, u které je snaha o co nejpřesnější reprezentaci rozmístění dat. Používá se například pro předpověď prodejů nebo obchodováním na akciovém trhu.

Dalším modelem pro **učení s učitelem**, je **rozhodovací strom**, využívající ke svému fungování klasifikaci a regresi. Rozhodovací stromy tedy fungují podobně jako expertní systémy, s rozdílem tvorby rozhodovacích pravidel. V expertních systémech se jedná o pravidle pevně napsaná lidskými experty, rozhodovací stromy si však díky využití učení s učitelem tyto podmínky dokáží tvořit sami.

Pro **učení bez učitele** jsou specifické modely, jejichž výsledkem je **shlukování** a **snížení dimenzionality**. Shlukování dat se používá pro detekci anomálií, třídění obrázků nebo dokumentů nebo pro doporučovací systémy. Snížení dimenzionality je používáno například pro zpracování jazyka, kde může být použito pro selekci nebo odstranění specifických frekvencí.

V algoritmech **zpětnovazebního učení** jsou nejčastěji používány algoritmy **Q-učení**, **SARSA**, nebo hluboké neuronové sítě s podporou Q učení (**DQN**). Tyto algoritmy jsou vzhledem k tématu práce více probrány v následujících kapitolách.

Zdroje [16][17][18][19][28]

4.5 Dělení dle algoritmu

Strojové učení lze dělit také dle algoritmu učení. Za zmínku stojí především algoritmus **lineární regrese**, často využívaný algoritmy **učení s učitelem** pro svou jednoduchost. Hledá se v něm přímka nejlépe odpovídající vzorku označených dat. Dále je dobré zmínit algoritmy **podpůrných vektorových strojů** (support vector machines) a **rozhodovací strom**, který je pevně spjatý s modelem.

Zdroje [20][22][25]

Z kategorie **učení bez učitele** je častým algoritmem **K-Means,** využívaný pro nejlepší možné rozřazení bodů do skupin – shlukování. Jedná se o iterativní algoritmus, jehož výsledkem jsou data rozřazená do skupin, a to tak, že každé datum patří právě do jedné skupiny dat.

Zdroje [21]

5 ZPĚTNOVAZEBNÍ UČENÍ

Zpětnovazební učení je vedle učení s učitelem a učení bez učitele jedním z hlavních typů strojového učení. Princip zpětnovazebního učení spočívá v nacházení optimální strategie metodou pokus-omyl, kdy jsou agentovi poskytnuta data o prostředí, na jejichž základě vybírá akci z předem definované množiny stavů. Po provedené akci je agentova akce ohodnocena odměnou ovlivňující změnu jedné z hodnot, na základě kterých jsou následně vybírány další akce.

Zdroje [24]

5.1 Základní principy zpětnovazebního učení

5.1.1 Agent a prostředí

Jako agent je v kontextu zpětnovazebního učení označen prostředník, který pozoruje prostředí a vykonává v něm akce, zatímco za prostředí se považuje kontext, ve kterém agent provádí operace. Agent provede v prostředí akci, na jejímž základě se prostředí změní do nového stavu a agent obdrží odměnu nebo trest za svou akci. Tento cyklus je následně opakován, dokud není dosaženo nějakého cíle nebo není splněn určitý počet kroků.

Jako příklady párů agent-prostředí lze uvést robotické rameno jako agenta, a prostor a objekty, se kterými operuje jako prostředí, autonomní vozidlo a reálný svět z oblasti dopravy, program překládající text a textová data z oblasti zpracování jazyka, nebo umělou inteligenci ovládající počítačovou hru z oblasti videoherního průmyslu.

5.1.2 Prostředí a jeho stavy

Za prostředí se považuje kontext, ve kterém agent pracuje, toto prostředí může nabývat různých stavů A to jak diskrétních (tedy konkrétně definované) nebo spojitých. Proces změny stavů prostředí je základním kamenem zpětnovazebního učení.

5.1.3 Akce a politika

Akce je krok, který agent provádí na základě stavů prostředí. Jako politika se označuje strategie využívaná k nalezení optimální akce agenta pro prostředí v aktuálně dosaženém stavu. Vývoj a optimalizace politiky je klíčovým aspektem zpětnovazebního učení. Politiky je možné mezi sebou různě kombinovat, upravovat a nastavovat.

Jako příklad politik je možné uvést deterministický přístup, kde jsou pro pevně daný stav prostředí pevně dané akce (robotický vysavač narazí na překážku, a tak vždy zatočí vlevo).

Tento přístup se ve zpětnovazebním učení využívá jako základ pro další úpravy chování pomocí ostatních politik.

Ve zpětnovazebním učení je obvyklé zohlednit také poměr využívání známých dat ku průzkumu prostředí a nových akcí. K tomu využíváme pravděpodobnost náhodné akce, ta může být absolutní, jako případě plně náhodné politiky, nebo se může měnit, jako v případě použití epsilon-greedy politiky, při jejímž použití se pravděpodobnost náhodné akce postupně snižuje.

5.1.4 Systém odměn

Ve zpětnovazebním učení se za odměny (nebo tresty) považují hodnoty, poskytnuté prostředím jako zpětné zhodnocení provedených akcí. Na základě odměn jsou aktualizovány pravděpodobnosti akcí pro další stavy.

5.2 Algoritmy zpětnovazebního učení

Zpětnou vazbu z prostředí lze ve zpětnovazebním učení používat mnoha způsoby, vznikají tak různé algoritmy pro úpravu hodnot v modelu. Algoritmy lze dělit dle několika kritérií, která je možné spolu různě kombinovat.

Dělit lze například dle způsobu získání dat pro trénování modelu. Vzorová dat lze získat buď stejnou politikou, jaká je využita pro nasazení modelu (*On-policy* metoda), nebo je získat s využitím jiného algoritmu, případně je uměle generovat (*Off-policy*).

On-policy a Off-policy může značit také způsob aktualizace dat pro model, kde On-policy znamená získání dat pro aktualizace stejným způsobem, jako získání dat, která budeme aktualizovat (vizte Q-učení vs. SARSA). Znalost prostředí je dalším důležitým kritériem dělení algoritmů, záleží na tom, zda je cílem vytvořit modelu obraz prostředí (*Model-based* vs *Mode-free*). Za další rozdělení se dá považovat množství agentů využitých pro učení (*single* nebo *multi* agentní přístup), metody úpravy poměru prozkoumávání prostředí, kdy je možné využití kontinuálního nebo epizodického trénování, nebo velikost skupin dat, na kterých se trénování modelu provádí.

Parametrů, dle kterých je se algoritmy dají třídit je mnoho, není tak možné je všechny vyjmenovat nebo přesně identifikovat. Je však možné je přiřadit k příkladům využívaných algoritmů.

Zdroje [24]

Q-učení

Jedná se o off-policy algoritmus nezávislý na modelu, učí se tedy pouze jaká akce je nejlepší na základě poskytnutého stavu. Funguje na principu tabulky, která reprezentuje kombinace stav (v řádcích) a akce (ve sloupcích). Aktualizace hodnot v tabulce je zajištěna vztahem:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma * \boldsymbol{max_{a'}} Q(s',a') - Q(s,a)]$$

SARSA

Stejně jako v případě Q-učení se jedná o algoritmus nezávislý na modelu.

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma * Q(s',a') - Q(s,a)]$$

Q - Q hodnota značící pravděpodobnost správnosti vybrané akce

s, s‘ - stav v aktuálním/následujícím okamžiku

a, a‘ - akce pro stav v aktuálním/následujícím okamžiku

r - odměna za akci

γ - důležitost budoucích odměn

α - rychlost učení

Hlavním rozdílem mezi těmito algoritmy je způsob aktualizace hodnoty, kde Q učení využívá maximální možnou odměnu následujícího stavu, zatímco algoritmus SARSA využívá pro aktualizaci odměnu za provedenou akci v následujícím stavu.

Zdroje [33]

5.3 Aplikace zpětnovazebního učení

Zpětnovazební učení lze aplikovat v mnoha odvětvích, nejčastěji se používá v oblastech hraní her nebo robotice, kde je možné při správné implementaci agenta naučit lepšímu ovládání, než by se kdy dovedl naučit člověk. Dále je možné tento typ učení nasadit na zpracování jazyka, například v překladačích pro přirozenější formulaci přeloženého textu, nebo v doporučovacích a reklamních systémech. Tato využití můžeme vidět například na webu, kde se nám na základě dat o prohlížení webových stránek přizpůsobují zobrazované reklamy, nebo na streamovacích službách, jako je Youtube nebo Netflix, kde nám umělá inteligence doporučuje nový obsah na základě historie přehrávání.

6 UMĚLÉ NEURONOVÉ SÍTĚ (ANN – ARTIFICIAL NEURAL NETWORKS)

Neuronové sítě (umělé neuronové sítě / Artificial neural networks – ANNs) jsou velmi rozsáhlým tématem, které nehovoří ani tolik o jednom konkrétním typu modelu, jako spíše o celé kategorii modelů strojového učení. Tyto modely jsou tvořeny vrstvami vzájemně propojených uzlů (neuronů), s jejichž pomocí zpracovávají data, struktura takové sítě je znázorněna na obrázku 3. Každý uzel dostane vstupní data, provede jednoduchou matematickou operaci, a data předává dalšímu uzlu. Struktura vzájemně propojených uzlů umožňuje neuronovým sítím učit se ze vstupních dat úpravou parametrů jednotlivých uzlů.

Každá neuronová sít obvykle obsahuje vrstvu vstupní, výstupní a jednu nebo více vrstev skrytých. Počet uzlů ve vstupní a výstupní vrstvě odpovídá počtu vstupů a výstupů agenta. Vzájemná propojení mezi vrstvami mají váhy, které jsou s odstupem času upravovány s učením modelu.

Zdroje [18][34][36]

Obrázek 3 – Příklad neuronové sítě [25]

Umělé neuronové sítě (NN) jsou přizpůsobivé velkému množství aplikací, mezi které patří například rozpoznávání obrázků nebo zpracování hlasu. Díky své schopnosti se dobře učit, bývají často nasazovány na problémy spojené se zpětnovazebním učením, jako jsou doporučovací, vyhledávací, nebo reklamní systémy. Velmi známým příkladem je například Google vyhledávač.

