FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Srovnání metod strojového učení v optimalizaci strategií hry Blackjack
Student:	Matěj Severýn
Vedoucí:	Ing. Mgr. Ladislava Smítková Janků, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Umělá inteligence 2021
Katedra:	Katedra aplikované matematiky
Platnost zadání:	do konce letního semestru 2025/2026

Pokyny pro vypracování

V práci se věnujte problematice optimalizace strategie hry Blackjack. Proveďte srovnání vybraných metod strojového učení.
Pokyny:
1) Popište hru Blackjack, systém pravidel hry a modifikace.
2) Zpracujte rešerši existujících metod a strategií pro hru.
3) Analyzujte metody založené na posilovaném učení a navrhněte vlastní model.
4) S využitím existujících nástrojů a knihoven proveďte implementaci.
5) Navrhněte experimenty, proveďte je a vyhodnoťte.

[1] Millman, M: A Statistical Analysis of Casino Blackjack, The American Mathematical Monthly, Vol. 90, No. 7 (Aug. - Sep., 1983), pp. 431-436 (6 pages)
[2] Watkins, J.C.Ch.H. et al.:Machine Learning, 8, 279-292 (1992), Kluwer Academic Publishers, Boston
[3] Carlin, I., B, Dayan, P. Fear and loathing in Las Vegas: Evidence from blackjack tables, Judgment and Decision Making, Vol. 4, No. 5, August 2009, pp. 385–396

Elektronicky schválil/a Ing. Magda Friedjungová, Ph.D. dne 19. listopadu 2024 v Praze.

Bakalářská práce

SROVNÁNÍ METOD STROJOVÉHO UČENÍ V OPTIMALIZACI STRATEGIÍ HRY BLACKJACK

Matěj Severýn

Fakulta informačních technologií
Katedra aplikované matematiky
Vedoucí: Ing. Mgr. Ladislava Smítková Janků, Ph.D.
16. května 2025

Odkaz na tuto práci: Severýn Matěj. *Srovnání metod strojového učení v optimalizaci strategií hry Blackjack.* Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2025.

Chtěl bych především poděkovat své vedoucí bakalářské práce, paní Ing. Mgr. Ladislavě Smítkové Janků, Ph.D., za odborné vedení, cenné rady a vstřícný přístup během celé doby zpracování mé práce.
Dále děkuji Fakultě informačních technologií ČVUT za poskytnuté zázemí a znalosti, které mi umožnily tuto práci realizovat.
Velké poděkování patří také mé rodině za podporu, trpělivost a motivaci během celého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona

Prohlašuji, že jsem v průběhu příprav a psaní závěrečné práce použil nástroje umělé inteligence. Vygenerovaný obsah jsem ověřil. Stvrzuji, že jsem si vědom, že za obsah závěrečné práce plně zodpovídám

V Praze dne 16. května 2025

Abstrakt

Tato bakalářská práce se zabývá problematikou optimalizace strategie hry Blackjack s využitím metod posilovaného učení. Cílem práce je porovnat vybrané algoritmy a vyhodnotit jejich efektivitu při hledání herní strategie, která by zaručovala dlouhodobý zisk a poskytovala hráči výhodu nad kasinem. Práce se soustředí na tři hlavní přístupy: tradiční metodu počítání karet, algoritmus Q-learning a jeho rozšířenou verzi Deep Q-learning. Nejprve je představena hra Blackjack, její pravidla a nejčastěji používané varianty. Následuje rešerše existujících strategií a analýza výše zmíněných metod. Pro každý z těchto přístupů je navržen a implementován model, který je následně testován v herním prostředí. Výsledky práce ukazují, za jakých podmínek a s jakými omezeními je možné ve hře Blackjack dosáhnout dlouhodobého zisku.

Klíčová slova Blackjack, počítání karet, optimalizace strategie, posilované učení, Q-learning, Deep Q-learning

Abstract

This bachelor's thesis deals with the problem of optimizing strategies for the game of Blackjack using reinforcement learning methods. The main objective of the thesis is to compare selected algorithms and evaluate their effectiveness in finding a playing strategy that ensures long-term profit and provides the player with an advantage over the casino. The thesis focuses on three main approaches: the traditional method of card counting, the Q-learning algorithm, and its extended version, Deep Q-learning. First, the game of Blackjack is introduced, including its rules and most commonly used variations. This is followed by a review of existing strategies and an analysis of the aforementioned methods. For each of these approaches, a model is designed and implemented, and subsequently tested in a simulated game environment. The results demonstrate under which conditions, and with what limitations, it is possible to achieve long-term profit in the game of Blackjack.

Keywords Blackjack, card counting, strategy optimization, reinforcement learning, Q-learning, Deep Q-learning

Obsah

1 Úvod **1**

2 Teoretická část **2**

2.1 Blackjack . . . 2
2.1.1 Pravidla hry . . . 2
2.2 Počítání karet . . . 4
2.2.1 Princip . . . 5
2.2.2 Historie a vývoj metody . . . 5
2.2.3 Systémy . . . 5
2.2.4 Hi-Lo systém . . . 7
2.3 Posilované učení . . . 11
2.3.1 Základy posilovaného učení . . . 11
2.3.2 Formální definice problému posilovaného učení . . . 12
2.3.3 Algoritmy posilovaného učení . . . 13
2.3.4 Průzkum vs. využití (Exploration vs. Exploitation) . . . 14
2.3.5 Aproximace hodnotových funkcí . . . 14
2.4 Q-learning . . . 15
2.4.1 Základní principy . . . 15
2.4.2 Q-funkce a Bellmanova rovnice . . . 15
2.4.3 Algoritmus Q-learning . . . 15
2.4.4 Pseudokód Q-learningu . . . 16
2.4.5 Explorace vs. exploatace . . . 16
2.4.6 Konvergence . . . 17
2.5 Deep Q-learning . . . 17
2.5.1 Hlavní myšlenka . . . 17
2.5.2 Ztrátová funkce . . . 17
2.5.3 Zlepšení stability učení . . . 18
2.5.4 Algoritmus Deep Q-learning . . . 19
2.5.5 Výhody a nevýhody . . . 19

3 Implementace **20**

3.1 Herní prostředí - Blackjack . . . 21
3.1.1 Struktura aplikace . . . 21
3.1.2 Průběh hry . . . 21
3.1.3 Herní logika . . . 22
3.1.4 Správa balíčku karet . . . 22

3.1.5 Sledování odehraných karet . . . 23
3.1.6 Interakce s hráčem nebo agentem . . . 23
3.2 Agent používající Hi-Lo systém . . . 23
3.2.1 Výpočet Hi-Lo skóre . . . 23
3.2.2 Hlavní metody . . . 24
3.2.3 Výběr základní jednotky . . . 24
3.3 Agent používající Q-learning . . . 25
3.3.1 Struktura Q-tabulek . . . 25
3.3.2 Fáze činnosti agenta . . . 25
3.3.3 Předzpracování dat . . . 26
3.3.4 Q-funkce . . . 27
3.3.5 Metoda *epsilon-greedy* . . . 28
3.4 Agent používající Deep Q-learning . . . 29
3.4.1 Architektura agenta . . . 29
3.4.2 Replay memory . . . 30
3.4.3 Modely neuronových sítí . . . 31
3.4.4 Q-funkce . . . 33
3.4.5 Explorace pomocí parametrického šumu . . . 34

4 Experimenty **36**
4.1 Srovnání metod na základě výkonnosti . . . 36
4.2 Porovnání metod po změně systému odměn . . . 38
4.3 Analýza strategií sázení a akcí . . . 40

5 Závěr **44**

Seznam obrázků

3.1 Průměrná odměna v závislosti na základní jednotce 24
3.2 Průměrný počet kol v závislosti na základní jednotce 24

4.1 Srovnání průměrné odměny v závislosti na počtu her 37
4.2 Srovnání průměrného počtu kol v závislosti na počtu her 38
4.3 Srovnání průměrné odměny v závislosti na počtu her při změněném systému odměn 39
4.4 Srovnání průměrného počtu kol v závislosti na počtu her při změněném systému odměn 40
4.5 Srovnání strategie sázení 41
4.6 Srovnání průměrného počtu pojištění v závislosti na počtu her 42
4.7 Srovnání strategie akcí 43

Seznam tabulek

2.1 Hodnoty karet v Hi-Lo systému 7
2.2 Tabulka rozhodnutích Hit nebo Stand pro hráče s hard hand . 9
2.3 Tabulka rozhodnutích Hit nebo Stand pro hráče s soft hand . . 9
2.4 Tabulka rozhodnutích Double pro hráče s hard hand 9
2.5 Tabulka rozhodnutích Double down pro hráče s soft hand . . . 10
2.6 Tabulka rozhodnutích Split pro hráče 10

4.1 Tabulka srovnání strategií sázení a akcí 41

Seznam výpisů kódu

3.1 Generování hash klíče pro fázi, stav a akci pomocí MD5 v Pythonu 27

3.2 Funkce Q-learningu pro výběr sázky na základě hodnoty epsilon v Pythonu 29
3.3 Výběr vzorků z prioritní replay paměti v Pythonu 31
3.4 Konstrukce dueling DQN sítě v Pythonu 33
3.5 Implementace vrstvy `NoisyLinear` pro parametrický šum . . . 35

Seznam zkratek

RL	Reinforcement learning
DQN	Deep Q-Network, resp. Deep Q-learning
Hi-Lo	High-Low systém
MDP	Markov Decision Process (Markovův rozhodovací proces)
LMDB	Lightning Memory-Mapped Database
TD error	Temporal Difference error

Kapitola 1
Úvod

Hazardní hry představují oblast, která již po staletí přitahuje pozornost matematiků, statistiků i běžných hráčů. Jednou z těchto her je i karetní hra blackjack, která se díky svým relativně jednoduchým pravidlům a částečně deterministickému charakteru stala populárním objektem pro analýzu a vývoj herních strategií. Přestože kasino ve většině případů disponuje výhodou, existují techniky, které mohou tuto výhodu snížit, případně ji dokonce obrátit ve prospěch hráče.

Tato práce se zabývá srovnáním tří přístupů ke hře blackjack, které mají potenciál zvýšit šance hráče na dlouhodobý zisk. Prvním z nich je klasická metoda počítání karet, která využívá znalosti o rozdaných kartách k odhadnutí výhodnosti následujících herních rozhodnutí. Druhým přístupem je metoda Q-learning reprezentující oblast posilovaného učení, kde se agent učí optimální strategii prostřednictvím interakce s prostředím. Třetí metodou je Deep Q-learning, která kombinuje principy Q-learningu s hlubokými neuronovými sítěmi a umožňuje řešit složitější úlohy s rozsáhlejším stavovým prostorem.

Cílem této práce je porovnat uvedené metody z hlediska jejich efektivity, schopnosti přizpůsobení se různým herním situacím a jejich potenciálu k dosažení dlouhodobé výhody nad kasinem. Důraz je kladen na objektivní vyhodnocení jednotlivých přístupů prostřednictvím simulací, přičemž závěrem práce je určení, zda některá z metod dokáže překonat výhodu kasina a přinést hráči statistickou výhodu v dlouhodobém horizontu.

Kapitola 2
Teoretická část

Cílem této kapitoly je poskytnout přehled teoretických základů, které budou sloužit jako rámec pro porovnání metod posilovaného učení v kontextu hry blackjack. V následujících podkapitolách se zaměříme na klíčové pojmy a principy, které jsou pro tuto problematiku zásadní. Nejprve se budeme věnovat samotné hře blackjack, následně se zaměříme na metody, které mohou být použity k analýze a zlepšení rozhodovacích strategií v této hře, konkrétně na počítání karet. Poté přejdeme k obecnému pojetí posilovaného učení a podrobně probereme dvě konkrétní techniky – Q-learning a Deep Q-learning, které jsou aplikovatelné na strategii hraní blackjacku.

