BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

DEEP LEARNING METHODS FOR MACHINE PLAYING THE SCOTLAND YARD BOARD GAME

METODY HLUBOKÉHO UČENÍ PRO STROJOVÉ HRANÍ HRY SCOTLAND YARD

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **ZUZANA HRKĽOVÁ**
AUTOR PRÁCE

SUPERVISOR **doc. Ing. FRANTIŠEK ZBOŘIL, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

144325

Ústav: Ústav inteligentních systémů (UITS)
Studentka: **Hrkľová Zuzana**
Program: Informační technologie
Specializace: Informační technologie

Název: **Metody hlubokého učení pro strojové hraní hry Scotland Yard**
Kategorie: Umělá inteligence
Akademický rok: 2022/23

Zadání:

1. Seznamte se s pravidly deskových her, kde aktuální stav hry bývá pro hráče po většinu času utajený. Inspirujte se hrou "Scotland Yard", kdy pozice jedné z figur bývá protihráčům ukázána jen v některých kolech hry.
2. Navrhněte obdobnou hru, ve které herní pole může být oproti hře Scotland Yard zjednodušené, ale zachovejte princip skrývání figury ve většině tahů.
3. Nalezněte způsob využití metod hlubokého učení zahrnující neuronové sítě, který by zvýšil schopnost stroje hrát hru určenou v předchozím bodě.
4. Pro jednotlivé role figur ve hře implementujte algoritmy řízení zahrnující tyto metody a ověřte jejich schopnost hrát danou hru.
5. Vyhodnoťte úspěšnost obou stran hry pro různé míry zapojení metod strojového učení a diskutujte zjištěné výsledky.

Literatura:

1. Dash, T., Dambekodi, S.N., Reddy, P.N. *et al.* Adversarial neural networks for playing hide-and-search board game Scotland Yard Neural Comput & Applic 32, 3149–3164, 2020
2. Russel, S., Norvig, P.: Artificial Intelligence, A Modern Approach, Pearson, 2009
3. J.P.A.M. NijssenMark H.M. WinandsMark H.M. Winands: Monte-Carlo Tree Search for the Game of Scotland Yard,Computational Intelligence and Games (CIG), 2011

Při obhajobě semestrální části projektu je požadováno:
První dva body zadání

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Zbořil František, doc. Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1.11.2022
Termín pro odevzdání: 10.5.2023
Datum schválení: 3.11.2022

Abstract

This theses concerns with deep learning methods applied to machine playing board games containing movement uncertainty. Reinforcement learning principles with main focus on Q-learning algorithms were studied, among which Deep Q–Network had been chosen and applied on simplified rules of the Scotland Yard board game. The final implementation was put to test against Alpha-Beta and Monte Carlo Tree Search. The results have shown that the hider driven by DQN represented the hardest opponent for the other two methods, while the DQN seekers did not manage to surpass past results. Although the implemented method did not reach better results than currently known methods, it proved to be the least demanding when considering computational resources and time needed to perform a given move, making it the most perspective to implement on original version of the game in the future.

Abstrakt

Táto práca sa zaoberá metódami hlbokého učenia, ktoré sú aplikovateľné na stolné hry s neurčitosťou. V rámci práce boli naštudované princípy učenia s posilňovaním, s hlavným zameraním na Q-learning algoritmy, spomedzi ktorých bol vybraný Deep Q-Network algoritmus. Ten bol následne implementovaný na zjednodušených pravidlách stolnej hry Scotland Yard. Konečná implementácia bola porovnaná s metódami Alpha-Beta a Monte Carlo Tree Search. S výsledkov vyplinulo, že schovávaný hráč riadený DQN algoritmom predstavoval pre ostatné metódy najťažšieho protihráča, narozdiel od hľadajúcich hráčov, ktorým sa nepodarilo zlepšiť existujúce riešenia. Napriek tomu, že implementovaná metóda nedosiahla lepšie výsledky oproti doposiaľ existujúcim metódam, ukázalo sa, že potrebuje najmenej výpočetných zdrojov a času na vykonanie daného ťahu. To ju robí najperspektívnejšou zo spomínaných metód na budúcu možnú implementáciu originálnej verzie danej hry.

Keywords

machine learning, deep learning, neural networks, reinforcement learning, Q-learning, DQN, board games, strategy games, games with uncertainty, Scotland Yard

Klíčová slova

strojové učenie, hlboké učenie, neurónové siete, posilované učenie, Q-learning, DQN, stolné hry, strategické hry, hry z neurčitosťou, Scotland Yard

Reference

HRKĽOVÁ, Zuzana. *Deep Learning Methods for Machine Playing the Scotland Yard Board Game*. Brno, 2023. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor doc. Ing. František Zbořil, Ph.D.

Rozšířený abstrakt

Úvod

Strojové hranie hier je na scéne už dlhšiu dobu. V tomto odbore bolo dosiahnutých hneď niekoľko míľnikov. Zatiaľ čo viac očivindým zámerom vyvíjania takto hrateľných hier je redukovanie potreby fyzického spoluhráča, tento prežitok nie je jediným účelom tvorby takýchto systémov. Jedným z príkladov je fakt, že ak je možné naučiť umelú inteligenciu hranie hry na schovávačku, potom je možné tieto vedomosti využiť pri tvorbe systému schopného detekcie mín na mínovom poly.

Hlboké učenie otvára dvere novým riešeniam pre strojové hranie hier. Pomocou neurónových sietí sa doposiaľ podarilo poriaziť hneď niekoľko expertov v daných hrách, čo bolo inšpiráciou k vytvoreniu práce. Účelom tejto práce bolo narvhnúť a implementovať metódu hlbokého učenia pre kooperatívnu, strategickú stolnú hru Scotlad Yard a následne porovnať dosiahnuté výsledky s implementáciami metód Alpha-Beta a Monte Carlo Tree Search. Po naštudovaní problematiky a potrebných zdrojov, došlo k záveru zvoliť metódu Deep Q-learning, ktorá bola implementovaná za účelom strojového hrania zjednodušenej verzie hry Scotland Yard.

Popis navrhnutého riešenia

Z dvôvodu, že Scotland Yard je asymetrickou hrou a teda sú agenti poverení inou úlohou ako pán X, boli navrhnuté dva rozdielne modely, ktoré implementujú jednotlivé chovania hráčov. Agenti síce nie sú schopní vidieť presnú polohu pána X, pokiaľ ju práve v tom kole neodhalí, no na vstupe neurónovej siete dostávajú vyznačené políčka s percentuálnou možnosťou výskytu pána X. Agenti sú potom odmenení zakaždým keď počas tréningu na takéto poličko stúpia, za účelom naučiť ich hľadať pána X práve na nich.

Pán X je naopak odmenený za kažké kolo ktoré pretrvá bez toho, aby bol chytený agentami. Experimenty ukázali, že pán X podáva najlepšie výsledky keď počas tréningu súperí proti agentom implementovaným oboma metódami a teda sa nenaučí len slepo vyhýbať jednej z metód odhalením jej stratégie.

Ako pogramovací jazyk danej práce bol zvolený Python za účelom využitia hracieho prostredia implementovaného v tomto jazyku v rámci bakalárskej práce M. Sovu zo zámerom jednoduchšieho porovnania výkonnosti jednodlvých metód proti sebe.

Experimenty

Pre zvolený algoritmus bolo navrhnutých niekoľko experimentov s cieľom nastaviť jeho parametre tak, aby dosahoval čo najlepšie možné výsledky. Podrobnejšie opísané experimenty v tejto práci opisujú proces výberu architektúry neurónovej siete využívanej algoritmom Deep Q-learning. Rovnaké architektúry boli testované na schovávajúcom pánovi X aj na hľadaúcich agentoch. Experimenty odhalili, že achritektúra schopná natrénovať najlepší možný model pre pána X nebola zhodná s najvýkonnejšou architektúrovu využívanou agentami. Ďalšie experimenty sa zamerali na výber metódy riadiacej pohyby protihráčov počas tréningu za účelom dosiahnuť čo možno najviac univerzálny model zvládajúci rovnako dobre poraziť obe metódy. Experimenty potvrdili mienku, že takýto model je možné získať využitím náhodného výberu medzi oboma metódami. Posledný z rady experimentov

sa zameral na spôsom odmeňovania agentov za ich akcie, kde došlo k záveru, že najlepšou stratégiou je odmeňovanie agenta len za jeho vlastné akcie a teda dostávať väčšiu odmenu len keď dojde k výhre jeho dočinením.

Zhrnutie výsledkov a budúca práca

Pomocou série spomínaných experimentov sa podarilo získať optimálne parametre pre trénovanie daných modelov ako aj najprospernejšie architektúry sietí a odmeňovacie funkcie. Výsledkom boli dva rôzne modely jeden využívaný pánom X a druhý agentami. Tie boli niekoľkokrát pretrénované a ešte raz naposledy otestované proti metódam Alpha-Beta a Monte Carlo Tree Search. Výsledky ukázali, že model natrénovaný pomocou DQN algoritmu v roly pána X predstavoval najťažšieho oponenta pre obe zo spomínaných metód. DQN agenti nedosiahli rovnako pozitívne výsledky. Po pozorovaní štýlu ich hry však bolo viditeľné, že sa naučili stúpať na políčka hracej plochy o ktorých vedeli, že majú nejakú percentálnu šancu výskytu pána X. To, že angenti pochopili koncept danej hry sa preukázalo aj faktom, že sa im darilo vyhrať 98,60% hier proti náhodnému pánovi X.

Výkonnosť agentov by sa v budúcnosti mohla zlepšiť využitím konvolučných sietí. Zároveň by sa už existujúca implementácia DQN mohla rozšíriť na implementáciu algoritmu DDQN spolu s metódou skoršieho zastavenia tréningu, aby sa vyhlo pretrénovaniu modelu hráčov.

Deep Learning Methods for Machine Playing the Scotland Yard Board Game

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of doc. Ing. František Zbořil, Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Zuzana HRKĽOVÁ
May 9, 2023

Acknowledgements

Major thanks to my work supervisor who helped this work come to fruition. Thank you, doc. Ing. František Zbořil, for your constant support, valuable advice, time and strong guidance.