6.1 Princip neuronových sítí

V neuronových sítích se často setkáváme s termíny backpropagation a feed forward. Tyto termíny vysvětlují cyklus trénování a následného předpovídání dat modelu. Po vytvoření modelu vzniká neuronová síť, kterou si lze představit podobně, jako je tomu na obrázku 3 na přechozí straně.

Zdroje [36]

6.1.1 Dopředné plnění – feed forward

Termín vysvětluje předávání informace mezi uzly v jednotlivých vrstvách v rámci sítě při zpracování vstupních dat. Díky předávání informace mezi vrstvami vzniká ze vstupních dat výsledná hodnota.

Vstupní vrstva

Vstupní vrstva značí způsob, jakým jsou data síti prezentována. Každý prvek dat je přiřazen jednomu neuronu v síti. Příkladem vstupních dat může být například obrázek, kde je každý pixel v obrázku reprezentován jedním uzlem, nebo sekvence datových bodů, využitá pro předpověď počasí nebo pole hodnot reprezentující stav agenta v rámci nějakého prostředí.

Skryté vrstvy

Po předání vstupních dat vstupní vrstvě v síti jsou skryté vrstvy využívány pro výpočet hodnot sloužících pro předpovězení výsledku. Vstupními daty pro jednotlivé uzly v každé z vrstev jsou všechny uzly ve vrstvě předchozí, spolu s váhami, které jsou uzlům nastavovány pomocí trénování. Výsledná hodnota je pak pro každý z uzlů vložena do aktivační funkce (nejčastěji sigmoid [hodnoty v rozmezí 0 a 1] a ReLU [pouze kladné výstupní hodnoty]). Tyto funkce výsledná data upravují a do sítě tak přidávají určitou nelinearitu hodnot umožňující jinak nemožné složitější výpočty.

Výstupní vrstva

Výstupní vrstvě jsou poskytnuty hodnoty z poslední skryté vrstvy. Výstupní vrstva pak tyto hodnoty formuluje do smysluplného formátu. Pro příklad dat z obrázku (jednociferná čísla) na vstupu může být výstupní hodnotou například 10 uzlů, každý reprezentující jiné číslo s pravděpodobností, že se jedná právě o něj. Tato data si následně přebírá uživatel a dále je zpracovává

6.1.2 Zpětné šíření – backpropagation

Výpočet chyby

Chyba je počítána jako srovnání výsledných hodnot předpovězených NN a skutečnou správnou hodnotou. Takto vypočítaná chyba následně slouží pro úpravu vah a biasů v síti. Pro výpočet chyb se nejčastěji používají chybové funkce jako „mean square error (MSE) – střední křížová chyba“ a „cross-entropy – křížová entropie“

Šíření chyby

Při šíření chyby v síti se používá právě zpětné šíření. Na rozdíl od předpovídání hodnot je v tomto případě postupováno odzadu – od vypočtené chyby na výstupních hodnotách po první skrytou vrstvu. Tato část procesu úpravy vah a biasů modelu se na základě vypočtené chyby věnuje identifikací takových hodnot, který nejvíce ovlivňují výslednou hodnotu.

Úprava vah a předpojatostí (bias)

Z předchozího kroku jsou již identifikovány hodnoty, které je potřeba upravit, , v tomto kroku se za pomoci metody gradientního sestupu hodnoty upraví. Tato metoda upravuje hodnoty uzlů ve směru vedoucím k největší změně správným směrem. Hodnota, o kterou se uvedené hodnoty mohou změnit je ovlivněna nastavením rychlosti učení, obvykle označenou jako anglicky jako „learning rate“.

Zdroje [29][30][35]

6.2 Dělení umělých neuronových sítí

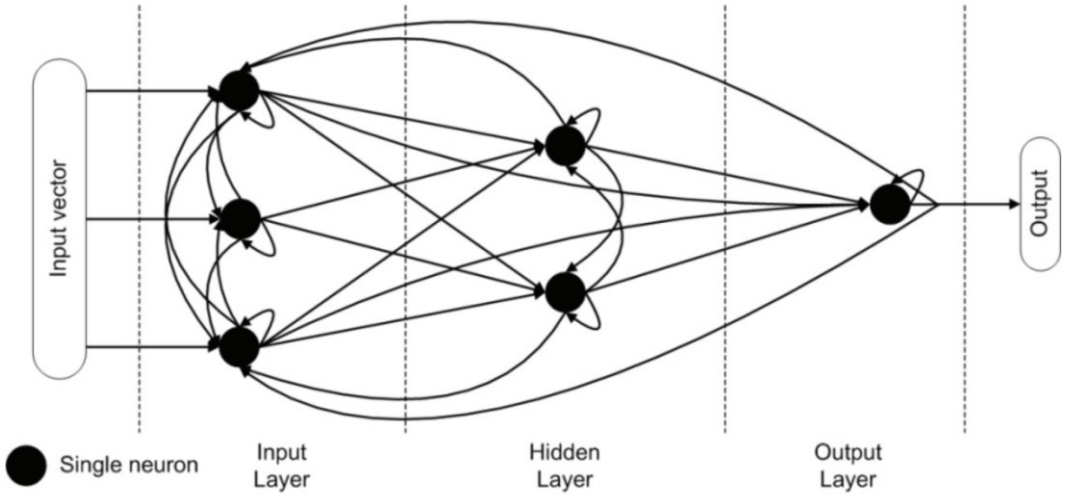
Dopředné sítě

Jsou nejčastěji používaným typem neuronových sítí, jsou tvořeny kombinací více uzlů ve více vrstvách, nastavením vah a biasů je do sítě přidána nelinearita a tak umožněno řešení složitějších problémů Nejčastěji jsou používány pro klasifikaci, regresi a rozeznávání vzorů v datech.

Konvoluční

Speciální typ dopředných sítí s přidanými filtrovacími (konvolučními) vrstvami, používané pro extrahovaní charakteristických vlastností pro vylepšení výkonu a kvality práce především s obrazovými daty. Používají se nejčastěji pro detekci objektů v obrazových datech, klasifikaci obrázků nebo rozdělení obrázků podle jejich obsahu.

Rekurentní



Obrázek 4 - Rekurentní neuronová síť

Jedná se o typ neuronových sítí s vlastní formou „paměti“ umožňující zpracování sekvenčních dat. Data jsou tak ovlivněna nejen aktuálním vstupem ale také vstupy předchozími. Nejjednodušší formou rekurentní neuronové sítě je plně rekurentní neuronová síť, kde je každý uzel v síti spojen s každým dalším uzlem v síti, jak je vidět na obrázku 4. Tyto sítě se používají především k práci s jazykem, tedy k překladům, rozpoznávání řeči nebo analýze textu, nebo v dnešní době rozšířené generativní modely.

Zdroje **Error! Reference source not found.**[36]

7 UMĚLÁ INTELIGENCE VE VIDEOHRÁCH

Hry odjakživa využívaly formy umělé inteligence. Ne vždy se však muselo jednat o umělou inteligenci založenou na strojovém učení. Tato (a další) forma umělé inteligence nabírá každým rokem na popularitě, nejen v oblasti videoherního průmyslu.

Tato kapitola je věnována umělé inteligenci v oblasti videoher, vývoji jejího využití a různým typům AI pro jednotlivé typy her.

Zdroje [26]

7.1 Historie a vývoj AI ve hrách

Za jednoduché a základní využití umělé inteligence, se dá považovat například i jednoduchý algoritmus následující míček ve známé hře pong, kde se v nejzákladnější podobě rozhoduje pro pohyb nahoru/dolů jen na základně aktuální pozice míčku, tedy za podmínky „je míček výš nebo níž“. Už v přechozí kapitole zmíněná hra Spacewar! Využívala podmínkové umělé inteligence pro ovládání protivníků (jednalo se o expertní systém).

V době arkádových her se začínaly objevovat první pokusy o umělou inteligenci v produktech pro masy lidí. Například ve hře PacMan byla využita umělá inteligence pro ovládání nepřátelských „duchů“. Stále se však jednalo o velmi jednoduché formy AI limitované možnostmi hardwaru. Za další zmínku stojí hra Doom, která do světa využívání umělé inteligence ve hrách přinesla základní formu „hledání cest“ (protivníci „pronásledující“ hráče po mapě).

AI se dále vyvíjela, postupem času bylo dosaženo komplexnějšího chování NPC (non-player character / nehráčská postava) nebo schopnost navigace po dynamicky se rozšiřující mapě. V posledních letech jsou také využívány formy AI učící se dle hráčova chování ve hře – což napomáhá individuálnímu hernímu zážitku.

Zdroje [26][27]

7.2 Praktické použití AI ve hrách

Umělá inteligence se ve hrách používá velmi často k mnoha různým účelům. Hlavním cílem využívání umělé inteligence je vylepšení hráčského zážitku a realističtější prostředí s uvěřitelnými postavami. AI má tedy místo v každé oblasti vývoje videoher.

Tvorba obsahu

Velmi často se můžeme setkat s využitím pro dynamickou tvorbu obsahu hry, především tvorbu herního prostředí, na které se přímo váže herní zážitek. Nejlepším příkladem toho jsou velmi známé sandboxové hry Minecraft nebo Terraria, ve kterých je otevřený svět přímo generován pro hráče.

Chování nehráčských postav

Chování nehráčských postav je další oblastí, které je umělá inteligence věnována. Je snaha chování postav co nejvíce přiblížit chování hráče. Je tedy třeba zohlednit reagování na různé podněty, navigaci po mapě světa, kde mapa nemusí být postavě známa.

Dynamická obtížnost

Úprava obtížnosti hry dle schopností hráče je využita pro dosažení optimálního herního zážitku. Tuto oblast lze řešit dynamickou úpravou přednastavených úrovní obtížnosti, nebo přímou úpravou chování postav dle akcí hráče. Postavy se tak mohou učit z akcí hráče a tyto schopnosti využívat proti, nebo pro, hráče samotného.

Chatovací roboti

Chatovací roboti jsou využíváni jako samostatné aplikace, kde jsou velmi často používáni jako „asistenti“ na webových stránkách, pro chytré domácnosti, nebo ve hrách.

Testování her

Umělá inteligence je používána také pro testování her, kde zajišťuje například generování skriptů pro automatizované testování, testování kompatibility s různými platformami, testy bezpečnosti, hledání chyb, výkonnostní testy, nebo simulaci hráčského chování. Při využití AI pro automatizaci některých částí testování je možné omezen lidské kapacity směrovat do potřebnějších oblastí.