2.1 Blackjack

Blackjack je populární karetní hra provozovaná v kasinech, která se hraje mezi jednotlivými hráči a krupiérem. Její jednoduchá pravidla, kombinace strategie a štěstí, a také možnosti pro hráče ovlivnit výsledky hry, činí tuto hru nejen oblíbenou mezi hráči, ale také předmětem mnoha studií a analýz. Mezi první systematické práce patří slavný článek Baldwina a kolegů z roku 1956, který jako první matematicky definoval optimální strategii pro hráče na základě pravděpodobnostních výpočtů [1]. Na tuto práci navázal například Millman (1983), který provedl statistickou analýzu herních výsledků a ukázal, jak drobné odchylky od optimální strategie mohou ovlivnit dlouhodobé výnosy [2]. Taktické aspekty hry otevírají prostor pro vývoj různých strategií, které mohou významně zvýšit pravděpodobnost úspěchu hráče [3, 4].

2.1.1 Pravidla hry

Jako každá hra, i blackjack má svá unikátní pravidla, která definují průběh a strukturu samotné hry. Tato pravidla jsou relativně jednoduchá, avšak jejich pochopení je klíčové pro úspěšné zapojení se do hry a pro aplikaci efektivních

strategií. Pravidla se mohou mírně lišit v závislosti na konkrétní variantě blackjacku, ale základní mechanismy a principy zůstávají konzistentní. V této sekci se podrobněji zaměříme na klíčová pravidla blackjacku tak, jak jsou formulována v knize Beat the Dealer od Edwarda Thorpa, přičemž vycházíme z části věnované právě pravidlům a hernímu rámci [5].

Cíl hráče

Cílem každého hráče je získat kombinaci karet s hodnotou co nejblíže 21, aniž by hráč překročil tuto hodnotu. Důležité je, že samotný cíl nespočívá pouze v dosažení hodnoty 21, ale především v poražení krupiéra. I když má hráč součet karet blízký 21, pokud je hodnota karet krupiéra vyšší, hráč prohrává, bez ohledu na sílu své vlastní ruky.

Počet hráčů

Ve většině kasin poskytujících blackjack může u jednoho stolu hrát 1 až 7 hráčů. I když je za stolem více účastníků, hráči nehrají proti sobě, ale pouze proti krupiérovi, který reprezentuje kasino. Každý hráč se snaží dosáhnout co nejlepšího výsledku v souboji s krupiérem, přičemž výsledky jednotlivých hráčů nejsou vzájemně závislé. To znamená, že i na plně obsazeném stole má každý hráč stejnou šanci na výhru, nezávisle na ostatních účastnících.

Obecně však platí, že čím méně hráčů u stolu hraje, tím výhodnější to může být pro jednotlivého hráče, což vyplývá z analýz uvedených v kapitolách [6] a [7].

Balíček karet

Počet balíčků použitých ve hře se může lišit v závislosti na variantě blackjacku a pravidlech konkrétního kasina. Nejčastěji se hraje se 4 až 8 standardními balíčky po 52 kartách, i když existují i verze využívající pouze jeden nebo dva balíčky. Počet balíčků ovlivňuje pravděpodobnosti jednotlivých výsledků a také efektivitu některých strategií – například počítání karet je snazší při menším počtu balíčků, zatímco více balíčků zvyšuje výhodu kasina.

Aby se zabránilo předvídání karet, balíčky se pravidelně míchají, a to buď ručně, pomocí automatických míchacích strojů, nebo kontinuálních míchaček, které průběžně vracejí použité karty zpět do hry.

Hodnoty karet jsou pevně stanovené: karty 2 až 10 odpovídají své číselné hodnotě, obrázkové karty (J, Q, K) mají hodnotu 10 a eso může mít hodnotu 1 nebo 11, podle toho, co je pro hráče výhodnější. Kombinace, ve které je eso počítáno jako 11, se nazývá *soft hand* (měkká kombinace), protože hráč si může vzít další kartu bez rizika okamžitého přetažení přes 21. Pokud má hráč kombinaci bez esa, nebo musí být eso počítáno jako 1, jedná se o *hard hand* (tvrdou kombinaci), kde je třeba hrát opatrněji, aby nedošlo k překročení 21. Tyto rozdíly ovlivňují strategická rozhodnutí během hry a mohou mít významný dopad na výsledek jednotlivých kol.

Průběh hry

Hra začíná tím, že každý hráč u stolu i krupiér obdrží dvě karty. Zatímco karty hráče jsou obvykle obě viditelné, krupiér má jednu kartu otočenou lícem nahoru a druhou skrytou. Na základě hodnoty svých karet se hráč rozhoduje,

jakou akci provede, přičemž hlavním cílem je dosáhnout součtu co nejbližšího hodnotě 21, aniž by ji překročil.

Během hry má hráč několik možností. Může si vzít další kartu (*hit*), zůstat se současným součtem (*stand*), zdvojnásobit sázku a vzít si pouze jednu další kartu (*double down*) nebo, pokud má dvě stejné karty, rozdělit je do dvou samostatných herních kombinací (*split*). V některých případech může také složit karty a ztratit pouze polovinu sázky (*surrender*), pokud pravidla kasina tuto možnost umožňují.

Pokud hráč dosáhne součtu 21 hned po rozdání prvních dvou karet a má kombinaci esa a karty s hodnotou 10, získává blackjack (*natural*). Tento výsledek obvykle přináší výplatu v poměru 3:2, což je vyšší než standardní výhra. Naopak, pokud hodnota hráčových karet přesáhne 21, dochází k *bustu* (přetažení) a hráč automaticky prohrává bez ohledu na karty krupiéra.

Po skončení tahů všech hráčů krupiér odkryje svou druhou kartu a podle stanovených pravidel táhne další karty. Obvykle musí táhnout, pokud má součet 16 nebo méně, a stát, pokud má 17 nebo více. Hra končí porovnáním součtu karet hráče a krupiéra. Hráč vyhrává, pokud má vyšší součet než krupiér, nebo pokud krupiér dosáhne *bustu*. Pokud má krupiér vyšší hodnotu, hráč prohrává. V případě shody se sázka vrací hráči.

Sázení

Na začátku každého kola hráči umístí své sázky na určené místo na stole. Minimální a maximální limity sázek se liší podle kasina a konkrétního stolu, přičemž některé stoly jsou určeny pro hráče s nízkými sázkami, zatímco jiné jsou přizpůsobeny pro vysoké sázky (*high roller*). Po umístění sázek krupiér rozdá karty a hra pokračuje podle standardních pravidel.

Během hry mohou hráči v určitých situacích navýšit svou sázku. Například možnost double down umožňuje zdvojnásobit původní sázku výměnou za jednu jedinou další kartu. Při rozdělení karet (*split*) je nutné vsadit stejnou částku na obě nové kombinace. Některé varianty hry nabízejí také pojištění (*insurance*), což je vedlejší sázka dostupná tehdy, pokud má krupiér jako první kartu eso. Hráč může vsadit až polovinu své původní sázky, a pokud má krupiér skutečně blackjack, pojištění se vyplácí v poměru 2:1. V opačném případě hráč o pojištěnou částku přichází.

Po ukončení kola se podle výsledku sázky buď vyplácejí, nebo hráč o svou sázku přichází. Standardní výhra se obvykle vyplácí v poměru 1:1, zatímco blackjack přináší výplatu 3:2. V případě, že dojde k remíze, sázka zůstává u hráče.

2.2 Počítání karet

Počítání karet představuje jednu z nejznámějších strategií využívaných hráči při hře blackjack s cílem získat matematickou výhodu nad kasinem (viz [8]). Tato technika spočívá v systematickém sledování již rozdaných karet a následném odhadu pravděpodobnosti, zda budou zbývající karty ve hře pro hráče

výhodné či nevýhodné. Hráči, kteří tuto strategii úspěšně ovládají, mohou výrazně zvýšit své šance na úspěch, neboť získávají přesnější představu o možném složení zbylého balíčku [8].

2.2.1 Princip

Princip počítání karet spočívá v tom, že hráč sleduje, které karty již byly rozdány, a na základě těchto informací se snaží odhadnout, jaké karty zůstali v balíčku. V blackjacku jsou některé karty pro hráče výhodnější než jiné. Například karty s hodnotou 10 (10, J, Q, K) a esa zvyšují šance hráče na dosažení blackjacku (tj. součtu 21), zatímco nízké karty (2 až 6) obvykle zvyšují šance krupiéra a snižují pravděpodobnost výhry hráče [9]. Systémy počítání karet se proto soustředí na sledování poměru vysokých a nízkých karet, které byly již odehrány, a na základě těchto údajů hráči upravují své sázky a herní strategii.

Tímto způsobem může hráč zvýšit své šance na výhru, avšak za cenu nutnosti neustálého udržování přesného přehledu o rozdaných kartách. To vyžaduje značnou míru soustředění, disciplíny a praxe. Zároveň je třeba počítat s tím, že kasina aktivně sledují neobvyklé vzorce sázení a mohou při podezření z počítání karet hráči omezit hru nebo jej dokonce zcela vykázat z herního prostoru.

2.2.2 Historie a vývoj metody

Metoda počítání karet se začala vyvíjet v 50. letech 20. století. Prvním známým matematikem, který se věnoval výpočtu pravděpodobnosti v blackjacku, byl Edward O. Thorp, profesor matematiky z MIT, který v roce 1962 publikoval knihu *Beat the Dealer* [10]. V této knize popsal první systematický přístup k počítání karet, který vycházel z teorie pravděpodobnosti a statistiky. Thorpova metoda se stala základem pro další systémy počítání karet.

V průběhu let byly vyvinuty různé varianty a systémy počítání karet, které se snažily zjednodušit nebo zefektivnit původní metodu, přičemž každý nový systém měl své specifické výhody a nevýhody. ak popisují Vidámi v [11], metody se stávaly sofistikovanějšími a umožňovaly hráčům stále lepší analýzu herní situace.

2.2.3 Systémy

Existuje několik různých systémů počítání karet, z nichž každý má své vlastní výhody a nevýhody. Některé jsou jednodušší na použití, ale méně přesné, zatímco jiné jsou složitější, ale mohou hráči poskytnout větší výhodu.

- **Hi-Lo systém** – Hi-Lo systém je základní metodou počítání karet. Přiřazované hodnoty karet jsou:
  - Karty 2 až 6: +1

- Karty 7 až 9: 0
- Karty 10, J, Q, K, A: -1

Tento systém je efektivní pro základní přehled o stavu hry. Je široce používaný mezi začátečníky i pokročilými hráči [12].

- **KO (Knock-Out) systém** – Varianta Hi-Lo systému, ale nevyžaduje konverzi na skutečné počítání (*true count*). Hodnoty karet jsou:
  - Karty 2 až 7: +1
  - Karty 8 a 9: 0
  - Karty 10, J, Q, K, A: -1

  Tento systém spočívá v jednoduchosti, protože hráč nemusí provádět úpravy podle zbývajících balíčků [13].
- **Omega II systém** – Pokročilejší systém, který přiřazuje kartám různé hodnoty, což umožňuje přesnější odhad výhodnosti hry:
  - Karty 4, 5, 6: +2
  - Karty 2, 3, 7: +1
  - Karty 8 a esa: 0
  - Karta 9: -1
  - Karty 10, J, Q, K: -2

  Tento systém vyžaduje více soustředění, protože hráč musí být schopen rychle spočítat různé hodnoty pro různé karty. Systém dává více bodů kartám, které jsou pro hráče výhodné (např. karty 4, 5 a 6), a více penalizuje karty, které jsou pro hráče nevýhodné (např. karty 10, J, Q, K) [11].
- **Zen Count systém** – Středně složitý systém s následujícími hodnotami:
  - Karty 2, 3, 7: +1
  - Karty 4, 5, 6: +2
  - Karta 9: -1
  - Karty 10, J, Q, K: -2
  - Eso: 0 (může být sledováno samostatně)

  Tento systém je oblíbený mezi pokročilými hráči, protože poskytuje lepší přesnost při odhadu výhodnosti balíčku. Využívá speciální přiřazení hodnot k jednotlivým kartám a poskytuje tak více informací pro úpravu sázek [14].