Contents

1 Introduction **2**

2 Hidden movement board games **4**
2.1 Hidden movement 4
2.2 Specific board games with movement uncertainty 4

3 The Scotland Yard board game **9**
3.1 The Scotland Yard rules 9
3.2 Game strategy 11
3.3 Current machine playing engines for the game of Scotland Yard 11
3.4 Simplified rules of Scotland Yard board game 12

4 Reinforcement learning **14**
4.1 Q–Learning 14
4.2 Deep Q–Learning 16
4.3 Double Q–Learning 16
4.4 Double Deep Q–Network 17

5 Deep Q-Learning implementation of Scotland Yard **18**
5.1 Used technologies 18
5.2 DQN implementation 18
5.3 Neural Network architecture 19
5.4 Rewarding system 22
5.5 Training parameters 23

6 Experiments **24**
6.1 Experimenting on Mr.X's network 24
6.2 Experimenting on detective's network 26
6.3 Results 28
6.4 Suggestions for future improvement 29

7 Conclusion **30**

Bibliography **32**

A Contents of the included storage media **35**

Chapter 1

Introduction

Nowadays, teaching artificial intelligence to play games is not anything ground-breaking. When successfully teaching an AI to play a certain game, by playing against a computer, one can train themselves to improve their skills or simply enjoy playing the game without the need of human opponent. However, creating a human–like system is not an easy task to master. Many decision making methods might be efficient in game–play, but easily mastered by a human player after learning its strategy, thus becoming repetitive. The goal is to create such an opponent that will stay equally challenging to a human player even after a certain amount of games played.

Although it might not seem like it at first, reducing the need of human opponent is not the only positive outcome. It also helps with discovering boundaries of technology. When addressing real–life problem solving done by AI, games can often be used as a starter point in early stages of testing the possibilities.

While successfully teaching AI a simple game of hide–and–seek might not seem as significant, it could very well be the first stage of creating an AI able to detect and disarm landmines.

This work is discussing deep learning methods for machine playing Scotland Yard, which is a cooperative strategic board game. While choosing suitable deep learning method for the given game, factors such as uncertainty and ability for agents to cooperate needed to be taken into consideration. After thorough consideration Deep Q–Network has been chosen as a method of interest.

Several bachelor thesis have been done at Brno University of Technology concerning different methods for machine playing the Scotland Yard board game, namely Alpha-Beta and Monte Carlo Tree Search, results of which are compared with the Deep Q–Network approach.

While Alpha-Beta does have the best results when compared with the two remaining methods, its outcome is also the most predictable. Using Deep Q–Network might not have as good results compared to the two remaining methods on the other hand it is significantly faster and less demanding on computational resources thus making it easier to expand on a full–sized board, and in future potentially create original version of the game as all the experiments in this paper were performed on a simplified version of this game.

The following chapter describes occurrence of hidden movement in board games while depicting it on specific examples. Thesis then continues with description of Scotland Yard's rules, strategy and currently existing techniques used to implement machine playing of this game. The reader is then presented with simplified rules of the Scotland Yard, which have been used in this thesis's implementation of DQN. Reinforcement algorithms with

main focus on Q-learning are carefully described in the following chapter. The thesis then progresses with description of the final DQN implementation resulting from experiments described in next chapter. The conclusion derived from presented experiments and comparison of DQN with other two mentioned methods is followed by suggested improvements for future work.

Chapter 2

Hidden movement board games

This chapter discusses what exactly hidden movement is and how imperfect information affects autonomous artificial intelligence made to play board games. Reader is presented with different types of board games containing movement uncertainty.

2.1 Hidden movement

As the name suggests, hidden movement occurs in a game when a movement or a certain action of a player is hidden from their opponent, thus putting opponent in a position of imperfect information about the move just made.

Hidden movement introduces uncertainty to the game. The player cannot tell with 100 percent accuracy in which direction their opponent moved or whether a certain action has been done. In order to win, without knowing the full extent of what happened in the opponent's move, player needs to use their deduction skills to narrow down opponent's possible location and adapt a strategy to increase their chance of winning. Therefore games with hidden movement are categorized as strategy games.

When creating autonomous artificial intelligence created to play a certain board game, decision–making under imperfect information represents one of the biggest challenges in the process. Some methods implement the best–move strategy. Increased complexity of the game however increases computational time for the next move as well as demand on computational resources resulting from increased number of states game which could end up making it harder to find the best possible move. Partially solving this issue can be done by simplifying game rules, which can however lead to lower interest of game enthusiasts to play a given game. Moreover, the best-move strategy can quickly get repetitive for the human player as they are able to predict what will the AI driven engine do in the current or even next move. The game–play misses a small amount of uncertainty of a human opponent, such as whether or not they will make a specific action.

2.2 Specific board games with movement uncertainty

In this section there are three different game descriptions, each of them carefully chosen from a slightly different type of hidden movement games.

Stratego has been chosen as a representative of a group of games where the placement of player's figures on the board is not hidden, however the identity of the figures is. Movement

uncertainty is thus present in the game not through the hidden direction of the moved figure but by not knowing which figure has been moved.

Nuns On The Run is a classical hide–and–seek game where positions of all players, except the one who is in a role of seeker, are hidden. Lastly the Fury of Dracula together with Scotland Yard are the reverse hide–and–seek games where location of one player is hidden and the objective of other players is to find the hidden player.

Each of the game descriptions is divided into three parts. Firstly, the reader is presented with brief rules overview to show what the game is about, followed by explanation of common strategies for the given games, finished by the summary of current state concerning the AI playing the given games.

Stratego

Stratego is a two–payer game consisting of a ten by ten board and 80 ranked game pieces in total. At the beginning of the game, each of the players gets 40 pieces and strategically places them on their half of the board, labeled side facing the player, leaving the middle two rows of the board empty. Players then alternate turns.

Each player can decide whether they are going to move one of their pieces or attack. When chosen to move a piece, they can place it one tile up, down, left or right. In case of attacking, player's piece needs to stand adjacent space to opponent's piece. The two pieces reveal their rank and the piece with higher rank wins the attack and captures opponent's piece by removing it from the board and taking its position. Each player at the beginning of the game possesses 7 immovable pieces, 6 of them being bombs that cannot attack any of the opponent's pieces, but in the case of opponent's attack they win and remove opponent's piece from the board. The remaining immovable piece is a flag piece.

The objective of the game is to capture opponent's flag piece. Once any of the players completes the objective, game ends and the player wins. In case a player cannot move any of their remaining pieces on the board or cannot attack any of the opponent's pieces, the game ends and they loose.

Each of the players possesses 3 types of special movement characters: 8 Scouts, 5 Miners and one Spy. Scouts enable the player to move more than 1 piece across the board as well as attack in the same turn. When a Miner attacks opponent's bomb piece, the bomb is diffused and removed from the board. Last but not least, Spy is the only piece that can win an attack against Marshal, piece with the highest rank. However, when Spy is attacked by any of the opponent's pieces, he looses, and gets removed from the board.

Full version of the rules along with strategy hints can be found here[19].

Figure 2.1: **Stratego game board displaying placement of the figures at the beginning of the game** Source:[1]

By not seeing the placement of opponent's pieces, game is filled with uncertainty and hidden moves as none of the players know which piece their opponent has just moved. Strategy needs to be established when placing the pieces on the board at the beginning of the game. Deduction skills are also a big part of the game. The player tries to narrow down which of the opponent's pieces could be the bombs and the flag by keeping track of all of the pieces the opponent has not moved throughout the game. The players also try to deduce what type of opponent's pieces are still on the board by knowing which ones have been removed and adjust the game–play accordingly.

Concerning AI playing Stratego, there have been multiple attempts to solve this issue, resulting in solutions that were weak or mediocre at best. Only recently DeepMind released a paper[24] on a new autonomous agent called „DeepNash" that learned to play the game Stratego at the level of a human expert.

Fury of Dracula

Fury of Dracula is a cooperative 2 to 5 player board game. Since it was first designed by Stephen Hand in 1987 the game evolved and there are currently 4 different editions. While the rules vary slightly in each edition, they all keep the same concept of the original game. The rules are as follows:

One of the players takes up a role of Dracula. All the remaining players are teamed up against him as hunters, each of them having one special ability. In case only two players engage in the game, one of the players plays for all four hunters in the game. Movement of Dracula is hidden from hunters and his position is revealed only when one of the hunters steps on the location Dracula currently occupies. During the day hunters can perform actions to prepare for taking down Dracula by moving around and obtaining items. When the night comes Dracula moves and sets up traps. The main objective of the hunters is to find and fight Dracula. Dracula's objective is to advance the influence track from zero to thirteen by creating vampires and defeating hunters. Once hunters or Dracula complete their objective, the game ends.

Full version of the rules for the fourth edition can be found here[13].

Figure 2.2: **The Fury of Dracula game board** Source:[31]

Besides the usual challenge of hidden movement games, the Fury of Dracula has fairly elaborate rules making it harder for players to master. The key strategy of hunters to win the game is to cooperate with the rest of the team. Hunters are able to block more paths by spreading out on the board, leaving Dracula with less options of locations to move to. Another good strategy is trying to acquire items early in the game while teaming up with the other hunters when attacking Dracula instead of striking alone once they track him down.

On the other hand Dracula aims for leaving behind as little clues about his location as possible and attacking hunters when they are by themselves not able to get help from another hunter.

While the game itself is not that young it has not risen much interest in the field of game theory. In 2020 Nomad Games Limited[10] published digital version of the game along with the option of a single player to compete against AI. However the AI received negative criticism from most of the players.

Nuns On The Run

Nuns on the Run is a 2 to 8 player game filled with hidden movement. Game consists of 2 guards and up to 6 novices. In case of 7 or less players, one of the players plays for both guards. Unlike in Scotland Yard and Fury of Dracula where all the players are trying to catch the same player here it is the other way around. Locations of all players taking up the role of a novice are hidden as well as their movement and the guards are trying to catch them.