Zdroje [29][28]

8 NÁVRH VIDEOHRY

Zadání je třeba rozdělit na dvě hlavní kategorie, těmi je návrh a implementace videohry a následně návrh a implementace klientské aplikace umožňující vlastní ovládání hry. Tato kapitola se věnuje návrhu videohry, jsou v ní rozebrány základní herní mechaniky, herní prostředí a předpokládané chování protivníků.

8.1 Koncepty a cíle

Cílem této části práce je vytvoření jednoduché závodní videohry, ve které bude mít hráč možnost závodit na různých tratích sám na čas, či s protivníky jejichž chování obstará oddělené klientská aplikace. Hra slouží především jako demo prostředí pro ověření schopnosti ovládání vozu klientskou aplikací. Klientská aplikace bude jednoduchá konzolová aplikace vytvořená v Pythonu.

8.2 Požadavky na hru

8.2.1 Funkční požadavky

- Ovládání hry zajištěno šipkami na klávesnici
- Vozidlo se pohybuje realistickým způsobem po trati
- Tratě jsou uživatelem vytvořitelné a editovatelné
- Auto může jezdit pouze v oblasti trati a nemůže vyjet ven
- Při jízdě po trati se hráči měří čas

8.2.2 Nefunkční požadavky

- Uživatelské prostředí by je přehledné
- Hra běží plynule při zachování konzistentní snímkové frekvence
- Hra je hratelné klávesnici a ovládá se šipkami

8.3 Herní prostředí a design úrovní

Hra samotná bude obsahovat jednoduché menu, ve kterém si hráč bude moci vybírat z jednotlivých herních režimů a nastavení hry. Herní módy budou obsahovat:

- Základní mód volné jízdy na čas, ve kterém se hráč do hry připojí sám (bez protivníků) a bude si moci trénovat jízdu na jednotlivých tratích.
- Mód závodu – jízdy proti protivníkům, umožňující hráči připojení klientů do hry (klienti se do hry připojují v separátní aplikaci

8.4 Herní mechaniky

Jedná se v základu o velmi jednoduchou závodní hru. Auto, ať už ovládané hráčem nebo AI klientem, má pevně nastavené maximální parametry, jimiž je limitována rychlost a úhel zatočení. Tyto limitní parametry jsou však jen maximální hodnotou, které auto může dosáhnout.

8.4.1 Rychlost

Auto si drží mimo maximální rychlosti také svou aktuální rychlost, jejíž hodnota s v čase mění na základě ovládání vozu. Rychlost se nemění skokově, ale je implementován její lineární nárůst a pokles, jedná se o jednoduchou simulaci reálné fyziky. Při přidání plynu auto postupně zrychluje templem definovaným parametrem akcelerace vozu až do dosažení maximální rychlosti, kde zrychlovat přestává a tuto rychlost si drží do puštění plynu případně započetí brzdění.

8.4.2 Zatáčení

Druhým parametrem, kterým je řízeno ovládání vozu, je zatáčení. Stejně jako v případě rychlosti je i zatáčení limitováno maximálním úhlem a rychlostí změny úhlu zatočení. Na rozdíl od maximální rychlosti je maximální úhel zatočení pro všechna auta stejný a parametry nastavení vozu mění jen rychlosti zatáčení.

8.4.3 Traťové limity

Trať je ohraničena obrubníkem a svodidly, kde obrubník je součástí trati. Vše ostatní, co se nachází mimo trať s obrubníkem a svodidla, je označeno za plochu „mimo oblast trati“. Do této oblasti není možné autem vjet, při pokusu o najetí do této oblasti nastává kolize.

8.4.4 Kolize

Ve hře je nutné implementovat kolizní systém, který zajistí chování vozů při nastalé kolizi. Chování vozu při nastalé kolizi bude stejné pro auta ovládaná hráčem i pro auta ovládaná AI klientem. Při kolizi se auto okamžitě zastaví.

8.5 Klientem ovládané vozy

Cílem pro chování aut ovládaných AI klientem je napodobení a co nejbližší přiblížení se autu ovládanému hráčem. Všechny vozy na trati tak budou mít stejné nastavení a stejné chování, aby nebyly klientské vozy znevýhodněny.

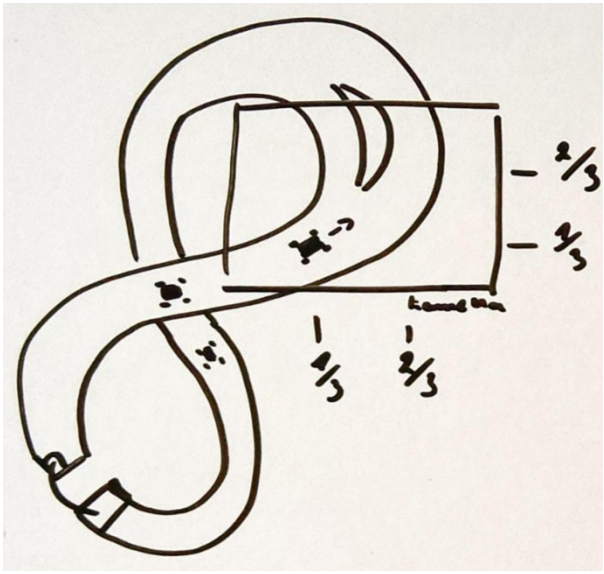
9 REALIZACE VIDEOHRY

Koncept videohry je z přechodí kapitoly, zabývající se konceptem samotným, jasný. Vlastní realizace však, podobně jako jakýkoli jiný vývoj přináší řadu problémů, které je třeba řešit, jak se říká, „za pochodu“. Tato kapitola je tedy mimo hlavních kapitol, kterými jsou programování, návrh a testování, věnována také vybraným konceptům, a to jak těm, které se úspěšně přetvořily v konkrétní herní mechaniku, tak i těm, které neuspěly a bylo třeba je přetvořit, případně ze hry vyřadit.

Pro práci platí využívání enginu Godot ve verzi 3.5.3 (v době začátku psaní práce aktuální vydaná LTS verze) a Pythonu ve verzi 3.12.3.

9.1 Ověření konceptu

Ověření konceptu, které se dá označit za „nultou“ fázi vývoje hry, bylo věnováno především ověření, možnosti hru vytvořit, napojit na ni klienta napsaného v Pythonu a dostat do stavu „spustí se to a nějak to funguje“. Tento stav obnášel jen jednoduchý obdélník reprezentující auto, jezdící po namalované trati se základním ovládáním, a to jak od uživatele, tak externím klientem.



Obrázek 5 - První koncept videohry

Obrázek 5 je původním jednoduchým návrhem videohry. Je v něm viditelná inspirace známými hrami typu Mario Kart. Dále je na obrázku znázorněna také pohyblivá kamera, jejíž pozice se dle rychlosti vozu mění. Ta byla navržena pro zlepšení pocitové dynamiky jízdy. V této fázi vývoje byla vynechána, neboť pro ověření fungování hry nebyla třeba.

Na obrázku jsou znázorněny také 2 pokročilé herní mechaniky, křížení trati a překážka na trati – skok. Tyto herní mechaniky byly pro byly pro účely práce vynechány a jsou zařazeny do dalších směrů, kterými je možné hru dále vyvíjet.

K první verzi videohry se řadí také první verze klienta – ve které byl pro rozhodování o akci použit jen jednoduchý generátor náhodně odesílaných stavů, později nahrazen jednoduchou podmínkou.

9.1.1 Vývoj

Vývoj hry v této (nulté) fázi, zahrnoval nalezení tutoriálů na projekty podobného typu: jezdění autem po okruhu, propojení s Python aplikací a rozhýbání vozu náhodnými pokyny. Vlastní programování bylo tedy ponecháno až do dalších fází vývoje hry.

9.1.2 Výsledek

Při testování nulté fáze vývoje hry byla potvrzena možnost spojení hry s python aplikací protokolem WebSocket, a to za pomoci integrovaného WebSocket serveru pro Godot a staženým balíčkem *websockets (verze 12.0)* pro klientskou Python aplikaci. V následujících kapitolách je rozebráno vlastní programování, popsána aktuální struktura projektu, tvorba grafiky a ostatní vývoj.

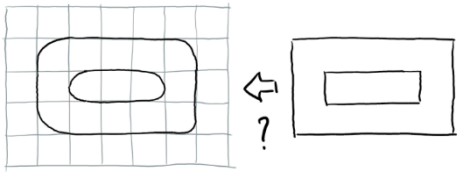
9.2 Fáze 1 – Seznámení s herním enginem Godot

Vývoj první verze hry byl zaměřen především na seznámení se s vývojem videoher, s herním enginem Godot, možnostmi jeho rozšíření o balíčky a skriptovacím jazykem GDScript. Vývoj videoher je velmi zajímavý proces, který přináší úskalí spojená s oblastmi, se kterými se v jiném vývoji nesetkáváme. Tato úskalí se nejčastěji pohybují okolo grafického zpracování projektu.

V případě vývoje videohry byl kladen důraz na co nejlepší hráčský zážitek. Ten je v případě závodní hry ovlivněn podobností s reálným světem. Od implementace složitější fyziky bylo ale ustoupeno, neboť by to přinášelo další komplikace pro klientskou aplikaci ovládající hru.

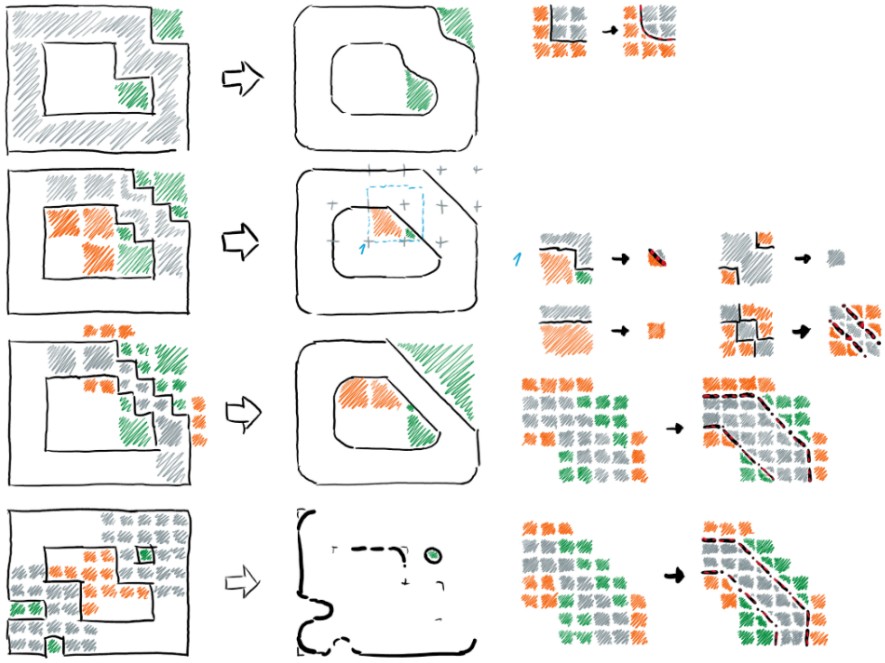
9.2.1 Grafika a fungování tratí

Tvorba grafiky si s v této fázi vývoje hry zasloužila vlastní kapitolu, především jejímu návrhu bylo totiž věnováno poměrně velké množství času.