- **Halves systém** – Pokročilý systém, který přiřazuje kartám zlomkové hodnoty:
  - Karty 2 a 7: +0.5
  - Karty 3, 4 a 6: +1
  - Karta 5: +1.5
  - Karta 9: -0.5
  - Karty 10, J, Q, K: -1

  Tento systém nabízí velmi přesné výsledky, ale je náročný na zapamatování a rychlé počítání v reálném čase [11].

2.2.4 Hi-Lo systém

Jak bylo uvedeno v předchozí části, Hi-Lo systém patří mezi nejjednodušší a současně nejrozšířenější metody počítání karet v blackjacku. Jeho úkolem je zvýšit šance na výhru hráče za pomocí matematické analýzy poměru vysokých a nízkých karet, které byly již rozdány. Tento systém umožňuje hráči lépe odhadnout, jaké karty mají pravděpodobnost být rozdány v následujících kolech. Díky tomu může hráč upravit svou strategii sázek, kdy zvyšuje sázky v případech, kdy je v balíčku více vysokých karet, a naopak snižuje sázky, když je pravděpodobnost vyššího výskytu nízkých karet větší. Cílem je maximalizovat výhody v těch okamžicích, kdy je pravděpodobnost výhry na straně hráče, a tím zvýšit jeho šance na dlouhodobý zisk.

Pro přehlednost je, jak uvádí Thorpe V [12], je níže uvedena tabulka přiřazovaných hodnot kartám v Hi-Lo systému:

Hodnota karty	Přiřazená hodnota
2, 3, 4, 5, 6	+1
7, 8, 9	0
10, J, Q, K, A	-1

Tabulka 2.1 Hodnoty karet v Hi-Lo systému

Pro výpočet Hi-Lo indexu použijeme následující vzorec z [12]:

$$\text{Hi-Lo index} = \left(\frac{\text{Počet}}{\text{Skryté karty}}\right) \times 100\% \tag{2.1}$$

Pro stanovení výše sázky na základě Hi-Lo indexu používáme následující pravidla. Jakmile hráč vypočítá Hi-Lo index, může podle něj určit, jakou sázku si zvolí. Čím vyšší je Hi-Lo index, tím vyšší sázku je doporučeno vsadit.

Výše sázky se, podle pravidel popsaných v [12], určuje takto:

- Pokud je Hi-Lo index menší než 4, doporučená sázka je 1 základní jednotka (tzv. *unit*).
- Pokud je Hi-Lo index menší než 6, doporučená sázka je 2 základní jednotky (2 *units*).
- Pokud je Hi-Lo index menší než 8, doporučená sázka je 3 základní jednotky (3 *units*).
- Pokud je Hi-Lo index menší než 10, doporučená sázka je 4 základní jednotky (4 *units*).
- Pokud je Hi-Lo index větší nebo rovný 10, doporučená sázka je 5 základních jednotek (5 *units*).
- Alternativně je doporučeno sázet polovinu hodnoty indexu, pokud není přísně stanoveno pravidlo pro konkrétní hodnotu indexu.

Základní jednotka (*unit*) je částka, kterou hráč obvykle sází na jednu hru v případě, že Hi-Lo index je nízký (např. 1 *unit*). Sázky jsou pak upravovány podle výše Hi-Lo indexu, což znamená, že při vyšších hodnotách indexu je doporučeno sázet více.

Pokud je Hi-Lo index větší než 8, doporučuje se vzít pojištění, protože vysoký Hi-Lo index naznačuje, že v balíčku je větší počet vysokých karet, což zvyšuje pravděpodobnost, že krupiér skutečně dostane blackjack [12].

Během hry v Hi-Lo systému se hráč řídí tabulkami s akcemi podle indexu, který je určen aktuálním počtem. Tyto tabulky určují, zda má hráč provést akce jako *hit* (táhnout kartu), *stand* (zůstat), *double down* (dvojnásobit sázku) nebo *split* (rozdělit dvojici). Akce, kterou hráč provede, závisí na hodnotě indexu, která odráží situaci v balíčku a pomáhá maximalizovat šance na výhru. Tabulky zároveň zohledňují specifické situace, jako jsou *soft hand* (karty hráče obsahující eso), *hard hand* (karty hráče neobsahují eso nebo eso se může počítat pouze jako 1) nebo stejný pár karet, což hráčům umožňuje přizpůsobit jejich rozhodnutí podle konkrétního stavu hry [12]. Níže jsou uvedeny rozhodovací tabulky, včetně popisů a specifikací jednotlivých herních situací, které mohou během hry nastat.

Suma karet hráče	2	3	4	5	6	7	8	9	10	A
18 a více	Stand									
17	Stand									-15
16	-21	-23	-30	-34	-35	10	11	6	2	14
15	-12	-17	-21	-26	-28	13	15	12	8	16
14	-5	- 8	-13	-17	-17	20	38	Hit		
13	1	-2	-5	-9	-8	50	Hit			
12	14	6	2	-1	0	Hit				

- Pokud je Hi-Lo index větší než číslo v tabulce, zůstaň stát (stand). V opačných případech, pokud je index menší nebo rovný, táhni kartu (hit).

Tabulka 2.2 Tabulka rozhodnutích Hit nebo Stand pro hráče s hard hand

Suma karet hráče	2	3	4	5	6	7	8	9	10	A
19 a více	Stand									
18	Stand							Hit	12	-6
17	Hit					29	Hit			

- Zůstaň stát (stand), pokud máš Hi-Lo index větší než číslo v tabulce. V ostatních případech (pokud je Hi-Lo index menší nebo rovný číslu v tabulce), táhni kartu (hit).
- Pokud máš součet karet menší rovno 16, táhni kartu (hit).

Tabulka 2.3 Tabulka rozhodnutích Hit nebo Stand pro hráče s soft hand

Suma karet hráče	2	3	4	5	6	7	8	9	10	A
11	-23	-26	-29	-33	-35	-26	-16	-10	-9	-3
10	-15	-17	-21	-24	-26	-17	-9	-3	7	6
9	3	0	-5	-10	-12	4	14			
8		22	11	5	5	22				
7		45	21	14	17					
6			27	18	24					
5				20	26					

- Zdvojnásob sázku (Double down), pokud je Hi-Lo index větší než číslo v tabulce.

Tabulka 2.4 Tabulka rozhodnutích Double pro hráče s hard hand

Karty hráče	2	3	4	5	6
A,9		20	12	8	8
A,8		9	5	1	0
A,7		-2	-15	-18	-23
A,6	*	-8	-14	-28	-30
A,5		21	-6	-16	-32
A,4		19	-7	-16	-32
A,3		11	-8	-13	-19
A,2		10	2	-19	-13

- Zdvojnásob sázku (Double down), pokud je Hi-Lo index větší než číslo v tabulce.
- *Pokud je index Hi-Lo větší než 1 a menší 10, pak zdvojnásob sázku (double down).

Tabulka 2.5 Tabulka rozhodnutích Double down pro hráče s soft hand

Karty hráče	2	3	4	5	6	7	8	9	10	A
A,A						-33	-24	-22	-20	-17
10,10	25	17	10	6	7	19				
9,9	-3	-8	-10	-15	-14	8	-16	-22		10
8,8									24*	-18
7,7	-22	-29	-35							
6,6	0	-3	-8	-13	-16	-8				
5,5										
4,4		18	8	0						
3,3	-21	-34					6*			
2,2	-9	-15	-22	-30						

- Pokud je Hi-Lo index větší než číslo v tabulce, rozděl hru (split pair).
- *Pokud má hráč dvojici 8 proti krupiérově 10, rozděl hru (split pair) v případě, že Hi-Lo index je menší než 24.
- Pokud má hráč dvojici 3 proti krupiérově 8, rozděl hru (split pair) v případě, že Hi-Lo index je větší než 6 a menší než -2.

Tabulka 2.6 Tabulka rozhodnutích Split pro hráče

2.3 Posilované učení

Posilované učení (angl. *Reinforcement Learning*, RL) představuje oblast strojového učení, ve které se agent učí optimálním rozhodnutím na základě interakce s prostředím. Cílem agenta je maximalizovat kumulativní odměnu skrze sled akcí. Na rozdíl od učení s učitelem (*supervised learning*), kde má model k dispozici správné výstupy pro dané vstupy, v RL se agent učí prostřednictvím zpětné vazby z prostředí, která nemusí být okamžitá ani jednoznačná, jak uvádějí Kaelbling, Littman a Moore ve své přehledové práci [15].

V této sekci budou představeny základní principy posilovaného učení, včetně klíčových pojmů jako jsou agent, prostředí, stav, akce, odměna a politika. Dále budou popsány konkrétní algoritmy využívané v této práci, které se aplikují na problém hry blackjack. Blackjack slouží jako vhodné testovací prostředí, neboť nabízí kombinaci náhody a strategie, a zároveň umožňuje srovnávat různé RL metody na dobře definovaném úkolu.

Cílem této sekce je představit teoretické základy posilovaného učení, které slouží jako východisko pro praktickou část práce. Uvedené poznatky jsou nezbytné pro implementaci metod popsaných v sekcích 3.3 a 3.4, které se věnují algoritmům Q-learning a Deep Q-learning.

2.3.1 Základy posilovaného učení

Posilované učení je jedním z klíčových směrů strojového učení, ve kterém se agent přizpůsobuje na základě zkušeností získaných při svém působení v prostředí. V rámci tohoto paradigmatu agent postupně buduje strategii (tzv. politiku), která mu umožňuje vybírat akce vedoucí k dosažení co nejpříznivějších výsledků. Ty jsou měřeny prostřednictvím odměn, které prostředí agentovi přiděluje v reakci na jeho chování [16].

Ve vztahu k ostatním přístupům strojového učení se posilované učení vyznačuje specifickým způsobem získávání informací: správnost jednání agenta není známa předem, ale je nepřímo odvozována ze zpětné vazby poskytnuté prostředím. Tím se odlišuje od učení s učitelem (*supervised learning*), které pracuje s trénovacími daty obsahujícími jednoznačně definované vstupy a výstupy, i od učení bez učitele (*unsupervised learning*), které se zaměřuje na analýzu dat bez předem daného ohodnocení [15].

Historie posilovaného učení sahá až do 50. let 20. století, kdy byly položeny první teoretické základy učení založeného na zpětné vazbě [16]. S postupným rozvojem výpočetní techniky a algoritmických metod nabýval tento přístup na významu, zejména v kontextu vývoje umělé inteligence. V současnosti je posilované učení využíváno v celé řadě aplikačních domén, včetně řízení agentů ve strategických hrách, dále při ovládání robotických systémů v reálném čase a také při řešení úloh autonomní navigace, například u samořídicích vozidel operujících ve složitých a dynamicky se měnících prostředích.

2.3.2 Formální definice problému posilovaného učení

Jak již bylo uvedeno, posilované učení je obvykle definováno jako proces, v němž agent interaguje s prostředím a optimalizuje svou politiku na základě zkušeností získaných prostřednictvím interakcí. Formálně je úkolem agenta nalézt optimální politiku π^*, která maximalizuje očekávanou kumulativní odměnu. Tento úkol je definován jako řešení následující optimalizační úlohy:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$$

Optimalizace politiky může být realizována pomocí různých algoritmů, mezi které patří metody založené na hodnotách (například algoritmus hodnotového iterování nebo Q-learning) a metody založené na politice (například metoda politické iterace nebo politika gradientních metod) [16].