At the beginning of the game all the players are placed on their starting positions marked down on the board. Every novice obtains a secret wish card with a location of a key and an object she wishes to get. Their goal is to first get to the key then continue to the object they wish to get according to the card and safely return to their starting position on the board, all that without getting caught by one of the guards. When moving, players need to show a special card revealing how many places they moved on the board. The

movement card together with throwing a dice determine whether or not the guards heard the novice. In case a novice is heard she needs to mark on the board which direction is the sound coming from.

Person playing the guards obtains card with a path marked down for each of the guards. When moving guards need to follow the path and the only time they can change direction is when they hear or see a novice. The novice is seen if she happens to stand or pass by the guard's field of view. Once the novice is caught she needs to return to her starting point leaving the wished object behind and try to obtain it again.

The game ends and the guards win if novices do not succeed in getting their wished objects by the fifteenth round or the guards catch a novice as many times as there are novices in the ongoing game. A novice wins if she returns to her starting point on the board with the wished item.

Official rules of the game can be found here[21]

Figure 2.3: **The game board of Nuns On The Run showing all novice players standing on their starting positions** Source:[9]

Nuns on the run is one of the easier hide–and–seek games which does not require complex strategy. The fact that most of the players are hidden makes it easier for them to distract the guards and run away without getting caught. In addition the cooperation of all the players is not as crucial as in Scotland Yard or Fury of Dracula. The best strategy for the novice players is to move as far as they can in the first rounds, making it harder for the guard player to determine direction of their movement. The rest is about correctly deducing where the guard's paths lead and to be able to avoid them in order to not get seen.

The role of a guard requires some type of cooperation, however by having predetermined paths it turns out to be very limited. Guards not being able to move freely can hardly divide their forces or try to surround any of the novice players.

Despite having multiple digital versions on several platforms, none of them supports the option of playing the game against AI.

Chapter 3

The Scotland Yard board game

Scotland Yard is a cooperative, asymmetric strategy board game created by the German game designer Werner Schlegelfirst. The game was first published and introduced on the market by Ravensburger in 1983 and in the same year won one of the most prestigious card and board game awards, Spiel des Jahres (Game of the Year). Several aspects of the games are considered when competing in Spiel des Jahres such as gameplay, game design, and overall presentation.

Scotland Yard was an innovative game in its early days for bringing the concept of hide–and–seek on the board as well as including cooperation of the players in the game. Although not being the first ever created hide–and–seek game, it helped to popularise the genre and was an inspiration to many other well–known games such as Clue The Great Museum Caper, Fury of Dracula or Letters from Whitechapel.

Since the original version turned out to be a huge success many alternative versions have been published. The versions vary in board designs featuring different cities or slight rule modifications.

Next section explains the rules of the original version. Later in the chapter basic game strategy is described followed by a section concerned with currently known implementations of artificial intelligence playing the Scotland Yard board game. Last but not least, the reader is introduced to simplified rules of Scotland Yard used later on in the paper for own game implementation of AI playing Scotland Yard.

3.1 The Scotland Yard rules

The game Scotland Yard is named after Scotland Yard, the headquarters of the Metropolitan Police and its board depicts map of London. Mister X is a criminal who is trying to hide and the Scotland Yard's detectives are trying to find him within the streets of London.

Game's board consists of 199 interconnected spaces representing public transport stations. There are 3 different types of connections between stations differentiated by colors according to the type of transport they represent, yellow used for taxis, green for buses, red for undergrounds and black for boat routes.

The game is played by 3 to 6 players where 1 player takes up a role of Mr. X and the remaining players are detectives. Players are taking turns, starting with Mr. X. The goal of detectives is to cooperate and catch Mr. X within 24 moves. If they will not succeed by then, Mr. X wins the game.

At the beginning of the game each of the detectives gets 10 taxi, 8 bus and 4 underground tickets. The detectives use the tickets to move around the board. Once they have no remaining transport ticket to travel from the station they occupy, they need to stand there until the end of the game.

Location of Mr. X is hidden and is revealed periodically throughout the game on the rounds 3, 8, 13, 18 and 24. While the detectives cannot see the exact location of Mr. X, they see the type of transport he used to move around the board. At the beginning of the game, Mr. X gets 4 taxi, 2 bus and 3 underground tickets along with the 2 double move tickets and as many black tickets as there are detectives in the game. In addition every time detectives move, the ticket they use from that point on belongs to Mr. X. Mr. X can use black tickets to travel by any type of transport on the board including the boat and thus giving the agents no clue whatsoever of where he went. Once Mr.X uses a ticket it is removed from the game.

Initial placement of each of the player's figures is given by randomly choosing a location card with the number of a given station. Players then take turns. Each turn starts with Mr.X and is followed by all the detectives. If Mr.X is supposed to be seen in the given round, he finishes his turn by moving to a chosen location and then reveals his whereabouts by placing Mr.X figure on the occupied station.

Full version of the rules can be studied even further here[4]

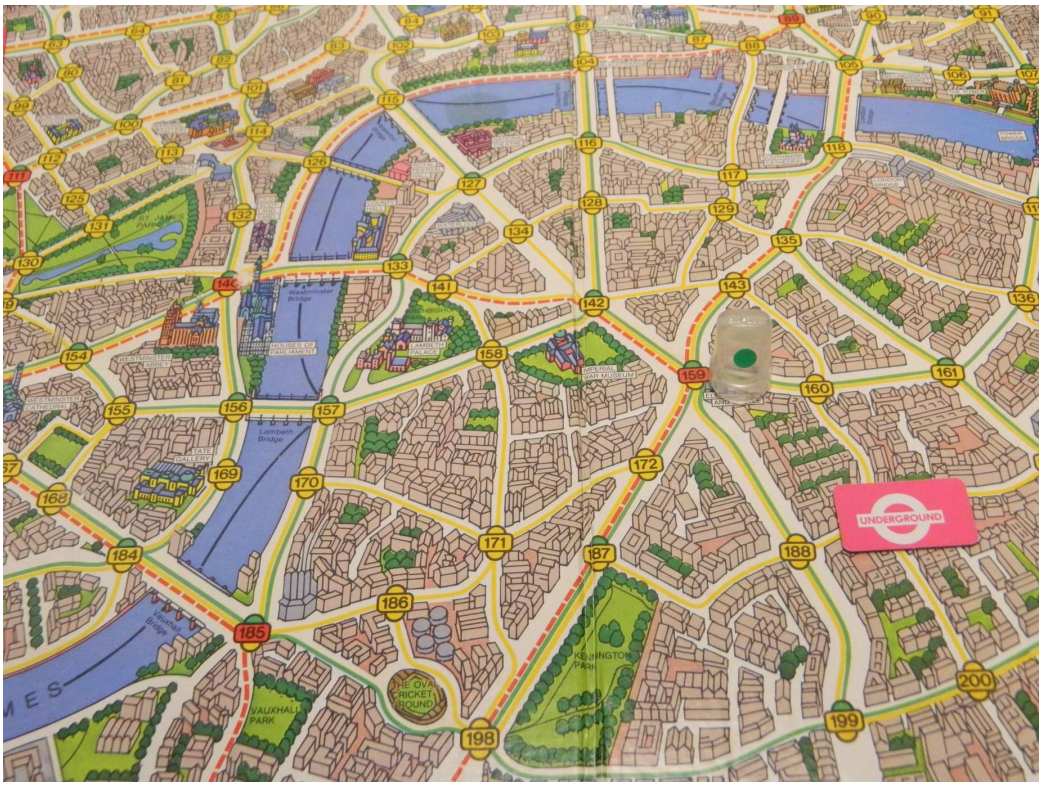

Figure 3.1: **Depiction of a part of Scotland Yard board showing different types of transport routs** Source:[20]

3.2 Game strategy

Strategy of Scotland Yard is dependent on cooperation of all detective players in the game. At the very beginning of the game detectives do not know anything about Mr.X's whereabouts. Therefore their goal should be to spread out on the map while trying to get to the strategic locations such as underground stations or longer bus routs. That way the detective players are able to move larger distances towards Mr. X once he reveals his location as well as blocking him escape routes in case any of the other detectives is close to him and he might want to transfer further away. By spreading across the board at least one of the players should always be relatively close to Mr.X once seen he reveals his position.

In order to win, players need to deduce which way Mr.X moved by knowing his previous location and the type of transfer he used. Before moving the detective player should consider Mr.X's next possible moves, the types of transport tickets he holds and thus can use and the position of other players. Later on in the game it is important for detective player to keep track of their as well as other player's remaining transport tickets in order to avoid the situation where any of them gets stuck on a station which they are not able to move from.

The detectives can take an advantage of the fact that the tickets used by them are given to Mr.X. Therefore by strategically playing certain types of tickets they can affect the ways Mr.X is going to be able to travel.

Mr.X on the other hand should be cautious of detective's locations and always think more steps ahead. He should prefer routes with more than one exit in case he gets surrounded by detectives.

It is considered a good strategy to use taxi tickets over other types when playing as Mr.X as it is the most common way of transportation making almost all of the stations on the board accessible by taxi. By using taxis Mr.X reveals less about his direction of the move.

Saving the black and double move tickets for the last rounds of the game is one of the best strategies to obtain when playing in a role of Mr.X as the detectives start to struggle with tickets shortage. It makes it much easier to move further away from them using underground or boat. At that point detectives might correctly guess Mr.X's location however they are usually unable to reach him.

3.3 Current machine playing engines for the game of Scotland Yard

Scotland Yard has been center of study in several publications. Although it does not have complex rules, thanks to its properties such as imperfect information in a form of hidden movement, asymmetry or need for cooperation of detectives it represents great challenges in the game theory.

Three bachelor thesis on this topic have been done just within the Brno University of Technology alone, results which are compared to results achieved in this thesis. Two of these papers written by M. Sova[28] and M. Gerža[11] implement and discuss Monte Carlo Tree Search method applied to the game of Scotland Yard. The third paper written by A. Tulušák[30] has chosen Alpha–Beta aproach. Although both Alpha–Beta and Monte Carlo seem to bring satisfactory results they tend to be too predictable by a human player, thus easily mastered.