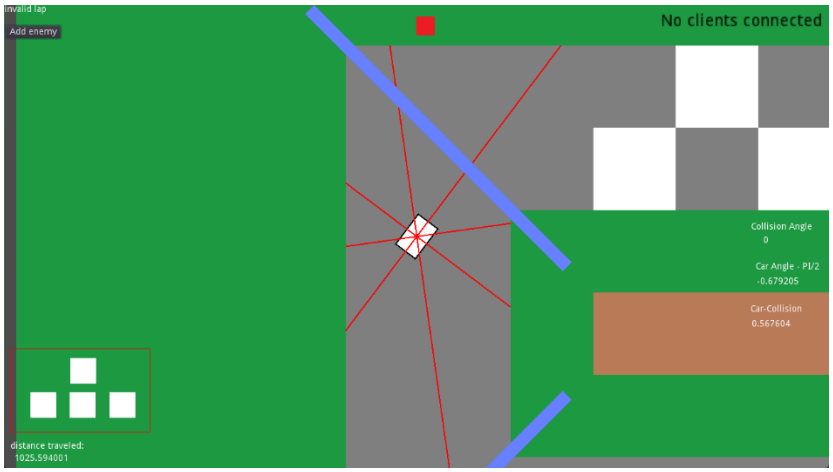
Obrázek 6 - Původní návrh generování tratě

Původním plánem bylo tratě nechat vytvářet jen jako jednoduchou matici s pozicemi „velkých dlaždic“ (obrázek 6) a tyto velké dlaždice následně dynamicky předělávat na dlaždice malé. Ukázalo se však, že tento přístup k tvorbě trati přináší více komplikací, než sám řeší. Cílem tohoto přístupu bylo zpříjemnění pohledu na hru, protože na zaoblené přírodou inspirované linky se vždy kouká příjemněji než na kolmé čáry.



Obrázek 7 - Návrh grafiky a generování trati

Při dalším postupu v implementaci vlastního algoritmu pro generování tratě se ukázalo, že by tento přístup přinášel složitou implementaci příliš neovlivňující výsledek práce, bylo od něj tedy upuštěno a přistoupeno k jednoduché variantě negenerované trati s výsledkem viditelným na dalším obrázku.



Obrázek 8 - Ukázka grafiky a UI pro verzi

Na obrázku 8 můžeme vidět také testovací uživatelské rozhraní. V levé části můžeme vidět ukazatel ovládání pro vizuální kontrolu činnosti AI klienta, v horní části kontrolu směru kontrolovaného vozu a okolo okrajů obrazovky různé informační hlášky pro kontrolu akcí

9.2.2 Testování a závěry

Při testování a revizi kódu bylo nalezeno několik problémů které by bylo nepřehledné, nebo příliš složité předělat, mezi ty se řadila především nepřehlednost kódu a nedodržování správných jmenných konvencí. Výsledkem testu byl tedy požadavek na přepracování hry do finální podoby.

9.3 Fáze 2 – Přepracování hry

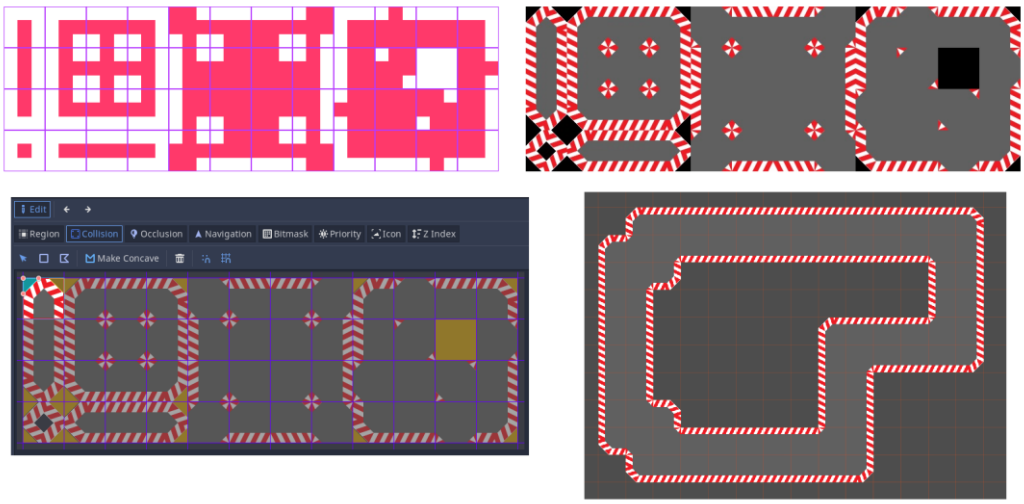
Z první fáze byly použity jen části vybraných komponent, jako je ovládání vozu, složková struktura programu a některé funkční komponenty, jako je ovládání WebSocket serveru.

Fungování hry jako takové bylo v pořádku, vývoj byl tedy ve fázi přepracování hry věnován především s ohledem na zpřehlednění kódu, sjednocení jmenných konvencí nebo přepracování struktury složek. Dále byla do hry implementována jednodušší verze grafiky dle návrhu z 1. fáze hry.

Od uživatelského editoru tratí bylo zatím upuštěno, neboť v omezeném čase na práci nebyl nalezen způsob jak správně generovat nebo uživatelsky přidávat checkpointy důležité pro kontrolu směru jízdy po trati.

Grafika

Do této fázi hry byla implementována jednoduchá verze původního nápadu týkající se generování trati. Byl využit komponent *Tilemap* – objekt nabízený enginem pro tvorbu map z uniformních dlaždic, jejich ukázku můžeme vidět na obrázku 9. Na rozdíl od původního nápadu však nebyl využit vlastní algoritmus pro generování poddlaždic, ale funkcionalita *Autotile*, která dokáže, na základě bitové mapy, spojovat dlaždice a automaticky z nich při práci ve 2D editoru vytvářet opticky spojité objekty.



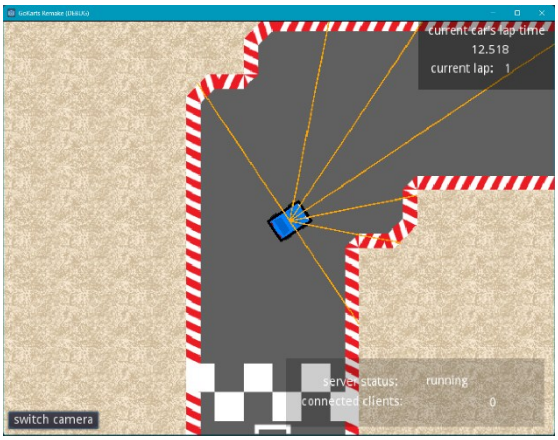
Obrázek 9 - Ukázka využití tilemap v hře

9.4 Fáze 3 – Doplnění chybějících částí a revize kódu

V rámci této fáze vývoje byly doplněny chybějící části hry, mezi které patří editor tratí, možnost jízdy na vlastnoručně vytvořených tratích lepší uživatelské nastavení hry nebo revize kódu, ve kterém, kvůli neznalosti platformy Godot a odlišnému fungování GDScriptu od ostatních programovacích jazyků, vznikaly nepřehledné části.

Uživatelské prostředí hry nyní nabízí menu, ze kterého je možné přejít do hry a práce s klientem, uživatelské nastavení, nebo editor tratí, ve kterém je možné vytvoření vlastní trati, na které je následně možné jezdit a testovat klientskou aplikace.

Obrázek 10
ukázka herního prostředí v aktuálním stavu



Obrázek 11
ukázka herního prostřední v aktuálním stavu

Jak je vidět na obrázcích 10 a 11, auto při svém pohybu po trati využívá senzory, tyto senzory odesílají informace o svých vzdálenostech do vzdálenosti 250px, což při velikosti dlaždice 64px a nastavení měřítka trati na 0,7 vychází ~ 5,55 dlaždice. Trať je široká 3 dlaždice, takže je tato vzdálenost dostačující.

9.5 Závěr vývoje hry

Při tvorbě práce byla přehodnocena část herních mechanik, jednalo o mechaniky, které výrazně neovlivnily základní princip hry. Hra je v aktuálním stavu použitelná spíše pro prezentační účely. Před dalším pokračováním vývoje hry v dalších fázích by bylo vhodné do hry implementovat více funkčních herních režimů pro zlepšení uživatelského zážitku nebo příjemnější uživatelské nastavení. Za důležité je také přidání možnosti spuštění hry na ostatních platformách, neboť se, i díky jednoduché implementaci multiplayeru, jedná o ideální jednoduchou hru pro telefon.

Možnosti budoucího rozvoje hry:

- Export pro mobilní platformy
- Dokončení rozpracovaných částí hry
- Aktualizace enginu na novou verzi
- Přenos klienta ze separátní aplikace do hry
- Optimalizace hry pro ostatní platformy (pro mobilní především)
- Přidat možnosti uživatelského nastavení
- Vylepšení prostředí
    - Algoritmy pro generování trati
    - Přidání nových herních mechanik
    - Skok, křížení trati, aplikace

10 NÁVRH AI KLIENTA

Tato kapitola se věnuje návrhu AI modelu, průzkumu možností možnostem využití různých typů strojového učení, finální volbě a finální volbě.