Posilované učení se tedy zaměřuje na nalezení politiky, která umožňuje agentovi efektivně a dlouhodobě maximalizovat svou odměnu v prostředí, které je často nejisté a dynamické.

Pro správné pochopení posilovaného učení je klíčové se seznámit s jeho základními komponentami, které definují interakci mezi agentem a prostředím. Následující definice, převzaté z práce Suttona a Barta [16], objasňují hlavní prvky tohoto procesu, které tvoří základ pro formulaci optimalizační úlohy.

▶ **Definice 2.1** (Agent, prostředí, stav, akce, odměna)**.** *V rámci posilovaného učení se definují následující komponenty:*

- ***Agent**: Entita, která se rozhoduje a vykonává akce v daném prostředí.*
- ***Prostředí**: Svět, ve kterém agent operuje a interaguje s ním.*
- ***Stav**: Reprezentace aktuální situace v prostředí.*
- ***Akce**: Volby, které agent může v daném stavu vykonat.*
- ***Odměna**: Kvantitativní zpětná vazba, kterou agent získává po provedení akce v daném stavu.*

Abychom mohli efektivně modelovat a analyzovat problémy v posilovaném učení, je nutné použít formální rámec, který umožní kvantifikovat a popsat dynamiku interakcí mezi agentem a prostředím. Tento rámec je obvykle poskytován Markovovými rozhodovacími procesy (MDP), které umožňují matematicky formulovat úlohy rozhodování v prostředí s nejistotou a dynamickými změnami. MDP modeluje rozhodování agenta na základě informací o aktuálním stavu, akcích, které může vykonat, pravděpodobnostech přechodů mezi stavy a odměnách, které jsou agentovi přiřazeny v reakci na jeho chování.

▶ **Definice 2.2** (Markovův rozhodovací proces (MDP))**.** *Markovův rozhodovací proces je definován 5-ticí:*

$$M = \langle S, A, P, R, \gamma \rangle$$

kde:

- S *je množina stavů v prostředí.*
- A *je množina akcí, které může agent vykonat.*
- $P(s'|s, a)$ *je přechodová funkce, která udává pravděpodobnost přechodu do stavu* s' *po provedení akce* a *ve stavu* s*.*
- $R(s, a)$ *je funkce odměny, která udává okamžitou odměnu, kterou agent získá po vykonání akce* a *v stavu* s*.*
- γ *je diskontní faktor, který určuje, jakou váhu agent přiřazuje budoucím odměnám ve srovnání s okamžitými odměnami, přičemž* $0 \leq \gamma \leq 1$*.*

Na základě výše uvedené definice MDP lze konstatovat, že tento rámec umožňuje přesně modelovat dynamiku prostředí a rozhodovací proces agenta. Klíčovou vlastností MDP je tzv. Markovova vlastnost, která říká, že pravděpodobnost přechodu do dalšího stavu závisí pouze na aktuálním stavu a akci, nikoli na předchozí historii. Tato vlastnost výrazně zjednodušuje analýzu a vývoj algoritmů posilovaného učení, protože umožňuje formulovat problém optimalizace politiky jako úlohu dynamického programování nebo aproximace hodnotových funkcí.

2.3.3 Algoritmy posilovaného učení

Mezi nejpoužívanější algoritmy posilovaného učení patří zejména Q-learning, SARSA (State-Action-Reward-State-Action), Monte Carlo metody a metody založené na aproximaci hodnotových funkcí. Každý z těchto algoritmů má své specifické vlastnosti, výhody a nevýhody v závislosti na tom, zda agent disponuje kompletními informacemi o prostředí a jakým způsobem se agent učí na základě získaných odměn [16].

Q-learning představuje široce používaný algoritmus, který využívá tabulkovou reprezentaci hodnotové funkce akce, kde agent iterativně aktualizuje hodnoty pro každou akci v každém stavu. Tento algoritmus je *off-policy*, což znamená, že politika, která se používá k výběru akcí, se může lišit od politiky, kterou agent optimalizuje.

SARSA, naopak, je *on-policy* algoritmus, což znamená, že aktualizace hodnoty akce je prováděna na základě akcí, které agent ve skutečnosti vykonává podle aktuální politiky.

Monte Carlo metody se zaměřují na učení hodnoty jednotlivých stavů na základě dlouhých epizod a celkových zisků, které agent během hry dosáhne.

Tento přístup se ukázal jako užitečný v blackjacku, kde agent získává odměny až po dokončení celé hry [15].

Metody využívající aproximaci hodnotových funkcí se uplatňují v případech, kdy je prostředí příliš složité na tabulkovou reprezentaci. Umožňují agentovi efektivně pracovat i s rozsáhlými stavovými prostory a budou podrobněji popsány v následující části [16].

2.3.4 Průzkum vs. využití (Exploration vs. Exploitation)

Jedním z klíčových problémů v posilovaném učení je dilema mezi průzkumem (*exploration*) a využitím (*exploitation*). Průzkum znamená, že agent se rozhoduje pro akce, které nemusí být okamžitě optimální, ale umožňují mu získat nové informace o prostředí. Využití znamená, že agent vybírá akce, které podle aktuální politiky vedou k co největšímu zisku [16].

V blackjacku je tento problém obzvláště důležitý, protože pokud agent stále využívá stejnou politiku bez průzkumu nových možností, může se stát, že uvízne v suboptimálním chování, které nevede k dosažení nejlepší možné strategie. Na druhou stranu, pokud agent nadměrně průzkum provádí, může se zdržet dosažení stabilní a optimální politiky.

Pro řešení tohoto problému se často využívají různé přístupy, jako je *epsilon-greedy* strategie, kde agent s pravděpodobností ϵ provede náhodnou akci (průzkum), a s pravděpodobností $1 - \epsilon$ provede akci podle současné politiky (využití).

2.3.5 Aproximace hodnotových funkcí

V mnoha případech, zejména v situacích s velkým počtem stavů a akcí, není možné uložit všechny hodnoty do tabulky. V takových případech se používají metody aproximace hodnotových funkcí, které umožňují agentovi pracovat s vysokodimenzionálními prostory.

Aproximace hodnotových funkcí může být prováděna různými způsoby, například lineární aproximací, kde se hodnoty stavů nebo akcí modelují jako lineární kombinace funkcí stavů. Dalšími pokročilejšími technikami jsou například metody založené na neuronových sítích, které umožňují flexibilnější a přesnější modelování hodnotových funkcí. Takové metody jsou zvlášť užitečné v komplexních prostředích, kde není možné explicitně uložit hodnoty pro všechny možné stavy a akce.

V blackjacku je možné využít aproximaci hodnotové funkce k urychlení učení a zlepšení výkonu agenta, zejména v případě, že agent není schopen prozkoumat všechny možné stavy nebo když se hraje více her současně [16].

2.4 Q-learning

Q-learning je jednou z nejpoužívanějších metod posilovaného učení [16]. Tuto metodu původně představil Watkins ve své disertační práci [17] jako bezmodelový (*model-free*) algoritmus, který umožňuje agentovi osvojit si optimální politiku jednání ve zcela neznámém prostředí pouze na základě interakcí s tímto prostředím.

2.4.1 Základní principy

Cílem agenta je naučit se optimální politiku $\pi^* : S \to A$, která maximalizuje očekávaný kumulativní zisk (součet diskontovaných odměn). Q-learning se snaží přímo aproximovat optimální akční hodnotovou funkci $Q^*(s, a)$, která udává očekávaný výnos při vykonání akce a ve stavu s a následném pokračování dle optimální politiky [16].

Hlavní výhodou Q-learningu je, že je bezmodelový (*model-free*) – agent nepotřebuje znát přechodové pravděpodobnosti mezi stavy ani přesnou podobu odměnové funkce [17]. Učení probíhá výhradně na základě interakcí s prostředím, tedy pozorováním přechodů stavů a obdržených odměn.

2.4.2 Q-funkce a Bellmanova rovnice

Optimální akční hodnotová funkce $Q^*(s, a)$ vyjadřuje maximální očekávaný výnos, kterého lze dosáhnout z daného stavu s vykonáním akce a a následným pokračováním optimální politikou. Tato funkce je řešením tzv. Bellmanovy optimální rovnice, která stanoví rekurzivní vztah mezi hodnotou aktuální akce a hodnotami budoucích akcí [16]:

$$Q^*(s, a) = \mathbb{E}_{s'} \left[R(s, a) + \gamma \max_{a'} Q^*(s', a') \right], \tag{2.2}$$

kde $\gamma \in [0, 1]$ je diskontní faktor určující význam budoucích odměn a $R(s, a)$ je očekávaná okamžitá odměna za akci a ve stavu s. Operátor očekávání $\mathbb{E}_{s'}$ je brán přes rozdělení přechodových pravděpodobností $P(s' \mid s, a)$, které popisují dynamiku prostředí.

2.4.3 Algoritmus Q-learning

Q-learning algoritmus aktualizuje odhad Q-funkce iterativně při každé interakci s prostředím podle následující aktualizační rovnice [17]:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right], \tag{2.3}$$

kde:

- s_t je aktuální stav,
- a_t je zvolená akce,
- r_{t+1} je obdržená odměna,
- s_{t+1} je nový stav po vykonání akce,
- $\alpha \in (0, 1]$ je učící rychlost (*learning rate*).

Termín $r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a')$ představuje cílovou hodnotu (*target*), která odhaduje výnos dosažitelný z nového stavu s_{t+1} při optimálním rozhodování. Rozdíl mezi touto hodnotou a stávajícím odhadem $Q(s_t, a_t)$ je znám jako temporální rozdíl (*temporal-difference error*), jenž slouží jako zpětná vazba pro učení.

2.4.4 Pseudokód Q-learningu

Níže je uveden základní pseudokód Q-learning algoritmu, jak jej uvádějí Sutton a Barto [16]:

Algorithm 1 Q-learning

Inicializuj $Q(s, a)$ pro všechny stavy $s \in S$ a akce $a \in A$ (např. na nulu)
for každou epizodu **do**
    Zvol počáteční stav s
    while s není terminální stav **do**
        Vyber akci a pomocí ε-greedy politiky z $Q(s, a)$
        Proveď akci a, pozoruj odměnu r a nový stav s'
        Aktualizuj:

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

        $s \leftarrow s'$
    end while
end for

2.4.5 Explorace vs. exploatace

Důležitou součástí Q-learningu je kompromis mezi *explorací* (zkoumáním nových akcí) a *exploatací* (využíváním již známých informací). Nejčastěji se používá ε-greedy strategie, kdy s pravděpodobností ε je vybrána náhodná akce a s pravděpodobností $1 - \varepsilon$ je vybrána akce s nejvyšší hodnotou $Q(s, a)$ [16].

2.4.6 Konvergence

Q-learning je zaručeně konvergentní k optimální akční hodnotové funkci $Q^*(s, a)$ za určitých podmínek, jako je dostatečný počet návštěv všech stavů a akcí a postupné snižování učící rychlosti α. Tento proces probíhá i bez nutnosti znalosti modelu prostředí, což činí Q-learning vhodným pro problémy, kde není k dispozici explicitní přechodová matice nebo funkce odměn [18].

V praxi se však často používají varianty Q-learningu, které přistupují k aproximaci hodnotové funkce pomocí neuronových sítí, jako je Deep Q-learning.