Another paper[22] from 2011, implementing Monte Carlo Tree Search(MCTS) shows impact of limiting the number of possible locations of Mr.X by using information about Mr.X's moves along with their newly created technique called Location Categorization. By the end of the paper they prove that their implementation of MCTS can perform better compared to a commercial Scotland Yard program on the Nintendo DS.

Concerning deep learning techniques there has been published a more recent paper[6] using adversarial neural networks combined with Q–learning. The performed investigation showed that the designed network driven detectives were able to grasp the concept of cooperation and, as the paper stated, by the end of training were making an effort to catch Mr.X.

Scientists from DeepMind made an effort to create more universal algorithm for AI player able to master both perfect and imperfect information games. They refer to the algorithm as Player of Game or PoG[26]. The algorithm defeated the state-of-the-art agent in Scotland Yard as well as reached great performance in chess, poker and Go.

Another work[18] greatly inspired by Scotland Yard's concept of periodically reappearing Mr.X and detectives trying to catch him tried to solve the same problem using a Gridworld game to contribute to inventing AI usable in real-world security system applications. The game Gridworld used in the paper had similar board to the one that is being used in this thesis for playing the Scotland Yard and is shown in the next section. Attacker who was in similar role of Scotland Yard's Mr.X was periodically seen every third move by the defenders who were trying to catch him. In the paper the idea of combining QMDP and Bayesian inference into BayesQMDP has been introduced and tested on the mentioned game.

3.4 Simplified rules of Scotland Yard board game

Previous three mentioned bachelor's thesis[30][28][11] worked with simplified rules of Scotland Yard in order to reduce computational resources while testing performance of Alpha–Beta and Monte Carlo Tree Search methods on a Scotland Yard board game. In order to be able to compare the results from these three papers with the ones reached by using Deep Q-Network the same rules of the game were adapted.

New rules were composed in a way to simplify the original game as much as possible while keeping the concept of hide-and-seek mechanics. To reduce the number of states instead of having 200 different stations, game board represents 5x5 grid resulting in 25 different stations the players can stand on.

Resized board however also meant smaller space for the player to play on which was the reason why number of players was reduced to three, two detectives and Mr.X. The decision was based upon the fact that more detectives could put Mr.X into disadvantage due to less options of stations he could use. On the other hand the detectives could overrule Mr.X without making more effort thus not showing their full potential.

Another difference from original rules of Scotland Yard is reduction of transport options. Instead of using four transport types with separate routs and being able to travel longer distances players are able to use only one type of transport allowing them to always move one space ahead. Board as small as the one proposed in the previous thesis would not make much use of more transports which however leaves a room for future expansions and studies.

To sum it up the resulting rules are as follows. There are three players in total, one being Mr.X and the remaining two are in a role of detectives. Board consists of a grid sized

5x5 spaces. Each of the players can move one space up, down, left or right. The players take turns always starting with Mr.X who is then followed by the other two players–detectives. Position of Mr. X is hidden for the detectives and revealed on rounds 3, 6, 9, 12 and 15 where the game ends and Mr.X wins if not caught by agents by then. Positions of the players are chosen by random just as in original rules of Scotland Yard.

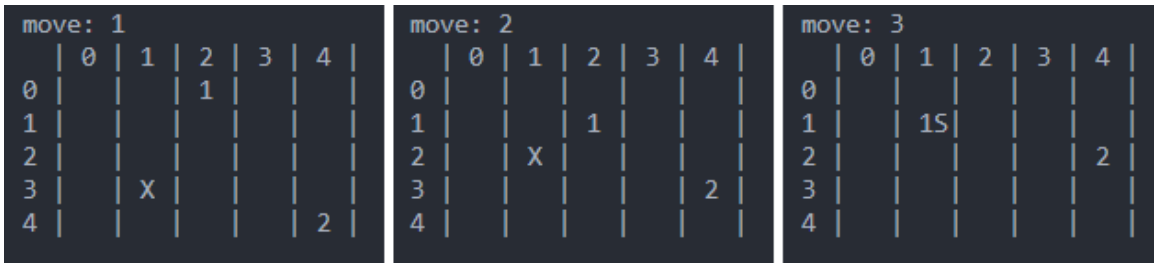

Figure 3.2: **Game played according to the simplified rules of Scotland Yard**

Figure 3.2 displays one complete game played by rules of Scotland Yard suggested in this section. The boards in the images show position of the players after both Mr.X and detectives moved in the given round. Letter „S“ in the rightmost image signifies position of Mr.X as well as the fact that at that given time his position is visible for other players.

Chapter 4

Reinforcement learning

In machine learning it is needed to gather vast amount of data to train a model successfully. The amount of data required for training increases with increasing complexity of a problem. Added to that, collected data in such amounts can often contain missing or incorrect values affecting the final results of a trained model[3].

Reinforcement learning overcomes the mentioned problems by reducing the need for data to train. Instead it aims for finding an optimal solution by an agent interacting with the environment. Every action taken during the learning phase is evaluated. Actions can lead to either positive or negative rewards. Agent then tries to take an action based on previously received rewards in order to maximize the notion of cumulative reward which leads to discovering an optimal policy for the given environment. An optimal policy is a policy that maximizes the expected total reward.

At the beginning agent does not know any information about its environment and gains knowledge based on rewards received. Therefore as [25] states the principal issue is exploration. An agent must experience as much as possible to learn how to behave within the environment. For this purposes Q–Learning for example, uses the exploration vs. exploitation trade–off in a form of ϵ–greedy strategy, which is introduced in the following section.

Reinforcement learning is a popular approach studied in game theory. Using Reinforcement Learning, computer algorithms such as Alpha Go and OpenAI Five have been able to achieve human level performance on games such as Go[27] and Dota 2[23].

One of the core concepts of Reinforcement Learning is the Deep Q-Learning algorithm that is described and implemented on a simplified version of Scotland Yard later on in this paper.

4.1 Q–Learning

Q–Learning is a model-free reinforcement learning algorithm used for learning the optimal policy in a Markov Decision Process. Its goal is to find an optimal policy by learning the optimal Q-values for each state-action pair. Q–Learning differs from other reinforcement learning algorithms by using off–policy method to separate the deferral policy from the learning policy [17]. In other words, to function, model first needs to undergo the learning process.

During this process Q–value is computed for each state–action pair and stored in a Q–table where rows represent states and columns represent actions that can be taken.

At the beginning of the learning process all values in a Q–table are initialized to zero. Values in the Q–table are then iteratively updated throughout the learning process. By the end of the learning phase Q–table is filled with the optimal Q–values which is the final result of the learning. From that point on to determine the action from a given state the model uses the optimal values from the table.

ϵ–greedy strategy is used to determine which of the actions should be executed from a current state. The strategy is based on exploration vs. exploitation trade off. Exploration allows the agent to explore so far unknown state–action pairs by choosing the action randomly which possibly might lead to higher reward. Exploitation on the other hand chooses the action with the highest Q–value present in the Q–table, making use of known facts. As the learning process begins the agent does not know anything about the environment therefore it must explore it which is done by setting the ϵ value to 1. The value of ϵ gradually decreases and exploitation is used more frequently as the time progresses and the agent gains knowledge. Further affects of ϵ–greedy strategy are described and discussed here[12].

To update Q-value for a given state–action pair the key formula is as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \underset{a'}{max} Q(s',a') - Q(s,a))$$

where a is an action executed in state s resulting in state s'. a' represents the future action taken from the state s'. α stands for learning rate and its value ranges from 0 to 1. R is the value of the reward received for taking an action a from the state s and γ is a discount factor [25].

To break it down the expression states that the Q–value for a given state–action pair is a sum of the existing Q–value for the given pair and the equation which determines the best action in the current state.

The goal is to bring the Q–value for the given state–action pair as close to the right hand side of the Bellman equation as possible. Q-value will eventually converge to the optimal Q-value q_*.

This is achieved over time by iteratively comparing the loss between the Q–value and the optimal Q–value for the given state–action pair and updating the Q–value in a Q–table to reduce the loss every time when encountered with the same state–action pair. This is done in a following manner:

$$\begin{aligned} q_*(s,a) - q(s,a) &= loss \\ E\Big[R_{t+1} + \gamma \underset{a'}{max} q_*(s',a')\Big] - E\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\right] &= loss \end{aligned}$$

As a part of their course Reinforcement Learning–Developing Intelligent Agents [7], DeepLizard explains the above principles of Q–Learning even further together with practical examples of the algorithm on a simple board game.

Although the Q–Learning algorithm has proven as one of the most effective reinforcement learning approaches, its advantages vanish with increasing complexity of the environment. Q–Learning relies on Q–table containing all possible state–action pairs. As [17] explains, when encountering problem with large state and action space, the algorithm demands enormous amount of memory to be able to save all existing state–action values.

Many Q–Learning algorithms, such as Deep Q–Learning, have been developed to solve the problem above while keeping the main concepts of the original algorithm described in this section.

4.2 Deep Q–Learning

Deep Q–Learning uses the main concept of Q–Learning but extends its idea even further by replacing the Q–table with a deep neural network – Q–network instead making it faster and more efficient at solving problems with large state–spaces. By leaving the Q–table behind, Deep Q–Network (DQN) algorithm solves the issue with unbearable amount of memory desired by Q–Learning to solve complex problems.

To achieve stability, DQN algorithm uses experience replay also known as replay memory to store the trajectory of the Markov decision process. During each iteration experience in a form of information: state, action, next state and reward is stored in replay memory. Later in the process mini–batches of states, actions, next states and rewards are sampled from the memory and used as observations to train the Q–network. The goal of DQN is to approximate the action-value function.

To obtain an unbiased estimator of the mean-squared Bellman error, DQN algorithm uses two separate neural networks[8]. It uses target network to obtain target value and Q–network which is using the obtained target value to learn, leading to reduction of correlations. This approach is called the target Q technique[17].