10.1 Popis požadavků na klienta

Klientská aplikace bude pouze konzolovou aplikací zpřístupňující nastavení a trénování modelu. Jeho základní funkcionalitou bude připojení k WebSocket serveru, běžícímu ve hře. Po připojení má klientská aplikace za úkol sbírat data. Tato data následně musí pro získání relevantních hodnot filtrovat a transformovat, vytrénovat na nich model a následně se připojit k serveru ve hře a spustit předpovídání a odpovídání s co nejkratší časovou prodlevou.

10.2 Typ využité AI

Před implementací klienta není požadavek na využití konkrétního algoritmu, práce se tedy bude zabývat implementací a srovnáním jednotlivých typů, od využití expertního systému až po implementaci neuronové sítě.

10.3 Průzkum ostatních implementací

10.3.1 Využití genetických algoritmů

Oblast autonomního řízení vozidel je v poslední době často skloňované téma. Při hledání návodů a nápadu na tvorbu simulace jízdy autem po trati je tak možné velmi často narazit na použití genetických algoritmů. Takové algoritmy je možné aplikovat především na tratě pevného rozložení. Zaměřují se na nalezení nejlepší cesty v rámci trati. Fungují na principu napodobování genetiky a evoluce, kdy se agenti posílají po „generacích“, kde jsou data modelů nastavená náhodně. Nejlepší z agentů jsou dále „množeni“ a jejich „potomci“ jsou do prostředí posláni znovu. Jejich parametry se různě mírně upravují, a díky tomu je dosahováno lepších výsledků.

Jak ve své práci uvádí tým ze španělských univerzit, „The use of GAs has been proved to be a successful technique for the optimization of a lap trajectory in a known circuit.“ Volně přeloženo jako „Použití genetických algoritmů se ukázalo být úspěšné pro použití k optimalizaci ideální jízdní stopy na známém okruhu.“ Stejně tak však ve své práci uvádí, že použití genetických algoritmů pro jízdu po trati je složité, pokud je naší snahou objet více než 1 kolo, protože výsledek předpovědi genetického algoritmu závisí na počátečním stavu vozu, a ten je v případě jízdy více než jednoho kola pokaždé jiný.

Zdroje **Error! Reference source not found.**[37]

10.3.2 Umělé neuronové sítě s využitím zpětnovazebního učení

Při využívání zpětnovazebního učení pro vytvoření modelu je většinou za potřebí algoritmu dodat nějaká základní data. Tato data jsou využívána pro základní ovládání modelu a následné získání zpětné vazby z prostředí, ve kterém se agent pohybuje.

Tým z University of Regina v Kanadě ve své práci přidává agentovi vlastní pohled na herní prostředí, kdy zohledňuje i blízké okolí vozu a předchází tak kolizím s ostatními vozy na trati.

Zdroje [38][39]

10.3.3 Srovnání

Týmy pro získání stavu vozu často používají dva přístupy. První přístup je kamera, ta je častěji využívána při aplikaci na osobní auto nebo na RC model. Druhým přístupem je nějaká forma senzorů vzdáleností, to je naopak přístup častěji používaný pro simulátory, neboť jejich implementace je jednodušší než využití kamery. Vzhledem k potřebám a představě o řešení této práce bylo vybráno řešení využívající neuronové sítě bez závislosti na modelu, tedy bez využití evolučních algoritmů.

11 REALIZACE AI KLIENTA

Pro klientskou aplikaci byl zvolen skriptovací jazyk Python (ve verzi 3.12.3) pro svou jednoduchost, a především rozšířenost v oblastech strojového učení. Pro Python je k dispozici velké množství knihoven a nástrojů pro práci s algoritmy strojového učení, neuronovými sítěmi nebo zpracování dat.

11.1 Ověření konceptu

První (nultá) verze klienta byla vytvořena za účelem ověření připojení klientské aplikace ke hře a implementaci ovládání. Jednalo se tedy především o výběr balíčků pro komunikaci a implementaci komunikace se hrou samotnou. Po krátkém průzkumu byla pro připojení k serveru vybrána knihovna websockets (ve verzi 12), která připojení na server vytvořený ve hře nabízí.

```
{
    "left": 23.368393,
    "front_left": 4.452423,
    "front": 6.42569,
    "front_right": 70.660263,
    "right": 70.660263,
}
```

Obrázek 12 - data odesílaná v původní verzi

V první řade bylo třeba zajistit komunikaci mezi serverem (hrou) a klientem (AI), pro to bylo tedy přistoupeno k odesílání dat ve formě textových řetězců JSON, jehož ukázka je na obrázku 12. Tento formát dat byl vybrán pro svou jednoduchost a přehlednost i v případě zachytávání komunikace. Touto formou jsou stavy odesílány i ve finální verzi projektu.

Dále bylo třeba ověřit rozebrání dat na straně klienta, což bylo díky formátu odesílaných dat velmi jednoduché. Data jsou následně převedena do objektu, na jehož základě je vybrána odpovídající reakce z klienta. V původní verzi byla odpověď jen náhodně generovaným celým číslem od 0 do 3, kde každé číslo reprezentuje nějakou akci, kterou má auto provést.

Důležitým parametrem souvisejícím s odesíláním dat je frekvence jejich odesílání, která byla na začátku nastavena na 100ms. Tato frekvence se zdála být dostatečně častá pro rychlost hry, a zároveň dostatečně dlouhá pro získání a zpracování odpovědi z klienta. Pro zjištění reakční doby byl využit nástroj na odchytávání komunikace – Wireshark. Komunikace byla odchytávána na lokálním loopback rozhraní.

Tento test ukázal, že doba zpracování dat není konzistentní a není tak možné ve hře na odpověď z klientské aplikace čekat. Komunikace se hrou probíhá nekontrolovaně (není

explicitně potvrzováno přijetí a zpracování akce). Hra – server odesílá stavy každých 100ms, klient stavy přijímá a reaguje na ně akcí.

Tato kapitola se dále věnuje popisu implementace využitých typů umělé inteligence, od nejjednodušší varianty ve formě expertního systému, po hluboké neuronové sítě. Některé typy umělé rozhodování o akci se ukázaly být pro hru nevhodné, jejich forma byla tedy zavrhnuta a nahrazena novou variantou.

11.2 Komunikace klienta se hrou

Klient se hrou komunikuje pomocí protokolu Websocket

```
# Handshake
# introduce 'self to the server
introduction = {"connection_type" : ConnectionType.PLAYER.name}
await connection.send(json.dumps(introduction, ensure_ascii = False).encode("utf-8"))

# wait for server confirmation
# confirmation message is expected to be str "CONFIRM"
incoming_bytes = await asyncio.wait_for(connection.recv(), 1)
if (incoming_bytes.decode("utf-8") != "CONFIRM"):
    raise Exception("Client not accepted")
else:
    print("Client accepted")
```

Obrázek 13 - ukázka handshaku klienta směrem k serveru

Pro zjednodušení práce při sběru dat a následném připojení pro ovládání vozu bylo nutné přidat klientské aplikaci možnost se serveru představit a oznámit, o jaký typ klienta se jedná (ukázka na obrázku 13). Tato implementace vlastní formy handshaku umožnuje jednoduché připojení klientů různých typů.

11.3 Verze 1 – Expertní systém / Final State Machine

Tento typ AI byl využit pouze v jedné z prvních verzí pro testování reakční doby klienta. Jedná se o jednoduchou strukturu podmínek, která dokázala auto po trati úspěšně navádět. Pro rozhodování o další akci auta byly využívány jen 3 z 5 senzorů vzdáleností, přičemž senzor vzdálenosti vpředu určoval pouze zda se auto má pohybovat vpřed, nebo couvat. Pro rozhodování o tom, kterým směrem se auto vydá, byly využity pouze 2 diagonální senzory a jednoduchá podmínka určující, že auto má jet tam, kde je vzdálenost delší. Ukázka kódu je na obrázku 14.

```
def choose_action_by_condition(game_state: GameState) -> int:
    accelerate = False
    turn_left = False
    turn_right = False
    go_backwards = False

    if game_state.sensor_front > 40:
        accelerate = True
    else:
        accelerate = False

    if game_state.sensor_front_left > game_state.sensor_front_right:
        turn_left = True
        turn_right = False
    elif game_state.sensor_front_left < game_state.sensor_front_right:
        turn_left = False
        turn_right = True
    else:
        turn_right = False
        turn_left = False

    ## výběr akce dle nastavených proměnných
    # nic / zastavení
    action = 0
    # couvat, aby se auto nezasekávalo
    if not accelerate:
        action = 4
    # akcelerace
    elif accelerate and not turn_left and not turn_right:
        action = 1
    # akcelerace a zatočení vlevo
    elif accelerate and turn_left and not turn_right:
        action = 2
    # akcelerace a zatočení vpravo
    elif accelerate and not turn_left and turn_right:
        action = 3

    return action
```

Obrázek 14 - ukázka jednoduchého rozhodování o akci

11.4 Verze 2 – Q-Učení tabulkovou formou

Druhou variantou pro implementaci strojového učení do hry bylo využití Q-Learningu(Q-učení), tabulkovou formou. Teoreticky se tento typ strojového učení zdál pro využití v tomto případě ideální, ukázalo se však, že této implementaci stálo v cestě několik překážek. Největší překážkou se ukázal být kontinuální stavový prostor (senzory odesílající stavy jako kladné reálné číslo). Tento stavový prostor bylo tak třeba vhodně rozdělit.

Pro výpočet odměny byla využita funkce, počítající změnu stavu při každém odeslání dat. Tato funkce byla implementována na straně serveru (ve hře) a odměna je počítána ze surových dat. Výpočet odměny byl počítán z rozdílu stavů mezi přijetím poslední akce a odesláním nového stavu.

11.4.1 Varianta 1 – omezená množina stavů

Jako první nápad pro implementaci tabulky bylo omezení množiny stavů. Stav, přijatý tak, jak je uvedeno na obrázku *„data odesílaná v původní verzi“*, byla odeslána do metody, která danou množinu omezila na 4 stavy pro senzor. Data pak vypadala asi takto:

```
{
    "left": 3,
    "front_left": 0,
    "front": 1,
    "front_right": 2,
    "right": 1,
}
```

Obrázek 15 - omezená množina stavů

Tento přístup se po pokusech o odladění ukázal být nevhodný. Problémovými se ukázala být frekvence komunikace v kombinaci se způsobem výpočtu odměny a příliš omezenou množinou stavů. Komunikace s prodlevou 100ms způsobila, že rozdíl stavů pro výpočet odměny byl příliš malý, a agent tak z vypočítaných odměn nedostával správné reakce. Stávalo se, že pro stav z omezené množiny, který agent viděl, byly hodnoty kladně vypočítány pro více stavů, a naučit agenta smysluplnému chování nebylo možné.