2.5 Deep Q-learning

Klasický Q-learning se ukázal jako účinný u problémů s malým a diskrétním stavovým i akčním prostorem. V případě rozsáhlejších prostředí (např. při zpracování vizuálních vstupů nebo spojitých stavových prostorů) se však tabulková reprezentace Q-funkce stává neefektivní a paměťově náročnou. Řešením tohoto problému je použití aproximace Q-funkce pomocí neuronových sítí, což vede ke vzniku metody **Deep Q-learning** (DQN). Algoritmus je podrobně analyzován ve studii [19] Volodymyra Mniha a jeho kolegů.

2.5.1 Hlavní myšlenka

Metoda DQN nahrazuje tradiční tabulkovou reprezentaci $Q(s, a)$ neuronovou sítí s parametry θ, která slouží jako aproximátor akční hodnotové funkce:

$$Q(s, a; \theta) \approx Q^*(s, a). \tag{2.4}$$

Vstupem neuronové sítě je reprezentace stavu s, výstupem je vektor Q-hodnot pro všechny možné akce v daném stavu. Síť je trénována tak, aby minimalizovala rozdíl mezi predikovanou Q-hodnotou a cílovou hodnotou, podobně jako v klasickém Q-learningu [20].

2.5.2 Ztrátová funkce

Cílem trénování neuronové sítě je minimalizace ztrátové funkce $L(\theta)$, která měří rozdíl mezi predikovanou hodnotou $Q(s, a; \theta)$ a cílovou hodnotou (*target*), danou modifikovanou Bellmanovou rovnicí:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim D}\left[(y - Q(s, a; \theta))^2\right], \tag{2.5}$$

kde (s, a, r, s') je jeden přechod uložený v paměti zkušeností D (replay buffer) a cílová hodnota y je definována jako:

$$y = r + \gamma \max_{a'} Q(s', a'; \theta^-), \tag{2.6}$$

a θ^- jsou parametry tzv. cílové sítě (*target network*), které se periodicky kopírují z hlavní sítě, aby se stabilizoval trénink [19].

2.5.3 Zlepšení stability učení

Původní Q-learning se při použití neuronové sítě potýká s několika problémy, jako je nestabilita nebo divergence učení. DQN zavádí následující techniky pro zajištění stability [20]:

- **Experience replay** – místo okamžitého učení z jednoho přechodu (s, a, r, s') se přechody ukládají do paměťového bufferu D a při tréninku se náhodně vzorkují mini-batche. Tím se eliminuje korelovanost dat a zlepšuje efektivita učení [21].
- **Target network** – používá se samostatná cílová síť s parametry θ^-, která se aktualizuje méně často než hlavní síť. Tím se snižuje oscilace cílových hodnot [20].
- **Clipped error (Huber loss)** – při výpočtu chyby mezi predikovanou a cílovou Q-hodnotou se místo klasické kvadratické ztráty používá Huberova ztrátová funkce. Ta je méně citlivá na odlehlé hodnoty a pomáhá stabilizovat učení [22].
- **Gradient clipping** – omezuje velikost gradientů během zpětné propagace na pevně daný rozsah, čímž zabraňuje výbuchu gradientů a numerické nestabilitě [22].
- **Double Q-learning** – zavádí rozdělení výběru a vyhodnocení akce do dvou sítí, čímž redukuje nadhodnocování Q-hodnot. Výběr akce se provádí pomocí hlavní sítě, ale hodnota se odhaduje pomocí cílové sítě [20].
- **Prioritized experience replay** – přechody v paměťovém bufferu nejsou vzorkovány rovnoměrně, ale na základě důležitosti (např. velikosti TD-chyby), což umožňuje efektivnější učení z klíčových situací [23].
- **Dueling network architecture** – architektura, která rozděluje výpočet Q-funkce na dvě větve: jednu pro odhad hodnoty stavu $V(s)$ a druhou pro odhad výhody akce $A(s, a)$. Výsledná Q-hodnota se získá spojením těchto dvou komponent. To umožňuje efektivnější učení v situacích, kdy je výběr akce méně důležitý [20].
- **Noisy linear layers** – místo klasické explorace pomocí ϵ-greedy strategie zavádí stochastické změny přímo do parametrů neuronové sítě pomocí náhodných šumových složek ve váhách lineárních vrstev. To umožňuje učením řízenou exploraci a může vést ke stabilnějším a rychlejšímu konvergenci [24].

Tyto techniky jsou zásadní pro úspěšné použití hlubokých neuronových sítí v oblasti zesíleného učení a výrazně přispívají ke stabilitě a konvergenci trénovacího procesu.

2.5.4 Algoritmus Deep Q-learning

Níže je zobrazen pseudokód pro implementaci algoritmu Deep Q-learning, inspirovaný prací Gu et al. [25]:

Algorithm 2 Deep Q-learning

Inicializuj replay buffer D
Inicializuj Q-síť s náhodnými vahami θ
Inicializuj cílovou síť s váhami $\theta^- = \theta$
for každou epizodu **do**
    Inicializuj počáteční stav s
    while s není terminální **do**
        Vyber akci a pomocí ε-greedy politiky z $Q(s, a; \theta)$
        Proveď akci a, pozoruj odměnu r a nový stav s'
        Ulož přechod (s, a, r, s') do replay bufferu D
        Náhodně vzorkuj mini-batch (s_j, a_j, r_j, s'_j) z D
        Pro každý prvek mini-batche spočítej cílovou hodnotu:

$$y_j = r_j + \gamma \max_{a'} Q(s'_j, a'; \theta^-)$$

        Minimalizuj ztrátu:

$$L = \frac{1}{N} \sum_j (y_j - Q(s_j, a_j; \theta))^2$$

        $s \leftarrow s'$
        Periodicky aktualizuj cílovou síť: $\theta^- \leftarrow \theta$
    end while
end for

2.5.5 Výhody a nevýhody

Hlavní výhodou DQN je schopnost generalizovat ve velkých a spojitých stavových prostorech, což jej činí vhodným pro komplexní problémy jako je řízení agentů v simulovaných herních prostředích (např. Atari), řízení robotických systémů nebo autonomní navigace [19].

Mezi nevýhody patří vyšší výpočetní náročnost a nutnost pečlivého ladění hyperparametrů (velikost bufferu, velikost mini-batche, frekvence aktualizace cílové sítě, atd.).

Kapitola 3
Implementace

V této kapitole se zaměříme na praktickou implementaci jednotlivých metod, které byly teoreticky popsány v předchozích kapitolách. Cílem je vytvořit simulační prostředí hry blackjack, ve kterém budou implementovány a následně testovány tři různé přístupy pro optimalizaci herní strategie hráče. Každý z těchto přístupů bude reprezentován specifickým agentem, který bude samostatně operovat v rámci daného prostředí. Simulace bude implementována v programovacím jazyce Python[1], s důrazem na modularitu, přehlednost a možnost snadného rozšiřování či opětovného použití jednotlivých komponent.

Základ simulace tvoří prostředí blackjacku, které musí přesně zachytit pravidla hry, včetně možností hráčových akcí, způsobu rozdávání karet, chování krupiéra i vyhodnocování výsledků. Na tomto základě budou vytvořeni tři agenti, z nichž každý bude implementovat odlišný přístup ke zlepšování své strategie.

První agent bude založen na heuristické metodě počítání karet Hi-Lo, která využívá jednoduché aritmetiky k odhadování výhodnosti další hry. Tento agent bude rozhodovat na základě aktuálního stavu počítadla a své strategie, která je předem definována a pevně daná.

Druhý agent bude implementovat algoritmus Q-learning, který představuje metodu učení s posilováním. Tento agent se bude učit optimální strategii postupným hraním her, přičemž bude aktualizovat hodnoty Q-funkce, která mapuje stav a akci na očekávanou budoucí odměnu.

Třetí agent bude rozšířením Q-learningového přístupu pomocí hluboké neuronové sítě, známe jako Deep Q-learning. Tento agent nahradí explicitní Q-tabuli aproximačním modelem v podobě neuronové sítě, čímž umožní práci s rozsáhlejším a spojitějším stavovým prostorem, což je zvláště výhodné pro komplexní úlohy jako blackjack.

Tento rámec slouží jako základ pro experimenty, jejichž cílem bude porovnání a hodnocení výkonu jednotlivých agentů v prostředí blackjacku. Tyto

[1] `https://docs.python.org/3/`

experimenty budou zaměřeny na ověření efektivity různých přístupů a analýzu schopnosti optimalizovat herní strategii hráče v reálných herních podmínkách.

3.1 Herní prostředí - Blackjack

Herní prostředí bylo implementováno v jazyce Python pomocí objektově orientovaného přístupu. Cílem bylo vytvořit simulaci karetní hry blackjack, kterou lze využít jak pro klasickou interakci s hráčem, tak i pro testování agentů využívajících strojové učení. Pravidla blackjacku jsou detailně popsána v sekci 2.1.1, přičemž konkrétní modifikace a úpravy oproti standardním pravidlům jsou rozepsány níže v této sekci.

3.1.1 Struktura aplikace

Aplikace je rozdělena do několika tříd, z nichž každá reprezentuje konkrétní komponentu hry:

- `Card` – Reprezentuje jednu hrací kartu, která je definována hodnotou (*rank*) a barvou (*suit*).
- `Deck` – Reprezentuje balíček karet. Umožňuje táhnutí karet, sledování již odehraných karet a případné promíchání balíčku.
- `Dealer` – Reprezentuje krupiéra. Řídí se předdefinovaným chováním dle pravidel blackjacku.
- `BlackjackGame` – Hlavní třída, která spravuje průběh celé hry. Zajišťuje interakci mezi hráči, krupiérem a balíčkem karet.
- `Player` – Reprezentuje hráče, včetně jeho sázek, financí, ruky a rozhodovací logiky.

Tyto třídy spolu vzájemně komunikují a umožňují plynulý průběh hry. Každá třída je zodpovědná za specifickou část herní logiky, což zajišťuje modularitu a přehlednost kódu.

3.1.2 Průběh hry

Celý herní proces je řízen metodou `start_game()`, která představuje hlavní smyčku hry a opakuje se, dokud hráči chtějí pokračovat. Tato metoda postupně volá tři klíčové fáze každého kola:

1. `place_bets()` – Hráči vkládají sázky.
2. `play_round()` – Probíhá celé jedno kolo hry: rozdání karet, tahy hráčů a krupiéra, vyhodnocení výsledků.

   a. `deal_initial_cards()` – Hráčům i krupiérovi jsou rozdány počáteční karty.
   b. `offer_insurance(dealer_card, player)` – Hráčům je poskytnuta možnost uzavřít pojištění.
   c. `player_turn(player)` – Hráči provádí své tahy (*hit*, *stand*, *double down*, *split*).
   d. `dealer_turn()` – Krupiér provádí své tahy podle definovaných pravidel.
3. `reset_hands()` – Dojde k vyčištění rukou hráčů a krupiéra před dalším kolem.

3.1.3 Herní logika

- **Vkládání sázek:** Před začátkem každého kola hráči vkládají své sázky. Bez aktivní sázky se hráč daného kola neúčastní.
- **Rozdání karet:** Každému hráči a krupiérovi jsou rozdány dvě karty. Jedna karta krupiéra zůstává skrytá.
- **Pojištění:** Pokud krupiérova odkrytá karta je eso, hráči mají možnost uzavřít pojištění proti tomu, že krupiér dosáhne blackjacku. V případě, že krupiér Blackjack skutečně má, pojištění hráči vyplatí kompenzaci.
- **Tah hráče:** Hráč má k dispozici základní volby jako *Hit*, *Stand*, *Double Down* a *Split*, pokud to situace umožňuje.
- **Tah krupiéra:** Krupiér táhne karty až do dosažení minimální hodnoty 17 bodů.
- **Vyhodnocení:** Hodnoty hráčů jsou porovnány s hodnotou krupiéra. Zohledňuje se výhra dosažená *Blackjackem* a případné pojištění (*Insurance*).