When implementing the DQN algorithm, weights of the Q–network need to be initialized with random values. Target network is then initialized as the exact copy of Q–network. Iteration through epochs starts taking place. Each epoch represents one whole game. During each step of the game current state is passed through Q–network after which the agent chooses an action based on ϵ–greedy strategy described earlier. The action is then executed into the environment and the agent is rewarded for the chosen action. The information gathered during this process is stored in replay memory. In the next step mini–batches of information stored in replay memory are created processed and passed through the target network. Resulting values from target network are used to calculate loss based on which the Q–network gets updated. After set number of iterations Q–network weights are copied to target network[16].

4.3 Double Q–Learning

Double Q–Learning is another variation of Q–Learning. It was introduced by Hado Hasselt[14] in order to solve the problem of Q–Learning not performing well in stochastic environment. The problem of Q–Learning is seeded in overestimation of action values. After certain time, original Q–Learning algorithm stops searching new optimal values. Instead, as described earlier, it repeatedly selects the highest action value from already existing values as an approximation for the maximum expected action value[17].

To avoid this, double Q–Learning uses two Q functions. Unlike in target Q technique used in deep Q-Learning, here both Q functions need to be learning from separate set of experiences and are both used for an action selection. This is done in order to avoid using the same set of values for both selection and evaluation of an action.

The algorithm is as follows[15]:

$$Q^A(s,a) \leftarrow Q^A(s,a) + \alpha(s,a)(R + \gamma Q^B(s',a') - Q^A(s,a))$$
$$Q^B(s,a) \leftarrow Q^B(s,a) + \alpha(s,a)(R + \gamma Q^A(s',a') - Q^B(s,a))$$

Double Q–Learning can either make use of Q–tables or replace them with neural networks just as deep Q–Learning does. As this paper describes deep learning methods, from this point on the method is described as a variant using neural networks.

The main principle of the algorithm stays as described in deep Q–Learning section. However after weight initialization of Q–Networks A and B, current state s is passed through both networks. The network with the higher Q value is chosen and action is selected by applying ϵ–greedy strategy. After executing the chosen action the reward and next state are observed and network update is performed.

When it comes to updating the networks, during each calculated step only one of the two networks gets updated. The selection of the network that gets updated is done by random. To update the chosen network evaluation of the performed action needs to be done. During the evaluation process, the network that is not chosen to be updated is used as a target network to calculate target Q value which can be seen in the equations above.

Step by step description of an algorithm along with code example is shown here[5].

4.4 Double Deep Q–Network

Despite the fact that both Double Q–Learning and Deep Q-Learning algorithms advance the original Q-Learning, there is still a room for improvement. As Hasselt has shown in his later paper[15] combining the two approaches into Double Deep Q–Network has proven even more efficient. The paper among other things, contains performance comparison of Double Deep Q–Network and Double Q–Learning on multiple games proving the rise of performance using this algorithm.

The Double Deep Q–Network (DDQN) algorithm uses two independent networks to choose the action from a given state in the same manner as Double Q–Network algorithm to avoid overestimation of action values. After performing the chosen action and observing its affect on the environment the networks need to be updated. In DDQN both networks get updated each step, which is done in the same way as in Deep Q–Learning.

Each of the two networks has its own target network used to estimate the target Q value needed for calculating the loss according to which the network gets updated. Target network is the exact copy of a network using its target values to update itself otherwise known as an online network. Target network gets updated periodically with online network weights each N steps.

Chapter 5

Deep Q-Learning implementation of Scotland Yard

After thorough research on a topic of AI algorithms known to be used to play Scotland Yard shown in section 3.3, Deep Q–network(DQN) approach had been chosen to implement and compare its results to other methods implementing this game.

One of the decisive factors was based on the fact that DQN is a foundation of Q learning algorithms, as it was explained in the previous chapter. Its results could thus, be considered a starting point in the future studies and easily build upon this paper.

The final decision to use DQN was inspired by T. Dash's study[6]. Unlike in his work, the implementation in this thesis is attempting to increase performance of the detectives by calculating all Mr.X's possible locations in a similar manner as it was originally proposed in the paper[22] using MCTS to solve autonomous game-play of Scotland Yard.

Main interest of this chapter is to describe final implementation of DQN algorithm on a simplified version of the Scotland Yard board game introduced in section 3.4. Firstly explaining the choice of used technologies the chapter then advances with walk-through the training parameters and used network layout together with its visuals.

5.1 Used technologies

The DQN algorithm has been implemented using Python alongside with PyTorch framework.

Part of M. Sova's thesis[28] contains an implementation of the environment for Scotland Yard written in Python. The very same environment has been slightly edited and used in implementation by M. Gerža[11]. As this paper aimed to compare the achieved results with all the three mentioned works the same game environment has been adapted which led to choosing Python as a programming language.

5.2 DQN implementation

DQN algorithm has been implemented as described in section 4.2. During the learning process, the detective agent is always paired up with the second detective using A.Tulušák's[30] implementation of agent movement. This decision was based on the assumption that the DQN agent could learn from more skilled agents as well as he could get used to the other agent moving around him on the board.

The agent, thus learns in the following way. In each round of a game first Mr.X makes his move using Alpha-Beta, MCTS or random combination of the two methods, choice of which is defined at the start of a program. After the first detective makes move by Alpha-Beta method, the current state of an environment is processed and passed to the neural network, which outputs the Q value.

The rest is a standard DQN routine procedure of choosing the action by *epsilon*-greedy policy, performing the chosen move, observing the next state, receiving the reward and finally updating the policy network all of which has been described in more details in section 4.2.

5.3 Neural Network architecture

Slightly different architectures have been used for training Mr.X and the detective models. Reasoning behind structures of used network layouts can be seen in the upcoming chapter 6 where the whole process is described on performed experiments.

Considering 5x5 size of the game board, the network architecture used to train model for game–play of Mr.X consists of 50 neurons input followed by two hidden layers of which the first is double the size of the input layer whereas the second comes back to using 50 neurons. Output layer consists of 4 final neurons outputting the Q value needed to determine the direction of the move.

The size of a network is flexible and can be easily modified for play on bigger board by setting the input neurons size on $n^2 * 2$. The next hidden layer is then of size $n^2 * 4$ and followed by the last hidden layer back at using $n^2 * 2$ neurons.

The network used by detective agents has identical layout of the one used by Mr.X except keeping the last layer of a size 100 neurons or $n^2 * 4$.

In both network architectures ReLu has been chosen as an activation function between all the layers besides the output layer where the linear activation has been kept. The choice of leaving linear activation function in the output layer over commonly used SoftMax activation was based on the fact that Q–learning makes use of Q values described in section 4.1. Instead of letting the network output the percentage of the best possible action as it is often times implemented, Q–learning aims for Q values which are used in later calculations and thus converting them into probabilities could affect results.

To speed up the learning process, instead of initializing weights of the network with random or zero values, Kaiming or also known as He initialization has been used. Opposite to inbuilt PyTorch initializer, the performance after using Kaiming uniform initialization has risen by 3%.

The visual representation of both used network architectures can be seen in the following diagrams.