11.4.2 Varianta 2 – rozšíření množiny stavů

V druhé verzi agenta byla množina stavů rozšířena o informace o kolizi a množina stavů, na které byly kontinuální stavy redukovány se zvýšil na 5 a 6 stavů pro vybrané senzory. Po tomto rozšíření se velmi brzy začalo zdát, že se model podaří v krátkém čase vytrénovat, ale kvůli špatnému výpočtu odměny bylo auto stále dostáváno do situací, ze kterých se nedokázalo dostat.

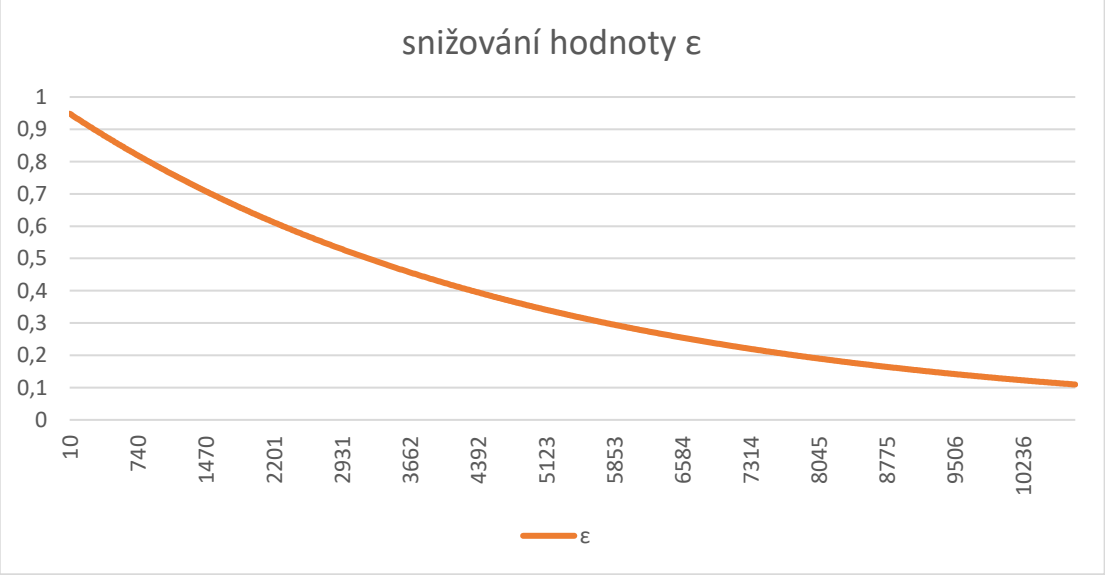
V této verzi agenta bylo také dosaženo limitů tabulkového Q-learningu. Rozšiřování množiny stavů nebylo dále možné a trénování by zabralo s nejistým výsledkem velké množství času. I s použitím redukovaných stavů je aktuálně v tabulce 10800 stavů. Skutečnost, že některých kombinací nemusí být na trati vůbec dosaženo, by problém nebyla, pokud není stavu možné dosáhnout, nijak nezatěžuje fungování modelu. Problém však je proměnná epsilon, pro kterou je složité navrhnout optimální hodnotu. Hodnota se snižuje s každým proběhnutým trénováním a určuje, s jakou pravděpodobností bude agent prozkoumávat prostředí (určuje pravděpodobnost náhodné akce).

10800 stavů bylo dosaženo pomocí jednoduché kombinatoriky. Stavový prostor se skládá z přijatých vzdáleností z 5 senzorů a informace o kolizi.

- Přední senzor: 6 stavů
- Přední – levý senzor: 6 stavů
- Přední – pravý senzor: 6 stavů
- Levý senzor: 5 stavů
- Pravý senzor: 5 stavů
- Kolize: 2 stavy

10800 = 2 * 5 * 5* 6 * 6 * 6

Při počtu 10800 možných stavů modelu a předpokladu, že model dosáhne všech z nich minimálně 1x, by trénování zabralo minimálně 18 minut.



Graf 1 - snižování hodnoty epsilon v čase: zdroj vlastní

S ohledem na množství stavů (10800), jakého bylo dosaženo v této variantě klienta s tabulkovou formou reprezentace dat bylo třeba způsob učení a výpočtu odměny předělat.

11.4.3 Varianta 3 – přepracování množiny stavů

Ve třetí implementaci tabulkové formy Q-Učení nebyla množina stavů definována pevně nastavenými hodnotami pro senzory, nýbrž poměry mezi vzdálenostmi. Využívá tedy principy expertního systému, s rozdílem, že výstupem podmínek nejsou akce, ale indexy stavů, vstupující do Q tabulky.

Hlavní problém tohoto přístupu k datům byl, po předělání převodu stavu na jeho index, způsob výpočtu odměny. Stavový prostor pro tabulku se výrazně zmenšil, vypočet odměny již tedy nebylo možné aplikovat každých 100ms, neboť by nastala stejná situace jako v případě první varianty s omezenou množinou stavů (nekonzistence odměn z důvodu více správných akcí pro jeden stav).

11.4.4 Zhodnocení

S ohledem na množství stavů a jejich kombinací se tabulková forma klienta s využitím Q – Učení se ukázala být nevhodná. V další verzi klienta bylo tedy přistoupeno k využití hlubokých neuronových sítí.

11.5 Verze 3 – Využití hlubokých neuronových sítí

11.5.1 Volba knihovny

Jako poslední a zároveň nejsložitější varianta implementaci klienta byly zvoleny hluboké neuronové sítě, jejichž fungování je popsáno v kapitole 6 – Hluboké neuronové sítě. Pro implementaci byla zvolena python knihovna TensorFlow ve verzi 2.16.1. Jedná se o open source platformu umožňující zjednodušenou práci s modely strojového učení. Knihovna TensorFlow je dostupná na často využívaných platformách, mezi nimiž jsou osobní počítače, mobilní, webové a cloudové platformy. V kontextu této práce je knihovna TensorFlow využívána po boku API Keras, které zjednodušuje práci s neuronovými sítěmi. Nabízí předdefinované modely, optimalizační funkce nebo různé typy vrstev. Obecně však dokáže API Keras využívat také knihovny JAX nebo PyTorch.

11.5.2 Tvorba a nastavení modelu

Základní nastavení modelu je spolu s API Keras jednoduchá záležitost. Pro základní fungování stačí znát velikost vstupního vektoru, formát výstupních dat a pro začátek odhadnout počet a velikost skrytých vrstev na základě komplexnosti úlohy. V tomto konkrétním případě se jedná o následující parametry:

- Vstupní vektor: 10 × X
- Formát výstupních dat: 4 × boolean hodnota

Velikost vstupního vektoru je 10 hodnot, kde 7 z nich jsou hodnoty senzorů, 1 aktuální rychlost auta, 1 je boolean značící kolizi, 1 aktuální natočení ko30l. Při využívání API Keras je vstupní pole vždy o pevné šířce i délce, to znamená, že i část vstupního vektoru značící kolizi je reprezentována stejně velkými daty jako hodnota určující vzdálenosti překážek. Optimalizace těchto hodnot by tedy znamenala například sloučení informace o kolizi s hodnotou natočení kol, tedy tvorba datového typu s vlastní bitovou reprezentací, což je při malém množství dat zbytečné.

11.5.3 Identifikace a řešení problémů

Při tvorbě modelu v rámci klientské aplikace bylo nalezeno několik překážek, které bylo potřeba překonat.

Jednou z překážek se zdál být výkon počítače, u kterého se zdálo, že nebude možné neuronovou síť využít, protože by model nestíhal odpovídat v požadovaném intervalu, který byl od začátku zvolen na 100 ms. V rámci pokračování ve vývoji se ukázalo, že výkon PC nebude problémem, a tak bylo možné interval komunikace zkrátit, a to až na dvojnásobnou frekvenci. Interval se tedy změnil ze 100 ms na 50 ms.

Jako největší překážka při tvorbě modelu se však ukázalo být vlastní nepochopení fungování modelu. Neuronové sítě fungují na principu reakcí aktivovaných uzlů na uzly jim předcházející. Jednotlivé uzly jsou reprezentovány bitovými hodnotami, stejně jako všechno ostatní ve světě digitálních technologií, a tak jsou problém především datové typy, jejichž špatná volba může vést až k fatálním problémům.

Při nastavování modelu se musí nastavit několik parametrů, mezi nimiž je například formát vstupních dat. Tato data jsou reprezentována dvourozměrným vektorem, kde modelu z API Keras stačí velikost jednoho rozměru tohoto vstupního vektoru, druhý rozměr je určen na základě vstupních dat.

Čísla jsou v pythonu defaultně reprezentována číselným formátem float, jenž je v paměti reprezentován 64 bity, to je množství informací, se kterými se umělá neuronová síť musí naučit pracovat. Hodnota float je uložena jako 64 bitů, je možné do ni tedy uložit až 2^64 (tj. 18 446 744 073 709 551 616) rozdílných hodnot. To je pro potřeby modelu zbytečné a výpočet takových dat stojí velké množství výpočetního výkonu. V tomto případě je však

výrazně důležitější rychlost odpovědi, neboť na odpověď má klient jen 50 ms, do kterých je třeba započítat i ztráty jednotek ms při komunikaci přes protokol Websockets.

Omezení množiny stavů mělo být zavedeno od začátku, problém byl však v její implementaci. Keras nevyžaduje zadání obou rozměrů vstupního vektoru, tak se může snadno stát, že jsou do vstupní vrstvy vložena data v nesprávném formátu.

Jako řešení byla zvolena reprezentace vhodnějším datovým typem, v tomto případě uint8 – unsigned int 8bit, který je, jak je z názvu zřejmé, reprezentován 8 bity bez znaménka a nabývá hodnot 0 – 255. Touto změnou bylo dosaženo schopnosti modelu vytrénovat se na poměrně malém vzorku dat (několik set hodnot). I takto malý vzorek dat modelu stačí pro aproximaci ostatních hodnot, neboť při 8bitové reprezentaci jsou hodnoty velmi podobné. V případě dalšího vývoje by stačilo data lehce nasegmentovat a bylo by tak možné i využití menšího vstupního vektoru dat.