3.1.4 Správa balíčku karet

V základní konfiguraci se používá 6 balíčků. Jakmile počet zbývajících karet klesne pod 104 (tj. ekvivalent dvou balíčků), dojde k automatickému promíchání balíčku. Tento proces zajišťuje metoda `reshuffle()`, která využívá funkci `random.shuffle()`[2] k přeskupení karet pomocí pseudonáhodného algoritmu. Přestože tento algoritmus není skutečně náhodný, protože je deterministicky závislý na počátečním stavu generátoru, poskytuje dostatečnou úroveň náhodnosti pro herní účely. Promíchání lze proto považovat za férové.

[2] `https://docs.python.org/3/library/random.html#random.shuffle`

3.1.5 Sledování odehraných karet

Třída `Deck` uchovává hodnoty již odehraných karet v proměnné `used_cards`. Tato funkcionalita je vhodná např. pro implementaci strategií jako počítání karet nebo pro využití při trénování modelů umělé inteligence.

3.1.6 Interakce s hráčem nebo agentem

Rozhodování hráče je abstrahováno metodou `play_action()` v třídě `Player` nebo `Agent`. Tím je umožněno snadno přepínat mezi manuálním ovládáním hry a automatizovaným učením.

3.2 Agent používající Hi-Lo systém

Implementace agenta pro počítání karet podle systému Hi-Lo vychází ze zásad určených v sekci 2.2.4. Cílem je využít Hi-Lo skóre pro optimalizaci sázek a herních rozhodnutí ve hře blackjack. Klíčovým prvkem je výpočet Hi-Lo skóre, které hodnotí šance na výskyt vysokých nebo nízkých karet v balíčku na základě již rozdání karet.

Tento systém je realizován třídou `HiLoAgent`, která je odvozena od základní třídy `Player`. Třída `HiLoAgent` implementuje metody, které umožňují agentovi počítat karty a optimalizovat jeho sázení a herní rozhodnutí.

3.2.1 Výpočet Hi-Lo skóre

Hi-Lo skóre se počítá na základě již odehraných karet. Každá karta je ohodnocena podle následující tabulky:

- Karty 2 až 6: +1
- Karty 7 až 9: 0
- Karty 10, J, Q, K, A: −1

Součet těchto hodnot tvoří tzv. běžné skóre (*running count*). Aby bylo skóre přizpůsobeno velikosti balíčku, přepočítává se na tzv. *true count* vydělením odhadovaným počtem zbývajících karet v balíčku. V této implementaci je skóre navíc vynásobeno 100 pro zvýraznění rozdílů:

$$\text{Hi-Lo skóre} = \left(\frac{\text{Počet nízkých karet} - \text{Počet vysokých karet}}{\text{Počet zbývajících karet}} \right) \times 100$$

Toto skóre se poté využívá při rozhodování o výši sázky, při zvažování pojištění a také pro úpravu základní strategie hraní.

3.2.2 Hlavní metody

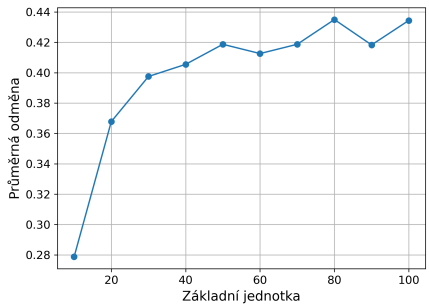
1. place_bet Tato metoda se používá k určení výše sázky na základě Hi-Lo skóre. Sázka se přizpůsobuje počtu nevyužitých karet v balíčku. Pokud je skóre nižší, agent sází méně, pokud je vyšší, sází více. Čím více nízkých karet (2–6) bylo rozdáno, tím vyšší je šance na vysoké karty, což agentovi umožňuje zvýšit sázku. Sázka je limitována počtem žetonů, které agent má.

2. place_insurance Tato metoda se používá k rozhodnutí, zda si agent vezme pojištění při výskytu esa u krupiéra. Pokud je Hi-Lo skóre dostatečně vysoké, agent si pojištění vezme, jinak ne. Pojištění je výška poloviny základní sázky.

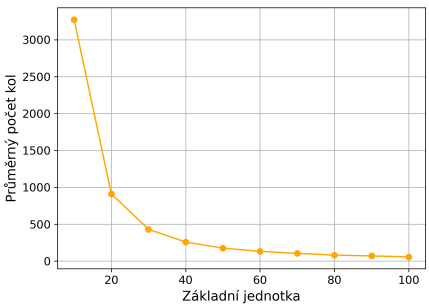
3. play_action Tato metoda určuje, jaká akce by měla být provedena v závislosti na kartách hráče a krupiéra, a to v kombinaci s Hi-Lo skóre. Zahrnuje akce jako rozdělení, stát, nebo brát další kartu. V některých situacích, jako je rozdělení es, agent upravuje své rozhodnutí podle Hi-Lo skóre, což znamená, že se rozhoduje na základě pravděpodobnosti, že v balíčku zůstávají vysoké karty.

3.2.3 Výběr základní jednotky

Pro efektivní fungování Hi-Lo agenta je klíčové zvolit vhodnou základní sázkovou jednotku, která ovlivňuje průběh i výsledek hry. Abychom identifikovali optimální hodnotu této jednotky, provedli jsme sérii testů, ve kterých jsme porovnávali chování agenta při různých hodnotách této jednotky. V rámci testu agent obdržel odměnu +1, pokud dokázal navýšit svůj kapitál z 1000 na 2000 žetonů, a odměnu 0 v případě bankrotu. Podrobně je tento odměňovací systém popsán v sekci 4.1.



Obrázek 3.1 Průměrná odměna v závislosti na základní jednotce



Obrázek 3.2 Průměrný počet kol v závislosti na základní jednotce

Na základě výsledků těchto testů jsme jako základní jednotku zvolili hodnotu 100, jelikož dosahovala nejlepších výsledků z hlediska pravděpodobnosti úspěchu.

3.3 Agent používající Q-learning

Implementace algoritmu Q-learning byla realizována ve formě třífázového rozhodovacího agenta v prostředí karetní hry blackjack. Tento agent si v průběhu hraní buduje Q-tabule pro tři klíčové fáze hry: **sázení**, **pojištění** a **hraní**. Agent postupně zlepšuje svá rozhodnutí na základě zkušeností získaných ze zpětné vazby.

3.3.1 Struktura Q-tabulek

Pro každou fázi rozhodování (sázení, pojištění, hraní) je použita samostatná Q-tabule, která je reprezentována jako databáze LMD[3]. Tento přístup umožňuje efektivní správu a načítání Q-hodnot v průběhu učení, přičemž každý záznam je uložen pod unikátním klíčem tvořeným kombinací fáze, stavu a akce.

Původně byly Q-tabule implementovány pomocí `defaultdict`[4], což je datová struktura v Pythonu, která umožňuje jednoduché a efektivní uchovávání hodnot i pro neexistující klíče. Tento přístup však vedl k problémům s přetížením operační paměti, zejména při větším počtu uložených hodnot v případě komplexních herních stavů.

Poté byla zvolena alternativa v podobě databáze SQLite[5], která umožňuje ukládat data na disk a tím eliminovat problémy s paměťovými nároky. Tento přístup byl ale příliš pomalý, zejména při častém zápisu a čtení dat. Pro řešení tohoto problému byla nakonec zvolena LMDB (Lightning Memory-Mapped Database), která nabízí vysoký výkon při práci s velkými objemy dat a efektivně řeší problém paměťové náročnosti, zároveň poskytuje rychlý přístup k uloženým Q-hodnotám.

Každá fáze tedy má svou vlastní Q-tabuli, která je uložena v samostatné LMDB databázi. Tento přístup zajišťuje efektivní a škálovatelné řešení pro učení agenta v různých fázích hry.

3.3.2 Fáze činnosti agenta

Agent je navržen tak, aby v průběhu hry vykonával rozhodnutí ve třech základních fázích: **sázení**, **pojištění** a **hraní**. Každá z těchto fází je reprezentována samostatnými funkcemi, které se zaměřují na rozhodovací proces v dané fázi hry.

- **Sázení** - Fáze sázení je realizována těmito funkcemi:
  - `get_betting_state` určuje stav sázení na základě počtu chipů a použité karty,

[3] `https://lmdb.readthedocs.io/en/release/`
[4] `https://docs.python.org/3/library/collections.html#collections.defaultdict`
[5] `https://docs.python.org/3/library/sqlite3.html`

- `choose_bet_action` rozhoduje o výši sázky pomocí *epsilon-greedy* strategie,
- `update_q_table_betting` aktualizuje Q-hodnotu pro daný stav-akci na základě odměny a nového stavu.

- **Pojištění** - V této fázi agent rozhoduje, zda si vezme pojištění proti blackjacku krupiéra. Fáze pojištění je realizována těmito funkcemi:
  - `get_insurance_state` vytváří stav, který zahrnuje hodnotu karet hráče a krupiéra, počet es a výši sázky,
  - `choose_insurance_action` vybírá akci (pojištění nebo ne) na základě hodnoty Q pro daný stav,
  - `update_q_table_insurance` aktualizuje Q-hodnoty pro akci pojištění.
- **Hraní** - Hlavní fáze hry, kdy agent rozhoduje, zda si vezme kartu, zůstane stát nebo učiní jinou akci. Fáze hraní je implementována pomocí těchto funkcí:
  - `get_playing_state` generuje stav hraní, který zahrnuje součet karet hráče, počet es, hodnotu karty krupiéra a výši sázky,
  - `choose_play_action` vybírá nejlepší akci na základě Q-hodnoty pro daný stav,
  - `update_q_table_playing` upravuje Q-hodnoty podle výsledku hry a získané odměny.

Každá fáze je tedy tvořena specifickými funkcemi, které se zaměřují na rozhodování a aktualizaci Q-hodnot pro danou část hry. Agent postupně vylepšuje svou strategii díky učení na základě odměn získaných v průběhu hry.

3.3.3 Předzpracování dat

Před samotným učením a aktualizací Q-hodnot je nutné vhodně připravit vstupní data, tedy stavy a akce, aby mohly být konzistentně uloženy a zpětně vyhledávány v Q-tabulkách. V rámci jednotlivých fází hry jsou stavy upravovány tak, aby byly jednoduše reprezentovatelné a zároveň poskytovaly dostatek informací pro rozhodování agenta.

Jedním z příkladů úprav je zaokrouhlování počtu žetonů hráče na nejbližší stovky. Tím se výrazně snižuje množství unikátních stavů a Q-tabule zůstává přehledná a efektivní. Kromě žetonů jsou do stavu zahrnovány i informace o použitých kartách, součtu hodnot karet hráče, počtu es a aktuální sázce.

Každá kombinace fáze, upraveného stavu a akce je následně převedena na hashovaný klíč pomocí funkce `md5` z knihovny `hashlib`[6]. Klíč se vytváří jako textový řetězec:

[6] `https://docs.python.org/3/library/hashlib.html`

```
raw_key = f"{phase}_{state}_{action}"
key = hashlib.md5(raw_key.encode()).digest()
```

Výpis kódu 3.1 Generování hash klíče pro fázi, stav a akci pomocí MD5 v Pythonu

Výsledný hash `key` slouží jako jednoznačný identifikátor v databázi LMDB, kde jsou Q-hodnoty ukládány a načítány. Díky tomuto postupu lze efektivně pracovat i s rozsáhlým prostorem stavů, aniž by bylo potřeba držet vše v paměti nebo používat složité struktury.

Tento přístup k předzpracování dat zajišťuje rychlé vyhledávání, konzistenci dat a zároveň zjednodušuje proces učení ve všech částech hry.