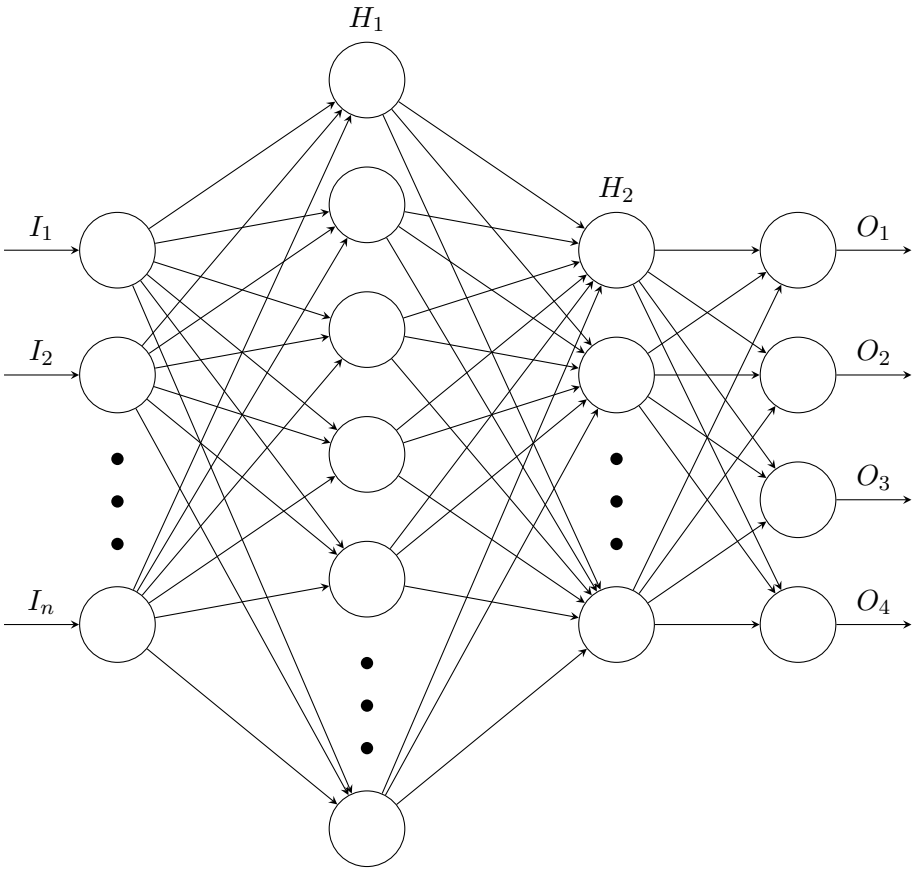

Figure 5.1: **Neural network architecture used to train Mr.X model**

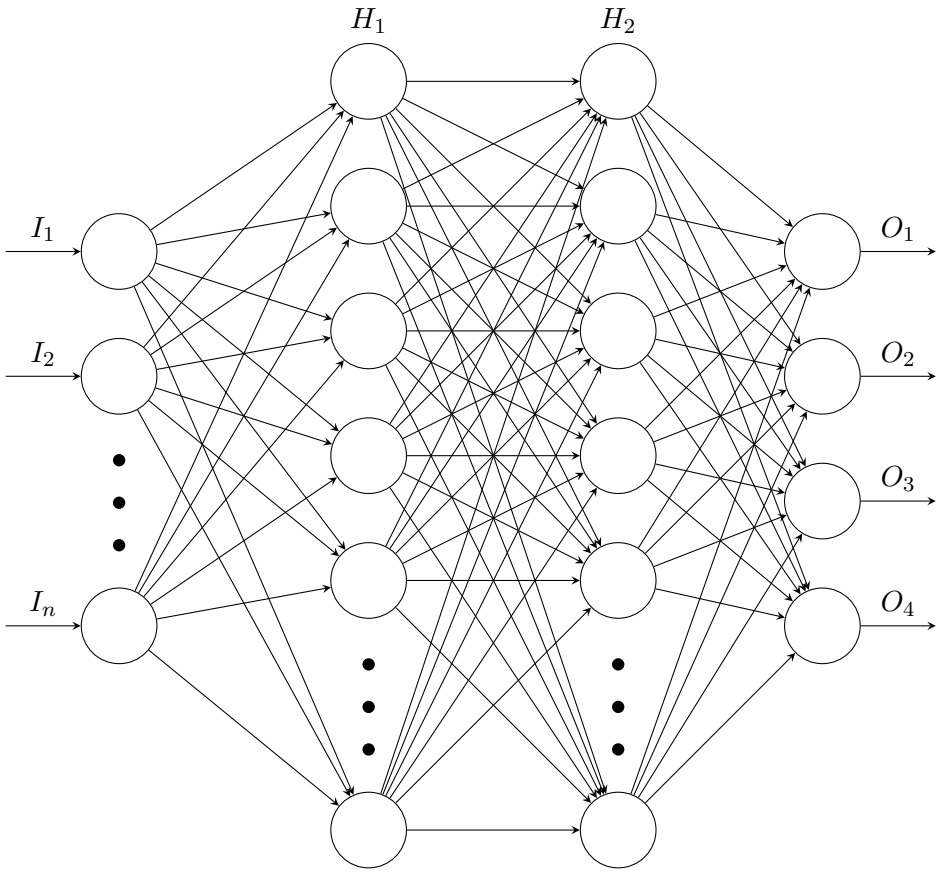

Figure 5.2: **Neural network architecture used to train detective model**

Each current state of the game is first preprocessed into two separate grids obtained in one 3D array. The first grid is used for marking the location of Mr.X and the second is indicating location of the detectives.

To represent the player's locations needed for Mr.X's move calculation, his and the detective's positions are marked in a corresponding grids by 1.

However, because the location of Mr.X is most of the time hidden, to indicate the detectives his location, Mr.X's grid is filled with probability with which he is standing on each station on the board giving the detectives clues where to search. All of these information are then reshaped into 1D array and passed to the neural network as an input.

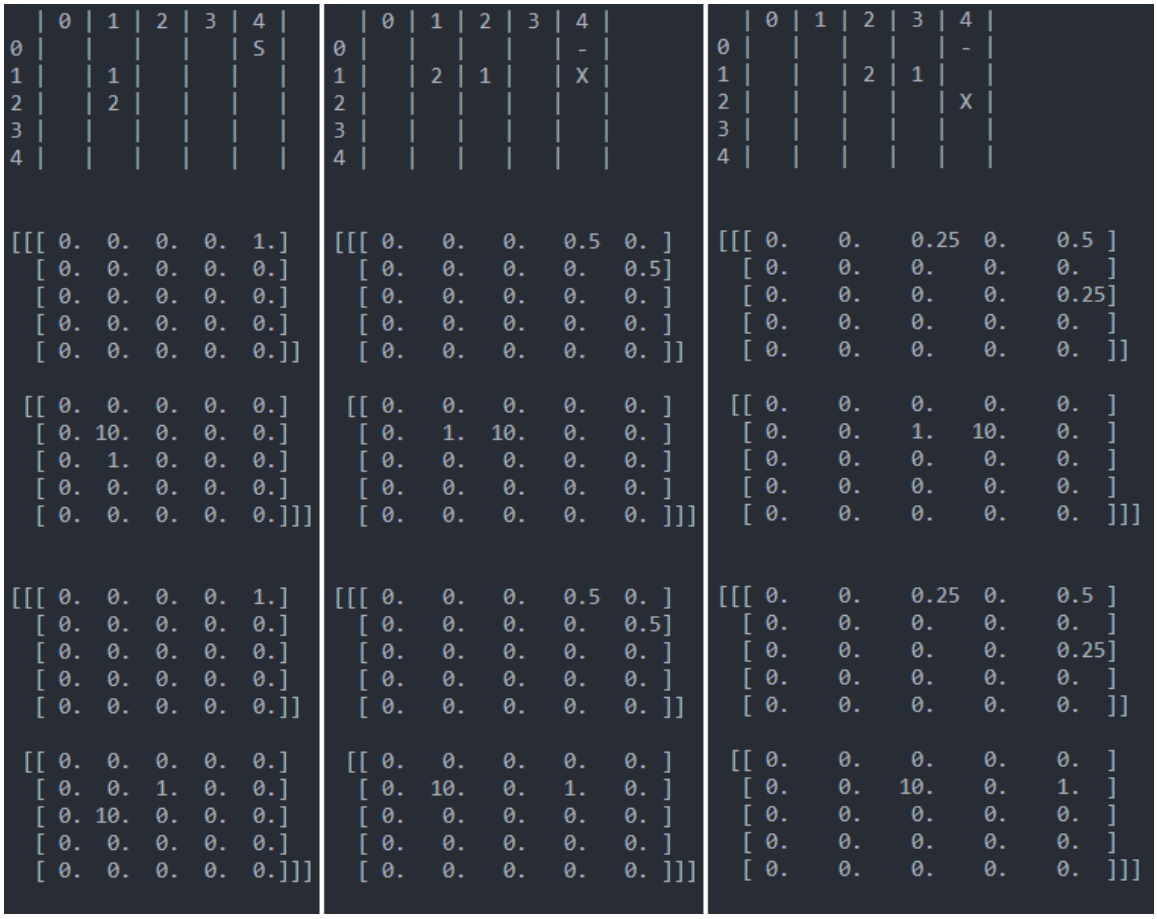

Figure 5.3: **Three consecutive rounds of game play visualizing input states for the neural network used for calculating next detective's action.** The first row represents move made by Mr.X. Second row shows preprocessed game state for the detective no.1. Finally the third row displays preprocessed state for the detective no.2

Each column in the figure above shows one of the three consecutive rounds of a game play after Mr.X took turn and agents are about to be moved. Below each board state there are two 3D arrays each of them used as an input state passed through neural network to calculate corresponding action for the first and the second detective. Top board visible in the 3D array depicts position of Mr.X, whereas lower board represents detective's current positions. Detective for whom the action is calculated is labeled as 10 and the other detective that is currently not being moved as 1.

5.4 Rewarding system

As it was mentioned in the section 4 positive and negative reinforcement is what drives the Q–learning agents and teaches them how to improve their actions. In this thesis's implementation of DQN several rewarding systems have been tested, two of which are discussed closer in section 6.2, however the rewarding system described in this section has proven to bring the best results.

The detective gets rewarded with positive 1 if stepping on station that could possibly be current location of Mr.X. In case he wins the game by either stepping on station Mr.X

occupies or helps the other agent to surround Mr.X in the corner of the board, he gets rewarded with 10 points. On the other hand if the detectives loose the game he is punished with negative 10 points. Awarding agent with negative reward when moving out of board space or stepping on a station that is occupied by the other detective has proven ineffective, thus in this case the reward value received by the agent is equal to 0.

Mr.X is rewarded 1 point each round he manages to avoid detectives and gets negative 5 points if he gets caught and looses the game. In case Mr.X agent chooses to move out of the game board his chosen move is abandoned and new move for the given round is chosen by random. If he survives such move without being caught by the detectives he is rewarded with 0, the same way detective agents are when encountering similar situation.

5.5 Training parameters

Based on the fact that neural networks learn by themselves, their behaviour varies each time when learning different task. Thanks to this, setting parameters of a given network is dependent on a problem it is meant to solve. Although knowledge of each parameter's purpose and its affect on learning process most certainly helps, because of unpredictability of neural network's behaviour, the parameters are many times tweaked into perfection by trial and error approach.

The parameters that have been chosen for this thesis's DQN implementation are presented in this section.

Batch size has been set to size of 512 samples suggested by paper[29], studying the accelerated methods for deep reinforcement learning, as the best behaving batch size for DQN.

The size of replay memory, from which batches were sampled while training, was set to 15000. Among tested 10000, 15000 and 20000 sizes of memory, this value proved to be the optimum size for this work's implementation of DQN.

Learning rate was tweaked into 0,001 final value that showed the best results, whereas γ made use of 0,975 value.

Policy network was set to update every step using HuberLoss function, used to calculate loss value needed for the update along with Adam optimizer. Its weight values are copied into target network every 100 steps. Besides HuberLoss function, Mean squared error (MSE) loss function has been tested, however results of the tests performed suggested HuberLoss was performing better within this thesis's implementation.

Value of ϵ is at the beginning of the training process equal to 1 representing 100 percent chance of random choice of action. The value updates after each game or otherwise called epoch, using this formula: $\epsilon = \epsilon - 1/epochs$ until it reaches 0,01 which is the lowest value possible for this parameter.

Chapter 6

Experiments

To perfect the learning process in order to get the best possible results, multiple experiments had been run on networks of Mr.X and detective separately. Used network architecture, rewarding system and DQN hyper-parameters presented in chapter 5 were all results of performed experiments.

This chapter describes the fine tuning process and tries to express reasoning behind the chosen network architectures, rewarding system and opponents used in the final version of implementation.

All of the test results included in this section were product of 10000 game repetitions following manners of M. Gerža[11] who used the same game count for testing his MCTS implementation.

M. Gerža's[11] as well as A.Tulušák's[30] implementations were used during the process of training the models, leaving out M. Sova's[28] implementation as it implemented the same method as M. Gerža while achieving lower performance.