Výsledkem této změny byl radikální posun v práci, kdy trénování modelu stačí jen malé množství označených dat, model se trénuje i odpovídá rychle, a zabírá výrazně méně místa v paměti. Takto optimalizovaný model je tedy možné provozovat i na slabším HW.

11.5.4 Trénování modelu

Trénování modelu má více fází, první fází trénování je sběr označených dat. To jsou data obsahující páry vstup-výstup, v případě tohoto modelu se jedná o data o vzdálenostech snímaných jednotlivými senzory, rychlosti, kolize s překážkou a úhlu natočení kol.

Další fází trénování modelu je poskytnutí dat modelu a spuštění tréninku. Trénink v tomto případě nezabral příliš času, neboť vstupní vektor ve formátu 10 × 8 bitů je poměrně malý. Data je i jednodušší aproximovat, tak pro správné odhady modelu stačí menší množství dat.

11.5.5 Optimalizace vstupních dat

```
"state": {
    "l": 67.259903,
    "fl": 93.310089,
    "ffl": 167.897095,
    "f": 250,
    "ffr": 184.132538,
    "fr": 96.864685,
    "r": 67.165421,
    "collision": False,
    "speed": 0,
    "turn_angle": 0
},
"action": "0000"
```

Obrázek 16 - ukázka nového formátu vstupních dat modelu

V rámci optimalizace bylo třeba pozměnit vstupní data modelu, která nebyla pro lepší výsledky jízdy dostatečná. Byly tak do hry přidány další 2 senzory pod úhlem 22,5°, díky nimž má klient lepší přehled o okolním prostředí, ukázka na obrázku 16. Ostatní parametry vozu jsou vyhovující, tak není třeba je měnit.

```
class ModelManagement():
    def create_model(verbose: bool = False) -> tf.keras.Model:
        verbose = True
        if verbose:
            print("creating model")

        input_shape = (10, )                    # 10 inputs of 8 bits

        base_model = Sequential([
            layers.Input(shape=input_shape),
            layers.Dense(128, activation='relu'),
            layers.Dense(32, activation='relu'),
            layers.Dense(4, activation='sigmoid')# sigmoid - 0 - 1, softmax: sum = 1
        ])

        optimizer = optimizers.Adam(learning_rate=0.0005)
        loss = losses.BinaryCrossentropy()
        metrics = ['accuracy']

        # compile model
        base_model.compile(optimizer=optimizer, loss=loss, metrics=metrics) #, run_

        return base_model
```

Obrázek 17 – ukázka nastavených parametrů modelu při tvorbě neuronové sítě

Při původní tvorbě modelu je vždy třeba zvolit původní parametry se bude model následně pracovat, jako jsou počty uzlů v jednotlivých vrstvách, počty samotných vrstev nebo různé optimalizační funkce. Tyto parametry nebylo třeba příliš měnit, hýbalo se s nimi jen pro možnosti testování. Jejich ukázku můžeme vidět na obrázku 17.

11.5.6 Test optimalizovaného klienta

Klient po přidání parametrů lépe jede po trati, přidaná informace nadcházející zatáčce o něco dříve zajistila, že klient nedělá tolik chyb při jízdě v přímém směru a ke kolizím nedochází tak často jako před optimalizací.

Dále je možné, aby nastala kolize s jiným autem, pro to je třeba počítat také pozici auta ovládaného klientem vzhledem k pozici ostatních aut. Na rozdíl od překážek jsou vozidla ostatních klientů pohyblivá, je třeba proto přidat jiné řešení.

Klient v tomto stavu dokáže projet část trati, stačí však, aby trénovací data obsahovala nějakou chybu, například kolizi s překážkou, kdy klient nemá data k tomu, aby zjistil, jak se ze situace dostat. V takových situacích se hodí modelu trochu vypomoci, a přidat mu například jednoduchou podmínku, ve které při nastalé kolizi klient autem chvíli couvá.

Jako další možnost, pro řešení problému s kolizí, se naskýtá přidání nějaké formy algoritmu zpětnovazebního učení, jako jsou v práci zmíněné algoritmy Q-Učení, SARSA, nebo vlastní implementace zpětnovazebního algoritmu.

11.5.7 Řešení kolizí

Pro vyřešení problému s kolizemi je implementován algoritmus, který při nastalé kolizi couvá po přednastavený čas a následně vrací rozhodovací kompetenci zpět již existující neuronové síti. Pokud se však s rozhodovacím algoritmem takto pracuje, je možné, že se ze situace nikdy nedostane, protože je klientské auto vraceno zpět do stavu, v jakém se klient rozhodl pro špatnou akci, a pro tuto akci se rozhodne znovu. Je tak dále třeba implementovat nějakou formu učení modelu.

11.5.8 Zhodnocení´

Aktuální model je trénován na trati o šířce 3 dlaždic, zvládá tak řešit situace na tratích o šířce 3 dlaždic, nebo je přechodně o něco širší či užší. Na tratích s širšími segmenty se ztrácí, protože data pro takovou trať nikdy neviděl. Pro schopnost klienta s aktuálním modelem řešit i situace s takovou tratí by tak bylo potřeba poskytnout mu více označených dat, a to takových, kde by se právě situace s širší tratí řešily. Dále by bylo pro řešení jízdy na širší trati vhodné prodloužit vzdálenost, na kterou vozidlo „vidí“.

Nejvíce limitujícím faktorem pro klienta v aktuálním stavu je však neznalost směru jízdy, pro pokračování ve vývoji a rozšíření implementace by tak bylo potřeba přidat právě tuto informaci. Problémem to může být ve chvíli, kdy klient naboural, následně si couvne a otočí se na trati, jízdu po trati v protisměru pak považuje za správnou, neboť nemá informaci o správnosti směru jízdy. V aktuální hře nebylo odesílání této informace implementováno z důvodu nenalezení jednoduchého způsobu jejího zjištění. Trať je tvořená uživatelem, který zadává pouze informace o jejím rozložení, ne o ideální ani středové linii trati.

[31]

12 UŽIVATELSKÁ DOKUMENTACE

Klientská aplikace nabízí pro běh jednoduché terminálové menu výčtem základních možností:

- Připojení ke hře a sběr dat z hráčova vozu
- Trénování modelu
- Připojení a ovládání klienta

Data je možné po připojení klienta do hry nasbírat, natrénovat na nich model a následně spustit klienta tak, aby se s připojením přidalo klientské auto a klient ho okolo trati sám řídil.Pro testování části práce s vlastní hrou je třeba spustit stáhnout a spustit engine Godot, ve verzi 3.5.3 a otevřít v něm projekt z přiloženého archivu. Hra se z prostředí herního enginu spustí jednoduše tlačítkem „přehrát“ v pravém horním rohu okna.

Pro editaci a prohlížení aplikace AI klienta je třeba mít na PC nainstalován Python a libovolný textový editor nebo lehké IDE, pro psaní práce byl využíván editor Microsoft VS Code ve verzi 1.85. Dále je třeba z přiloženého archivu rozbalit do libovolné složky, ve zvoleném editoru složku. Přes terminál nebo příkazovou řádku navigovat do něj a spustit příkazem „py main.py“ nebo „python main.py“, dle nastavení PC. Na obrázcích 18 a 19 jsou zobrazena hlavní menu pro každou z aplikací. Obě aplikace jsou nastaveny tak, aby bylo jejich ovládání co nejlpřehlednější, a v menu obou aplikací byla tak snaha o jednoduchost.

```
1 - collect data
2 - train model
3 - add and controll car
4 - connect and manually control
0 - exit
```

Obrázek 18 - menu klientské aplikace

Obrázek 19 - menu hry

13 ZÁVĚR

Hlavním cílem práce bylo ověření možností a složitosti implementace závodní videohry s pohledem z vrchu spolu s externím klientem, který měl hru na základě vybraných dat dokázat ovládat. Základní princip se ověřit podařilo a v dalším rozvoji práce je možné pokračovat. Pro další práci na projektu je ale dobré zhodnotit několik poznámek. Projekty je možné nadále vyvíjet, a to i odděleně, neboť na sobě přímo nezávisí, jedinou souvislostí mezi projekty je formát vstupních dat, které je možné přizpůsobit aktuální potřebě.

Hra dle zadání nabízí možnost uživatelské tvorby okruhů, je zpracována ve 2D prostoru s pohledem shora, a je možné k ní připojit více klientů, které přidají jednotlivá vozidla, na trati se tak v jednu chvíli může nacházet i více než 2 auta.

Klientská aplikace při ovládání auta úspěšně manévruje po trati a ve většině případů zvládá trať vcelku přirozeně projet. Pro ovládání vozu se ukázal systém sběru dat pomocí metody ray-castingu jako vyhovující. Naopak vytvoření takového klienta, který by dokázal zajistit hráči kompetitivní závod se ukázalo být složité a tato část zadání tak nebyla naplněna.

Největší problém při vývoji práce nastal při zjištění pomalého a nepřesného odpovídání z ANN klienta, to se však vyřešilo výrazným zmenšením vstupního vektoru. Práce tak může být dále rozvíjena přidáním nějaké formy učení ANN, a to ať už formou předepsané knihovny, nebo formou vlastního algoritmu učení v reálném čase.

Pro další vylepšení chování modelu by bylo do hry nutné implementovat libovolnou formu kontroly směru jízdy po trati, díky které bude možné kontrolovat směr jízdy vozu po trati a zabrání se tak nevyžádanému otáčení do špatného směru.

Co se týká dalšího rozvoje hry samotné, nabízí se například: přesun AI klienta z externí aplikace do GDNative/GDExtension knihovny, předělání herní grafiky a doplnění mnoha funkcionalit, jako je například přidání online multiplayeru pro nějž je hra díky komunikaci s klintem přes protokol WebSocket připravena. Díky multiplayeru by hra nabízela zajímavou funkcionalitu a její vydání by dávalo smysl i z hlediska uživatelské základny a zpětné vazby na provedené změny.

Vývoj hry začal v době, kdy poslední vydanou verzí herního enginu Godot byla verze 3.5.2. Pro další pokračování práce na projektu by tak bylo vhodné aktualizovat na některou z aktuálních verzí 4.x.x. Verze je však nekompatibilní s aktuálně použitým kódem.