3.3.4 Q-funkce

Q-funkce, známá také jako funkce akční hodnoty, je klíčovým prvkem v algoritmu Q-learning. Určuje, jak dobrá je určitá akce v daném stavu, což agentovi umožňuje rozhodovat se o nejlepších akcích na základě dosavadního učení. Q-hodnota pro konkrétní kombinaci stavu a akce je určena následujícím způsobem:

$$Q(s,a) = Q(s,a) + \alpha \left(r + \gamma \cdot \max_{a'} Q(s',a') - Q(s,a) \right)$$

kde:

- s je aktuální stav,
- a je vybraná akce,
- r je získaná odměna,
- γ je faktor diskontování (charakterizuje, jak moc agent zohledňuje budoucí odměny),
- $Q(s',a')$ je maximální Q-hodnota pro následující stav s' a akci a'.

V rámci implementace se funkce `update_q_table_playing` používá pro aktualizaci Q-hodnoty při hraní, kdy agent provádí rozhodnutí na základě aktuálního stavu. Funkce provádí následující kroky:

1. **Načtení staré hodnoty Q**: Nejprve se načte stará hodnota Q pro aktuální stav a akci pomocí funkce `_get_q_value`.
2. **Výpočet budoucí hodnoty Q**: Dále se spočítá maximální hodnota Q pro následující stav a všechny možné akce. Tento krok simuluje očekávaný zisk z nejlepší možné akce v budoucnosti.

3. **Penalizace odměny**: Pro zajištění regulace učení a zamezení přetížení Q-tabule v pozdějších fázích trénování se do procesu zavádí penalizace. Penalizace je definována jako:

$$\text{penalization} = \exp\left(-k \cdot \text{num\_rounds}\right)$$

kde k je konstanta závislá na parametru α a *num_rounds* je počet odehraných kol. Penalizace postupně klesá s rostoucím počtem kol, což znamená, že agent se stává méně závislý na minulých rozhodnutích a více se soustředí na aktuální situaci. Tato penalizace je aplikována na odměnu, což znamená, že starší odměny mají menší váhu v rozhodování.

4. **Aktualizace Q-hodnoty**: Konečně se stará Q-hodnota aktualizuje na základě nové odměny, budoucí hodnoty Q a rozdílu mezi těmito hodnotami. Nová hodnota Q je počítána podle vzorce:

$$Q_{\text{new}}(s,a) = Q_{\text{old}}(s,a) + \alpha \cdot (\text{penalized\_reward} + \gamma \cdot \text{future\_value} - Q_{\text{old}}(s,a))$$

kde α je rychlost učení, která určuje, jak rychle agent reaguje na nové informace, a γ je faktor diskontování, který zohledňuje budoucí odměny.

Penalizace v tomto případě slouží k tomu, aby agent neupřednostňoval příliš dlouhé sekvence akcí a místo toho usiloval o dosažení cíle v co nejkratším možném čase. Tato penalizace motivuje agenta k hledání strategií, které vedou k rychlému dosažení odměny, a zabraňuje přílišnému protahování procesu, čímž se zajišťuje efektivní učení a optimalizace chování. Parametry jako α (rychlost učení) a γ (faktor diskontování) ovlivňují, jak rychle agent reaguje na nové informace a jak velký význam přikládá budoucím odměnám.

3.3.5 Metoda *epsilon-greedy*

Při výběru akcí agent používá metodu *epsilon-greedy*, která zajišťuje rovnováhu mezi objevováním nových akcí (*exploration*) a využíváním znalostí z předchozího učení (*exploitation*).

V každé fázi rozhodování (sázení, pojištění, hraní) probíhá výběr akce následovně:

- S pravděpodobností `epsilon` se akce vybere náhodně ze všech možných.
- Jinak se vybere akce s nejvyšší Q-hodnotou pro daný stav.

Hodnota `epsilon` se na začátku nastaví na určitou počáteční hodnotu (např. 1.0) a postupně se po každé iteraci snižuje pomocí násobení s parametrem `epsilon_decay`. Zároveň je omezená minimální hodnotou `epsilon_min`, aby agent stále občas zkoušel i nové akce.

Příklad z implementace výběru sázky:

```
if random.uniform(0, 1) < self.epsilon_betting:
    return random.choice(possible_bets)
else:
    return max(possible_bets, key=lambda bet:
    ↪  self._get_q_value("betting", state, bet))
```

Výpis kódu 3.2 Funkce Q-learningu pro výběr sázky na základě hodnoty epsilon v Pythonu

Analogicky se metoda používá i pro výběr akce při pojištění a hraní. Pro každou fázi existuje vlastní hodnota `epsilon` a její samostatné řízení (`epsilon_decay`, `epsilon_min`), což umožňuje nezávisle upravovat průběh učení v jednotlivých částech hry.

Tato strategie je jednoduchá na implementaci a zároveň velmi efektivní pro trénování agenta v prostředí s velkým množstvím možných akcí.

3.4 Agent používající Deep Q-learning

Poslední implementovaná metoda využívá DQN, který kombinuje klasický Q-learning s hlubokými neuronovými sítěmi. Tento přístup umožňuje efektivně řešit rozhodovací problémy s rozsáhlým nebo spojitým stavovým prostorem, kde tradiční tabulkové metody selhávají. V našem případě se jedná o nadstavbu třífázového agenta založeného na Q-learningu, který je implementován v sekci 3.3. Díky DQN je agent schopen postupně zlepšovat svou strategii na základě zpětné vazby z prostředí a optimalizovat svá rozhodnutí v jednotlivých fázích hry.

3.4.1 Architektura agenta

Pro každou fázi hry – **sázení**, **pojištění** a **hraní** – agent využívá dvě neuronové sítě a jednu paměť (replay memory). Jedna síť je primární Q-síť, která se používá pro volbu akcí a učení, a druhá je tzv. target síť, která poskytuje stabilnější cílové Q-hodnoty během trénování.

Ve fázi **sázení** je stav reprezentován pomocí počtu žetonů hráče a počtu již odehraných karet v balíčku (rozlišujeme deset hodnot karet). Ve fázi **pojištění** stav zahrnuje hodnotu hráčovy kombinace, počet es, hodnotu karty krupiéra, aktuální sázku, počet žetonů a znovu rozložení karet v balíčku. Pro fázi **hraní** je stav složen obdobně jako u pojištění, ale navíc reflektuje specifickou sázku aktuální ruky, protože hráč může mít více rukou kvůli rozdělení karet (*split*).

Trénovací metoda **replay** náhodně vybírá mini-batch zkušeností z paměti, což umožňuje efektivnější a stabilnější učení, jelikož snižuje korelaci mezi následnými zkušenostmi a umožňuje opětovné využití dřívějších situací. Pomocí

těchto zkušeností se aktualizuje primární Q-síť s cílem minimalizovat rozdíl mezi aktuální odhadovanou Q-hodnotou a cílovou hodnotou, která se počítá na základě target sítě.

Použití **target experience replay** je klíčové pro stabilizaci učení. Target síť je kopie hlavní Q-sítě, která se aktualizuje méně často. Tím se zabrání oscilacím a divergence během tréninku, protože cílové hodnoty Q-learningu zůstávají konzistentnější během několika iterací. Tento přístup umožňuje efektivnější konvergenci Q-funkce.

3.4.2 Replay memory

Paměť pro opakované přehrávání (*Replay memory*) slouží jako zásobník dříve nasbíraných zkušeností agenta, které jsou následně opakovaně využívány při trénování neuronové sítě. Tento mechanismus přispívá ke zvýšení datové efektivity učení a k jeho stabilitě. V této implementaci je replay paměť rozšířena o **prioritní výběr zkušeností (Prioritized Experience Replay)**, který umožňuje efektivnější učení na základě významnosti jednotlivých přechodů.

Každá uložená zkušenost má formát:

$$(\text{stav}, \text{akce}, \text{odměna}, \text{nový stav}, \text{dokončeno}),$$

kde:

- `stav` představuje aktuální pozorování prostředí,
- `akce` je akce, kterou agent v daném stavu provedl,
- `odměna` je číselné ohodnocení výsledku akce,
- `nový stav` je pozorování prostředí po provedené akci,
- `dokončeno` je binární indikátor, zda epizoda skončila.

Paměť je implementována jako prioritní buffer s omezenou kapacitou. Každé zkušenosti je přiřazena priorita, která je typicky odvozena z velikosti TD-chyby (*temporal difference error*). Tato priorita určuje pravděpodobnost, s jakou bude zkušenost vybrána při trénování.

Naše konkrétní implementace se řídí schématem představeným v [23], kde pravděpodobnost výběru zkušenosti i je definována vztahem:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha},$$

kde p_i je priorita zkušenosti i a α určuje míru vlivu priorit (při $\alpha = 0$ se výběr stává uniformním).

Abychom mohli kompenzovat zkreslení způsobené nerovnoměrným výběrem, zavedli jsme se váhování ztráty pomocí *importance sampling* koeficientu w_i, jak je rovněž navrženo v [23]::

$$w_i = \left(\frac{1}{N \cdot P(i)} \right)^{\beta},$$

kde N je velikost paměti a β je parametr, který roste v průběhu tréninku směrem k 1.

Nové zkušenosti jsou do paměti vkládány s maximální prioritou, což jim zajišťuje vysokou pravděpodobnost výběru bezprostředně po uložení. Pokud je paměť naplněna, dochází k odstranění nejstarších záznamů. Po každém trénovacím kroku jsou priority použitých zkušeností aktualizovány na základě nově spočítané TD-chyby.

Následující výňatek z implementace třídy `ReplayBuffer` ilustruje postup výběru vzorků na základě priorit a výpočet odpovídajících *importance sampling* vah:

```
def sample(self, batch_size):
    self.beta = min(1.0, self.beta + self.beta_increment)
    priorities = self.priorities[:self.size]
    probabilities = priorities ** self.alpha
    probabilities /= probabilities.sum()
    indices = np.random.choice(self.size, batch_size, p=probabilities,
    ↪  replace=False)
    samples = [self.buffer[idx] for idx in indices]
    weights = (self.size * probabilities[indices]) ** (-self.beta)
    weights /= weights.max()
    return samples, indices, weights
```

Výpis kódu 3.3 Výběr vzorků z prioritní replay paměti v Pythonu

3.4.3 Modely neuronových sítí

V rámci implementace jsou využívány pokročilé neuronové sítě s **dueling architekturou**, **noisy lineárními vrstvami** a **normalizací vrstev** (Layer Normalization), které jsou navrženy pro každou fázi hry samostatně – konkrétně pro sázení, pojištění a hraní. Každá z těchto sítí je reprezentována dvojicí: hlavní (primární) Q-sítí a tzv. *target* sítí. Obě sítě sdílí stejnou architekturu, přičemž target síť je aktualizována s nižší frekvencí, což přispívá ke stabilitě trénovacího procesu.

Primární Q-síť je zodpovědná za predikci Q-hodnot pro všechny možné akce na základě aktuálního stavu prostředí. Trénována je na základě zkušeností uložených v replay paměti, přičemž jejím cílem je minimalizovat rozdíl mezi odhadovanou Q-hodnotou a cílovou hodnotou, která je spočtena pomocí target

sítě. Aktualizace parametrů Q-sítě probíhá po každém trénovacím kroku a tato síť se využívá i pro výběr akcí.

Target síť je synchronizovaná kopie Q-sítě, sloužící výhradně k výpočtu stabilních cílových Q-hodnot během trénování. Její parametry jsou aktualizovány méně často, čímž dochází k omezení fluktuací v cílových hodnotách a ke zvýšení stability procesu učení. Tato technika, známá jako *target experience replay*, je běžně využívána v architekturách založených na hlubokém Q-učení.