All of the models presented in this chapter were trained on 10000 games. The amount of games needed to train the model was determined by training a model with the identical parameters for a period of 5000, 10000 and 20000 games. While 5000 games was not enough time to bring satisfactory results, 20000 games turned out to be more time than needed and the model started to retrain making the whole training process counterproductive.

6.1 Experimenting on Mr.X's network

The first experiment included in this section was aimed to determine the best opponent to train Mr.X's model with in order to maximize his winning rate as well as to create the most universal opponent possible.

The second experiment helped to define the most efficient architecture of a neural network used to train the model of Mr.X.

The last experiment tried to combine knowledge learned from the previous two experiments which led to achieving the best results and thus the strategy was applied in the final implementation.

Experiment 1

To determine which method represents the best opponent for Mr.X to learn on, we proposed an experiment to test 3 different opponent types. The first model was trained using only Alpha-Beta while the second model used only MCTS method to play detective's moves

while training. The third training applied a random choice between the two mentioned types of methods reset before each game.

	A–B	MCTS 2022	A–B&MCTS
random	24,35%	23,09%	22,41%
A–B	52,96%	75,2%	61,83%
MCTS	80,4%	69,05%	69,61%

Figure 6.1: **Performance of trained models where columns represent method used to train each model and rows represent the method used while testing.**

As it can be seen in the table above, Mr.X won the most games against A–B driven detectives when the network was trained using A–B detectives as an opponent.

The same could be applied for MCTS as Mr.X performed the best when trained against this method yet his performance dropped when playing against A–B.

This was caused by the fact that Mr.X learned a specific strategy to win against one type of method. However, the same strategy he acquired was not applicable to a different game style used by another method.

Although the model trained using both methods did not reach the best results when compared to the models playing against the same methods they were trained on, it has been considered the best approach as it turned out to be the most universal.

Experiment 2

Another important aspect of the process was to determine the architecture of a neural network used by DQN agent. For this purpose there had been suggested experiment to perform three sets of training each with the same DQN parameters but modified architecture of neural network.

The first architecture used two hidden layers, the first of a size 100 neurons and the second 50 neurons.

The second architecture made use of one additional hidden layer of a size 100 neurons. Its full structure then consisted of 50 neurons in input layer followed by two 100 neuron hidden layers continuing with one more hidden layer made of 50 neurons and finished by 4 neurons in output layer.

Third and the last model used 4 hidden layers three of which consisted of 100 neurons each and the last one consisted of 50 neurons just as the two previously used models.

All the three models have been trained using Alpha-Beta implementation of the detectives as an opponent as the experiment was taking place simultaneously with the first experiment which led to not being able to apply its results in use.

	2 hidden	3 hidden	4 hidden
random detectives	22,04%	22,41%	21,76%
A–B 2020	53,23%	61,83%	67,4%
MCTS 2022	80,89%	69,61%	68,01%

Figure 6.2: **Performance of models trained using architectures consisting of different number of hidden layers.** Rows represent detective's method while testing.

The performed experiment revealed the architecture with the lowest number of hidden layers could produce model with the highest winning scores against the Alpha-Beta method which essentially was a method the model was trained on, thus it meant it could have the best learning potential. Therefore one additional experiment was proposed using network architecture similar to the best performing one at that time.

The last network consisted of two hidden layers sized 100 neurons each. However, the results of model trained using this network layout did not surpass the performance of the models displayed in the table above therefore it was no longer used.

Experiment 3

The first experiment proved that the model learning next to mix of both Alpha–Beta and MCTS methods as training opponent received the best performance. However, from the second experiment it was still uncertain whether using two or three hidden layers in the network brought better results.

Therefore, the third experiment was proposed. Training both architectures using mix of A–B and MCTS methods following the first experiment was supposed to reveal which of the two architectures could reach better overall performance.

	2 hidden layers	3 hidden layers
random detectives	17,36%	22,41%
A–B 2020	60,83%	61,83%
MCTS 2022	62,05%	69,61%

Figure 6.3: **Performance comparison of Mr.X models trained using network architecture consisting of two and three hidden layers.**

From results included in the table above, it was concluded the most efficient neural network architecture consisted of two hidden layers. Diagram of architecture chosen based on this experiment can be seen in section 5.1.

6.2 Experimenting on detective's network

Experiments in this section made use of knowledge gained by previous experiments performed on model for Mr.X. All of the training sessions described in this section therefore used combination of both Alpha-Beta and MCTS methods to determine movement of Mr.X on the board.

While similar experiment was proposed to determine the method for the second detective, as only one of them was driven by neural network during training, the experiment did not reach the successful end. Performing more than 5000 game iterations needed for model training on MCTS implementation while using only one of the detectives resulted in segmentation fault each time before it reached the end as the implementation was never meant to be used with one agent only. The decision was made to make use of A–B implementation instead, which was used in all the experiments.

The first included experiment was targeted to find the best possible network architecture to train the detective agent with, whereas the second experiment aimed to express the reasoning behind the choice of rewarding system in the final implementation.

Experiment 1

Experiment performed on Mr.X's model revealed that the most efficient network layout used 2 hidden layers out of which the first one consisted of 100 and the second 50 neurons. However based on the fact that the objective of detective differs from the objective of Mr.X it was decided to put into test three different layouts and observe the results.

The first layout represented the architecture chosen in the final implementation 5.2 of Mr.X using 2 hidden layers, with the same size of 100 neurons.

The second layout consisted of 2 hidden layers, one of a size 100 and second of 50 neurons.

The third layout was an extended version of the second layout using two 100 neuron layers and the third 50 neuron hidden layer.

	First layout	Second layout	Third layout
random detectives	96,65%	93,94%	86,15%
A–B 2020	27,5%	27,08%	9,2%
MCTS 2022	29,8%	23,07%	11,23%

Figure 6.4: **Performance comparison of detective models trained using network architecture consisting of two and three hidden layers.**

This experiment revealed that the architecture achieving the best results while training model of Mr.X did not reach the same best results while training detective model. On the other hand the first design of network architecture which performed weak when training Mr.X turned out to bring the best results in detective's training.

Experiment 2

While determining rewarding system for Mr.X was a straight forward process, there were more options of rewarding the detective for his action. Two of these possible rewarding systems were chosen and put to test in this section.

Both of the rewarding systems shared 1 point reward for detective agent stepping on a station with possible occurrence of Mr.X. This was done in order to teach the agent that such stations were more desirable for him to search through.

The systems also shared -10 reward for loosing the game as it was important to teach the agent to avoid such situation.

The systems differed in following ways.

The first system rewarded the agent with 10 points when the detectives won the game not taking into consideration whether the agent who won the game used DQN or A–B implementation. The idea behind this was to teach the agent cooperation with the other agent and let him know he is not the only one interacting with the environment. Inspiration for this approach came from OpenAI's paper[2] where they successfully taught neural network agents to play the game of hide-and-seek.

The second system rewarded the agent with 10 points when the detectives the game but the DQN detective contributed to the win to see if the agent would start to rely on the other agent less and tried to win a game by himself.

	First system	Second system
random detectives	92,30%	96,31%
A–B 2020	10,49%	21,13%
MCTS 2022	12,41%	18,78%

Figure 6.5: **Performance of models trained using different types of rewarding systems**

The performed experiment proved DQN agent was able to learn better strategy when rewarded only for his actions. However, rewarding the DQN agent when helping AB agent to surround Mr.X in a corner of the game board, making it impossible for him to run away did achieved success of teaching the DQN agent somewhat cooperation.

6.3 Results

After the first training of both Mr.X and detectives models by using final DQN implementation described in section 5, models were retrained multiple times each time getting slightly better results. When retraining a given model, ϵ value has been set to 0.20 in order to allow the retraining model to use gained knowledge while leaving 20% chance of discovering better strategy by random movement.

The best achieved results were compared to the performance of the other already existing methods by running 10000 game iterations.

The value of simulation time needed by MTCS implementations was set to 0,05 in an attempt to fasten the process as the other two used methods needed less computation time. This time reduction however showed not to have significant implact on method's performance.

	DQN 2023	MCTS 2022	MCTS 2021	A–B 2020
DQN Hrkľová 2023	–	39,88%	88,89%	33,79%
MCTS Gerža 2022	62,05%	–	93,83%	66,02%
MCTS Sova 2021	25,11%	24,39%	–	17,86%
A–B Tulušák 2020	60,03%	87,39%	99,1%	–

Figure 6.6: **Performance comparison of final DQN implementation.** Rows represent method applied on detectives, columns represent method used for Mr.X movement.

The overall results showed that detectives and Mr.X using DQN were able to grasp the main concept of a proposed game. As it can be seen in the table 6.6 DQN Mr.X succeeded to overcome both MCTS and Alpha-Beta method's Mr.X and represented the toughest opponent for the two methods.

The detectives on the other hand managed to surpass only one of the existing implementations. However by playing against random Mr.X and reaching 98,60% winning rate it is more than safe to say the agents understood the rules and concept of the game.

The experiment from section 6.1 proved it was possible to reach better performance against each of the models presented in the table by training the model against one certain type of method. By training Mr.X using Alpha-Beta agents it was possible to lower their learning rate to 53,23% chance of winning only after first training of the given model. However, the aim of this thesis was to create the most possible universal model for both detectives and Mr.X.

6.4 Suggestions for future improvement

This thesis proved that by using the most basic form of Q–learning combined with neural network in the form of Deep Q-learning (DQN), the detectives were able to understand rules of the game presented to them as well as they would make an effort to win the game by trying to search through location of possible occurrences of Mr.X.

This discovery led to the conclusion that the results achieved in this thesis could be greatly improved by one of the presented modifications of DQN presented in chapter 4.

In addition, the DQN implementation could possibly bring much better results by making use of convolutional neural networks instead of the linear layers used in this thesis's implementation.

By implementing the algorithm using convolutional neural network(CNN) the game could be possibly expanded to a version closer to the original Scotland Yard by adding at least one type of different transport, making it more interesting to play and thus more appealing to real-life players.

Chapter 7

Conclusion

The aim of this thesis was to acquaint myself with deep learning methods used in game theory and apply the gained knowledge on the simplified rules of the Scotland Yard board game in order to improve performance of already existing methods implemented for this game.