Práce může sloužit jako základní seznámení se s videohrami a jejich vývojem v herním enginu Godot, nebo různými metodami strojového učení. Za zmínku pak stojí také implementace hluboké neuronové sítě s použitím Keras a TensorFlow v jazyce Python.

POUŽITÁ LITERATURA

[1] METUARAU, Tuakana. *A History of Video Games* [online]. B.m.: Open Access Te Herenga Waka-Victoria University of Wellington. 1. leden 2017 [vid. 2024-05-09]. Dostupné z: doi:10.26686/wgtn.17059826

[2] WOLF, Mark JP. *The video game explosion: a history from PONG to Playstation and beyond*. Bloomsbury Publishing USA, 2007. ISBN 031308243X.

[3] Zenva. What is Unity? – A Top Game Engine for Video Games. *Game Dev Academy* [online]. 2024 [cit. 2024-05-09]. Dostupné z: https://gamedevacademy.org/what-is-unity/

[4] *Unity documentation* [online]. 2024 [cit. 2024-05-09]. Dostupné z: https://docs.unity.com/

[5] GREGORY, Jason. *Game Engine Architecture, Third Edition* [online]. 3rd ed. A K Peters/CRC Press, 2018 [cit. 2024-05-09]. ISBN 9781315267845. Dostupné z: doi:10.1201/9781315267845

[6] Arithmetic Operators in LISP. *Geeks for geeks* [online]. 2024 [cit. 2024-05-09]. Dostupné z: https://www.geeksforgeeks.org/arithmetic-operators-in-lisp/

[7] TECUCI, Gheorghe. Artificial intelligence. WIREs Computational Statistics [online]. 2012, 4(2), 168-180 [cit. 2024-05-09]. ISSN 1939-5108. Dostupné z: doi:10.1002/wics.200

[8] *Godot docs* [online]. 2024 [cit. 2024-05-09]. Dostupné z: https://docs.godotengine.org/

[9] GRUNITZ, Mario. Rule-based AI vs machine learning: what's the difference? *We Are Brain* [online]. 2024 [cit. 2024-05-09]. Dostupné z: https://wearebrain.com/blog/rule-based-ai-vs-machine-learning-whats-the-difference

[10] GROSAN, Crina, Ajith ABRAHAM, Crina GROSAN a Ajith ABRAHAM. Rule-Based Expert Systems. In: *Intelligent Systems* [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, s. 149-185 [cit. 2024-05-09]. Intelligent Systems Reference Library. ISBN 978-3-642-21003-7. Dostupné z: doi:10.1007/978-3-642-21004-4_7

[11] EL NAQA, Issam a Martin J. MURPHY. What Is Machine Learning? In: EL NAQA, Issam, Ruijiang LI a Martin J. MURPHY, ed. *Machine Learning in Radiation*

Oncology [online]. Cham: Springer International Publishing, 2015, s. 3-11 [cit. 2024-05-09]. ISBN 978-3-319-18304-6. Dostupné z: doi:10.1007/978-3-319-18305-3_1

[12] MAHESH, Batta. *Machine Learning Algorithms -A Review*. 2019/01/01. Dostupné z: doi:10.21275/ART20203995

[13] CUNNINGHAM, Pádraig, Matthieu CORD a Sarah Jane DELANY. Supervised Learning. In: CORD, Matthieu a Pádraig CUNNINGHAM, ed. Machine Learning Techniques for Multimedia [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, s. 21-49 [cit. 2024-05-09]. Cognitive Technologies. ISBN 978-3-540-75170-0. Dostupné z: doi:10.1007/978-3-540-75171-7_2

[14] Semi-Supervised Learning, Explained with Examples. *Altexsoft* [online]. 2024, 2024-03-29 [cit. 2024-05-09]. Dostupné z: https://www.altexsoft.com/blog/semi-supervised-learning/

[15] Types of Machine Learning. *Java T point* [online]. 2024, 2024-03-29 [cit. 2024-05-09]. Dostupné z: https://www.javatpoint.com/types-of-machine-learning

[16] SYDENHAM, Peter H. a THORN, Richard (ed.). *Handbook of Measuring System Design*. Online. Wiley, 2005. ISBN 9780470021439. Dostupné z: https://doi.org/10.1002/0471497398. [cit. 2024-05-09].

[17] PRIDDY, Kevin L. a Paul E. KELLER. *Artificial Neural Networks: An Introduction* [online]. SPIE, 2005 [cit. 2024-05-09]. ISBN 9780819478726. Dostupné z: doi:10.1117/3.633187

[18] What is a neural network? *IMB* [online]. 2024 [cit. 2024-05-09]. Dostupné z: https://www.ibm.com/topics/neural-networks

[19] OBERHUBER, Tomáš. *Neuronové sítě*. Faculty of Nuclear Sciences and Physical Engineering Czech Technical University in Prague, 2024. Dostupné také z: https://geraldine.fjfi.cvut.cz/~oberhuber/data/vyuka/aom/04-neuronove-site.pdf

[20] Linear Regression in Machine Learning. *Java T point* [online]. 2024, 2024-03-29 [cit. 2024-05-09]. Dostupné z: https://www.javatpoint.com/linear-regression-in-machine-learning

[21] K-Means Clustering Algorithm. Java T point [online]. 2024, 2024-03-29 [cit. 2024-05-09]. Dostupné z: https://www.javatpoint.com/k-means-clustering-algorithm-in-machine-learning

[22] Support Vector Machine — Introduction to Machine Learning Algorithms. *Towards Datascience* [online]. 2018 [cit. 2024-05-09]. Dostupné z: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[23] Gentle Introduction to the Adam Optimization Algorithm for Deep Learning. *Machine Learning Mastery* [online]. 2018 [cit. 2024-05-09]. Dostupné z: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

[24] LI, Yuxi. DEEP REINFORCEMENT LEARNING: AN OVERVIEW [online]. 2018. 2018. Dostupné také z: https://arxiv.org/pdf/1701.07274

[25] SAINI, Anshul. Guide on Support Vector Machine (SVM) Algorithm. *Analytics Vidhya* [online]. 2024 [cit. 2024-05-09]. Dostupné z: https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/

[26] Raju, Nitin SIKKA, Sanjeev KUMAR a Rahul GUPTA. Artificial Intelligence in Games. International Journal of Soft Computing and Engineering (IJSCE) [online]. Blue Eyes Intelligence Engineering & Sciences Publication, 2012, 2012(2 / 5), 269 - 273 [cit. 2024-05-09]. ISSN 2231-2307,. Dostupné z: https://www.ijsce.org/wpcontent/uploads/papers/v2i5/E1050102512.pdf

[27] FILIPOVIĆ, Aleksandar. THE ROLE OF ARTIFICIAL INTELLIGENCE IN VIDEO GAME DEVELOPMENT. KULTURA POLISA [online]. 2023, 2023-11-08, 20(3), 50-67 [cit. 2024-05-09]. ISSN 2812-9466. Dostupné z: doi:10.51738/Kpolisa2023.20.3r.50f

[28] Machine Learning Models. JavaTpoint [online]. 2021 [cit. 2024-05-09]. Dostupné z: https://www.javatpoint.com/machine-learning-models

[29] BARTÁK, Ondřej. DEEPLY. Neuronové sítě: Vaše tajná zbraň pro úspěch v podnikání!. DEEPLY AI SOFTWARE, S.R.O. Deeply.cz [online]. [cit. 2024-05-09]. Dostupné z: https://deeply.cz/blog/umele-neuronove-site

[30] UNIVERZITA KARLOVA. . UNIVERZITA KARLOVA. Computer Graphics Group [online]. 2024 [cit. 2024-05-09]. Dostupné z: https://cgg.mff.cuni.cz/~pepca/prg022/mucha/

[31] TENSORFLOW. *Tensorflow* [online]. 2015 [cit. 2024-05-09]. Dostupné z: https://www.tensorflow.org/

[32] Sutton smazat

[33] GEEKSFORGEEKS. *Deep Q-Learning*. GEEKSFORGEEKS. GeeksforGeeks [online]. 2008 [cit. 2024-05-09]. Dostupné z: https://www.geeksforgeeks.org/deep-q-learning

[34] LEK, S. a Y.S. PARK. *Artificial Neural Networks*. In: . Oxford: Academic Press, 2008, s. 237-245. ISBN 978-0-08-045405-4. Dostupné z: doi:https://doi.org/10.1016/B978-008045405-4.00173-7

[35] AL-MASRI, Anas. BUILTIN. *How Does Backpropagation in a Neural Network Work?* Builtin.com [online]. 2024, 7. 3. 2024 [cit. 2024-05-09]. Dostupné z: https://builtin.com/machine-learning/backpropagation-neural-network

[36] SUZUKI, Kenji. *Artificial Neural Networks - Methodological Advances and Biomedical Applications*. 2011. Janeza Trdine 9, 51000 Rijeka, Croatia: InTech, 2011/04/11. ISBN 978-953-307-243-2. [cit. 2024-05-09]

[37] SAEZ, Yago; PEREZ, Diego; SANJUAN, Oscar a ISASI, Pedro. *Driving Cars by Means of Genetic Algorithms*. Online. In: RUDOLPH, Günter; JANSEN, Thomas; BEUME, Nicola; LUCAS, Simon a POLONI, Carlo (ed.). Parallel Problem Solving from Nature – PPSN X. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, s. 1101-1110. ISBN 978-3-540-87699-1. Dostupné z: https://doi.org/10.1007/978-3-540-87700-4_109. [cit. 2024-05-09].

[38] SHAKYA, Amit Raj, Kushal Sagar SHRESTHA, Manish SHRESTHA a Nikesh SUBEDI. Simulation-based Self-driving Car using Reinforcement Learning [online]. Lalitpur, Nepal, 2021 [cit. 2024-05-09]. Dostupné z: https://kec.edu.np/wp-content/uploads/2021/09/Paper-46-Simulation-based-Self-driving-Car-using-Reinforcement-Learning.pdf. Výzkumná. Kantipur Engineering College, Department of Computer and Electronics Engineering.

[39] CHAN, Marvin T.; CHAN, Christine W. a GELOWITZ, Craig. Development of a Car Racing Simulator Game Using Artificial Intelligence Techniques. Online. International Journal of Computer Games Technology. 2015, roč. 2015, s. 1-6. ISSN 1687-7047. Dostupné z: https://doi.org/10.1155/2015/839721. [cit. 2024-05-09].