In the early stages it was crucial to fully grasp the concept of uncertainty in a form of hidden movement present in the game of Scotland Yard. Therefore several board games with the mentioned mechanism were studied 2. Each game was broken down and looked at from three different angles; its rules, game strategy and research of existing artificial intelligence able to play described games.

After that thorough research of the Scotland Yard board game 3 had been performed to learn its rules, strategy of game-play, but most importantly currently known techniques used to design machine learning agents for this game.

After proposing simplified rules of Scotland Yard 3.4 used in this thesis's implementation of the given game, research of deep learning methods had been done with the main focus on Q-learning algorithms 4.

Deep Q-Network(DQN) algorithm had been chosen and implemented to play the mentioned board game. The decision was made, based on the fact that DQN algorithm represented the basis of Q–learning. Thus, in the future it could be easily expandable into one of the other forms of Q-learning algorithms mentioned in section 4 while using this thesis to build upon.

There were several experiments 6 performed on the proposed implementation to improve training process of the models used to determine next move of the players placed on the board. The experiments described in this thesis included description of designing the architecture of neural network used in the final implementation as well as process of choosing the opponents to train both models on.

In order to serve this thesis's purpose of becoming a good starting point of extended Q-learning studies, the final implementation of DQN had been fully described in section 5.

The final trained models were put to test and their results were compared with three existing implementations of two different methods namely Alpha-Beta and Monte Carlo Tree Search 6.3.

While the experiments have shown that the algorithm was able to train better performing models by training them using one of the mentioned methods only, the goal set at the beginning was to create the most universal model possible in order to achieve better game experience to human player. Therefore, both methods were used during training the final models, which led to better overall performance. The results showed that final Mr.X

represented the toughest opponent for both M. Gerža's and A. Tulušák's implementations of detectives. However, when tried to play against human opponent it was visible that the implementation still had a lot to improve.

Model used by detectives managed to surpass only one of the three implementations it was tested on. However, from observing its game-play and move choices it was visible that the detectives tried to search for Mr.X in the places with certain amount of probability of his occurrence.

The biggest advantages of DQN opposite to other methods turned out to be computational resources and time needed to perform a given move running 10000 games against random player under 4 minutes, making it the most perspective method to implement on original rules of Scotland Yard in the future.

Making use of convolutional neural networks could greatly improve performance of the implemented DQN algorithm alongside with upgrading the algorithm to Double Deep Q-Network.

Bibliography

[1] Atlas, A. I. *Stratego Board Game*. 2019 [cit. 2023-04-30]. Available at: https://www.boardgamehalv.com/stratego/.

[2] Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G. et al. Emergent Tool Use From Multi-Agent Autocurricula. 2020.

[3] Banoula, M. *What Is Q-Learning: The Best Guide To Understand Q-Learning* [online]. 2023 [cit. 2023-04-20]. Available at: https://www.simplilearn.com/tutorials/machine-learning-tutorial/what-is-q-learning.

[4] Burggraf, M., Garrels, D., Hoermann, W., Ifland, F., Werner, S. et al. *Scotland Yard* [Rulebook]. Milton Bradley Co., 1985. Available at: https://www.hasbro.com/common/instruct/ScotlandYard.PDF.

[5] Code, R. *Double Q-Learning Python and Open AI* [online]. 2020 [cit. 2023-04-25]. Available at: https://rubikscode.net/2020/01/20/double-q-learning-python/.

[6] Dash, T., Dambekodi, S., Reddy, P. and Abraham, A. Adversarial neural networks for playing hide-and-search board game Scotland Yard. *Neural Computing and Applications*. april 2020, vol. 32. DOI: 10.1007/s00521-018-3701-0.

[7] DEEPLIZARD. *Reinforcement Learning - Developing Intelligent Agents* [online]. 2018 [cit. 2023-04-19]. Available at: https://deeplizard.com/course/rlcpailzrd.

[8] Fan, J., Wang, Z., Xie, Y. and Yang, Z. A Theoretical Analysis of Deep Q-Learning. In: Bayen, A. M., Jadbabaie, A., Pappas, G., Parrilo, P. A., Recht, B. et al., ed. *Proceedings of the 2nd Conference on Learning for Dynamics and Control*. PMLR, 10–11 Jun 2020, vol. 120, p. 486–489. Proceedings of Machine Learning Research. Available at: https://proceedings.mlr.press/v120/yang20a.html.

[9] Fan, M. B. *Sneaking around the abbey is sinfully-silly fun in Nuns On The Run.* [cit. 2023-04-30]. Available at: http://www.megabearsfan.net/post/2022/12/02/Nuns-on-the-Run-board-game-review.aspx.

[10] Games, N. *FURY OF DRACULA: DIGITAL EDITION* [online]. 2020 [cit. 2023-04-27]. Available at: https://nomadgames.co.uk/blog/fury-of-dracula-dev-blog-5.

[11] Gerža, M. *Strategická desková hra s neurčitostí*. Brno, CZ, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Available at: https://www.fit.vut.cz/study/thesis/24711/.

[12] Guo, M., Liu, Y. and Malec, J. A new Q-learning algorithm based on the metropolis criterion. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*. 2004, vol. 34, no. 5, p. 2140–2143. DOI: 10.1109/TSMCB.2004.832154. Available at: https://ieeexplore.ieee.org/abstract/document/1335509.

[13] Hand, S. *Fury of Dracula* [Rulebook]. 4th ed. Games Workshop Limited, 2018. Available at: https://wizkids.com/posters/repository/wizkids/FoD_4E_LearnToPlay-Online-OPT.pdf.

[14] Hasselt, H. Double Q-learning. In: Lafferty, J., Williams, C., Shawe Taylor, J., Zemel, R. and Culotta, A., ed. *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2010, vol. 23. Available at: https://proceedings.neurips.cc/paper_files/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf.

[15] Hasselt, H. van, Guez, A. and Silver, D. Deep Reinforcement Learning with Double Q-learning. 2015. Available at: https://arxiv.org/abs/1509.06461.

[16] Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T. et al. Deep Q-learning from Demonstrations. 2017. Available at: https://ojs.aaai.org/index.php/AAAI/article/view/11757.

[17] Jang, B., Kim, M., Harerimana, G. and Kim, J. W. Q-Learning Algorithms: A Comprehensive Classification and Applications. *IEEE Access*. 2019, vol. 7, p. 133653–133667. DOI: 10.1109/ACCESS.2019.2941229. Available at: https://ieeexplore.ieee.org/abstract/document/8836506.

[18] Klima, R., Tuyls, K. and Oliehoek, F. A. Model-Based Reinforcement Learning under Periodical Observability. In: *2018 AAAI Spring Symposia, Stanford University, Palo Alto, California, USA, March 26-28, 2018*. AAAI Press, 2018. Available at: http://www.fransoliehoek.net/docs/Klima18MALIC.pdf.

[19] Mogendorff, J. J. *Stratego Instructions For 2 Players Ages 8 to Adult* [Rulebook]. Milton Bradley Company, 1986.

[20] Mortensen, E. *Scotland Yard Board Game Review and Rules*. 2018 [cit. 2023-04-30]. Available at: https://www.geekyhobbies.com/scotland-yard-board-game-review-and-rules/.

[21] Moyersoen, F. *Nuns On The Run!* [Rulebook]. Mayfair Games, Inc., 2010. Available at: https://www.mind-maze.uk/nuns-on-the-run/rules/notr-rules.pdf.

[22] Nijssen, J. and Winands, M. Monte-Carlo Tree Search for the Game of Scotland Yard. In:. October 2011, p. 158 – 165. DOI: 10.1109/CIG.2011.6032002.

[23] OpenAI, :, Berner, C., Brockman, G., Chan, B. et al. Dota 2 with Large Scale Deep Reinforcement Learning. 2019. Available at: https://arxiv.org/abs/1912.06680.

[24] Perolat, J., Vylder, B. D., Hennes, D., Tarassov, E., Strub, F. et al. Mastering the game of Stratego with model-free multiagent reinforcement learning. *Science*. American Association for the Advancement of Science (AAAS). dec 2022,

vol. 378, no. 6623, p. 990–996. DOI: 10.1126/science.add4679. Available at: https://doi.org/10.1126%2Fscience.add4679.

[25] Russell, S. J. and Norvig, P. *Artificial Intelligence: a modern approach*. 3rd ed. Pearson, 2009. ISBN 0-13-604259-7.

[26] Schmid, M., Moravcik, M., Burch, N., Kadlec, R., Davidson, J. et al. Player of Games. 2021. Available at: https://arxiv.org/abs/2112.03178.

[27] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L. et al. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*. Nature Publishing Group. january 2016, vol. 529, no. 7587, p. 484–489. DOI: 10.1038/nature16961. Available at: https://www.nature.com/articles/nature16961#citeas.

[28] Sova, M. *Strategická desková hra s neurčitostí*. Brno, CZ, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Available at: https://www.fit.vut.cz/study/thesis/23706/.

[29] Stooke, A. and Abbeel, P. Accelerated Methods for Deep Reinforcement Learning. 2019. Available at: https://arxiv.org/abs/1803.02811.

[30] Tulušák, A. *Strategická desková hra s neurčitostí*. Brno, CZ, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Available at: https://www.fit.vut.cz/study/thesis/22304/.

[31] Unknown. *Fury Of Dracula Review*. [cit. 2023-04-30]. Available at: http://www.polyhedroncollider.com/2016/08/fury-of-dracula-review.html.

Appendix A

Contents of the included storage media

- `src/` – Folder with source files
- `latex/` – Folder with LaTeXsource files of this thesis
- `README.md` – README file for the project
- `thesis.pdf` – Thesis report file
- `thesis-print.pdf` – Thesis report file for print

`src/` folder besides source files implementing DQN algorithm further contains:

- `AT_2020/` – Folder with source files implementing Alpha–Beta by A. Tulušák
- `MC_2022/` – Folder with source files implementing MCTS by M. Gerža
- `MS_2021/` – Folder with source files implementing MCTS by M. Sova
- `trained_models/` – Folder with pretrained models