Pro implementaci neuronových sítí je využita knihovna **PyTorch**[7], která poskytuje flexibilní prostředí pro návrh, trénování a ladění hlubokých modelů. Díky dynamickému výpočetnímu grafu umožňuje efektivní experimentování s různými architekturami. Navíc podporuje výpočty na GPU, což výrazně urychluje trénovací proces.

Architektura jednotlivých sítí je navržena takto:

- **Vstupní vrstva**: Lineární vrstva převádějící vstupní vektor reprezentující aktuální stav do latentního prostoru. Následuje normalizace výstupu a aktivační funkce GELU.
- **Skryté vrstvy**: Celkem 5 vrstev s dimenzemi [128, 256, 384, 256, 128]. Sestávají se z posloupnosti plně propojených vrstev doplněných o normalizaci a aktivační funkce. Mezi vybrané vrstvy jsou vložena reziduální spojení pro zlepšení gradientního toku. Architektura tak kombinuje výhody hlubokých reprezentací a stability trénování.
- **Dueling architektura**: Výstupní blok je rozdělen na dvě větve: hodnotovou a výhodovou. Obě větve využívají `NoisyLinear` vrstvy, které umožňují vkládat do trénovacího procesu parametrický šum, čímž dochází ke zlepšení průzkumu prostoru stavů a akcí.
  - **Value stream**: Skládá se ze dvou vrstev - první `NoisyLinear` převádí vstup na `hidden_dims`$[-1]//2$ neuronů, druhá produkuje skalární odhad hodnoty stavu.
  - **Advantage stream**: Tvoří jej také dvě vrstvy - první `NoisyLinear` převádí vstup na `hidden_dims`$[-1]//2$ neuronů, druhá produkuje výhodu pro každou možnou akci.

Tato síťová architektura neuronové sítě umožňuje efektivní aproximaci akční hodnotové funkce prostřednictvím hluboké reprezentace a pokročilých stabilizačních technik, jako jsou normalizace vrstev, dueling architektura a využití cílové sítě.

[7] `https://pytorch.org/docs/stable/index.html`

Níže je uvedena implementace struktury sítě:

```
class DuelingNoisyDQNNetwork(nn.Module):
    def __init__(self, input_dim, output_dim, hidden_dims=[128, 256,
    ↪  384, 256, 128]):
        super(DuelingNoisyDQNNetwork, self).__init__()

        self.input_layer = nn.Linear(input_dim, hidden_dims[0])
        self.layer_norm1 = nn.LayerNorm(hidden_dims[0])

        self.hidden_layers = nn.ModuleList()
        self.layer_norms = nn.ModuleList()

        for i in range(len(hidden_dims)-1):
            self.hidden_layers.append(nn.Linear(hidden_dims[i],
            ↪  hidden_dims[i+1]))
            self.layer_norms.append(nn.LayerNorm(hidden_dims[i+1]))

        self.value_noisy = NoisyLinear(hidden_dims[-1],
        ↪  hidden_dims[-1] // 2)
        self.value_out = NoisyLinear(hidden_dims[-1] // 2, 1)

        self.advantage_noisy = NoisyLinear(hidden_dims[-1],
        ↪  hidden_dims[-1] // 2)
        self.advantage_out = NoisyLinear(hidden_dims[-1] // 2,
        ↪  output_dim)

        self._initialize_weights()
```

Výpis kódu 3.4 Konstrukce dueling DQN sítě v Pythonu

3.4.4 Q-funkce

Q-funkce, nebo také funkce akce-hodnota, je klíčovým prvkem v metodách učení posilováním, konkrétně v Q-learningu. Q-funkce $Q(s, a)$ přiřazuje každému stavu s a každé možné akci a hodnotu, která odhaduje dlouhodobou odměnu, kterou agent získá, pokud vykoná akci a ve stavu s, a poté bude následovat optimální politiku. Cílem je maximalizovat tuto hodnotu pro každou možnou akci, což umožňuje agentovi vybrat nejlepší akci v daném stavu.

V rámci našeho agenta používáme Q-funkci pro určení hodnoty jednotlivých akcí v každé fázi hry. Q-funkce je aproximována neuronovou sítí, což znamená, že místo explicitního výpočtu hodnoty pro každou kombinaci stavu a akce používáme model, který tuto hodnotu odhaduje na základě tréninkových dat.

- **Learning rate** ($\alpha = 0.001$): Určuje, jak rychle se model přizpůsobuje novým zkušenostem. Nižší hodnota znamená pomalejší, ale stabilnější učení.

- **Discount factor** ($\gamma = 0.99$): Určuje důležitost budoucích odměn. Vyšší hodnota znamená, že agent preferuje dlouhodobé zisky.
- **Exploration rate** (ϵ): Řídí rovnováhu mezi explorací a exploatací. Hodnota ϵ se obvykle snižuje během tréninku, čímž agent postupně upřednostňuje získané znalosti před náhodným chováním.

Použitá architektura sítě je založena na dueling DQN přístupu. Místo přímého odhadu Q-hodnoty každé akce se nejprve odhaduje hodnota stavu $V(s)$ a výhoda akce $A(s, a)$, které jsou následně zkombinovány podle vzorce:

$$Q(s,a) = V(s) + \left(A(s,a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s,a') \right),$$

kde $\mathcal{A}$ značí množinu všech možných akcí. Tento přístup umožňuje efektivnější učení, jelikož síť se může lépe soustředit na odhad hodnoty samotného stavu a relativních výhod jednotlivých akcí.

Q-funkce je trénována pomocí metody *target experience replay*, jak bylo popsáno v předchozích sekcích, kde je cílová Q-hodnota aktualizována na základě hodnoty z target sítě.

3.4.5 Explorace pomocí parametrického šumu

Místo tradiční metody *epsilon-greedy*, která náhodně vybírá akce podle parametru ϵ, využíváme pokročilejší techniku explorace pomocí **Noisy Networks**. Tato metoda vnáší do rozhodovacího procesu adaptivní šum přímo v rámci neuronové sítě, konkrétně v `NoisyLinear` vrstvách.

Každá NoisyLinear vrstva zavádí do vah a biasů parametrický šum, který se při trénování náhodně resetuje. Díky tomu je rozhodování agenta stochastické, stejné vstupy mohou vést k odlišným výstupům, což zajišťuje průzkum prostředí bez nutnosti ručně nastavovat a ladit parametr ϵ.

Hlavní výhody této metody jsou:

- **Efektivní průzkum prostoru stavů**: Průzkum je integrován přímo do modelu a je řízen učením.
- **Odstranění potřeby ladění** ϵ: Není třeba ručně nastavovat hodnoty ϵ, ϵ_{decay} a ϵ_{min}.
- **Lepší adaptivita**: Parametry šumu jsou trénovány společně s ostatními váhami sítě, což umožňuje adaptaci strategie explorace během trénování.

Při každém kroku trénování dochází k `reset_noise()`, čímž se vygenerují nové náhodné složky pro dané vrstvy. Při evaluaci (např. při testování agenta) se šum vypne, a síť se chová deterministicky.

Tato metoda je zvláště vhodná pro prostředí, kde má být průzkum adaptivní a méně závislý na ručním ladění hyperparametrů.

Níže je uvedena ukázka implementace vrstvy `NoisyLinear` v jazyce Python pomocí knihovny PyTorch:

```
class NoisyLinear(nn.Module):
    def __init__(self, in_features, out_features, std_init=0.4):
        super(NoisyLinear, self).__init__()

        self.in_features = in_features
        self.out_features = out_features
        self.std_init = std_init

       self.weight_mu = nn.Parameter(torch.FloatTensor(out_features,
        ↪  in_features))
        self.weight_sigma =
        ↪  nn.Parameter(torch.FloatTensor(out_features, in_features))
        self.register_buffer('weight_epsilon',
        ↪  torch.FloatTensor(out_features, in_features))

        self.bias_mu = nn.Parameter(torch.FloatTensor(out_features))
        self.bias_sigma =
        ↪  nn.Parameter(torch.FloatTensor(out_features))
        self.register_buffer('bias_epsilon',
        ↪  torch.FloatTensor(out_features))

        self.reset_parameters()
        self.reset_noise()
```

Výpis kódu 3.5 Implementace vrstvy `NoisyLinear` pro parametrický šum

Kapitola 4
Experimenty

Po implementaci a úspěšném natrénování jednotlivých metod popsaných v sekcích 3.2, 3.3 a 3.4 byla provedena série experimentů, jejichž cílem bylo srovnat výkonnost těchto přístupů ve hře blackjack. V této kapitole popisujeme jednotlivé metody, specifikujeme nastavení experimentálního prostředí a prezentujeme výsledky měření.

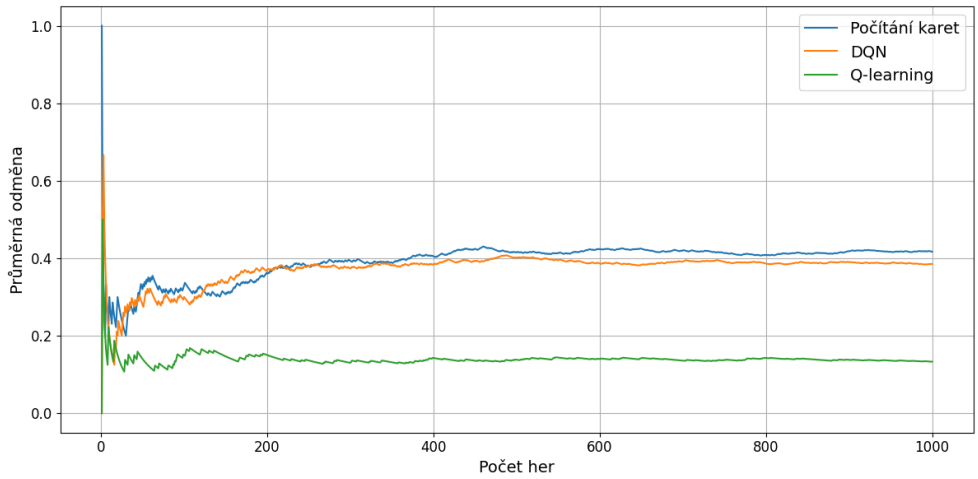
Cílem experimentální části je komplexně porovnat jednotlivé metody nejen z hlediska jejich úspěšnosti ve hře, ale také s ohledem na rychlost hraní, konzistenci výsledků a schopnost adaptace na různé herní situace.

4.1 Srovnání metod na základě výkonnosti

Pro porovnání tří zvolených metod jsme si vytvořili jednoduchý systém odměn, který slouží k objektivnímu vyhodnocení úspěšnosti agenta v jednotlivých hrách. Po každé odehrané hře agent obdrží odměnu ve formátu binární hodnoty – buď 1, nebo 0. Hodnota odměny je určena výsledkem hry. Pokud agent během jedné hry dosáhne alespoň 2000 žetonů (tj. dvojnásobku výchozího rozpočtu), je mu přiřazena odměna 1. Naopak v případě, že agent ve hře zbankrotuje, tedy přijde o všechny své žetony, obdrží odměnu 0. Tento systém umožňuje snadné srovnání výkonnosti jednotlivých metod na základě podílu úspěšných her.

Při prvním testu srovnání jsme provedli celkem 1000 testovacích běhů pro každého z agentů. Na základě těchto běhů jsme vytvořili graf průměrné úspěšnosti jednotlivých metod v závislosti na počtu odehraných her. Tento graf slouží jako hlavní nástroj pro vyhodnocení efektivity daných modelů, přičemž úspěšnost je měřena pomocí průměrné odměny popsané v předchozí části.

Dalším hlavním testovacím parametrem je průměrný počet kol v jedné hře, tedy kolik kol agentovi průměrně trvá, než buď dosáhne dvojnásobku počátečního vkladu, nebo zbankrotuje. Stejně jako u předchozího testu jsme každého agenta otestovali na 1000 nezávislých hrách.



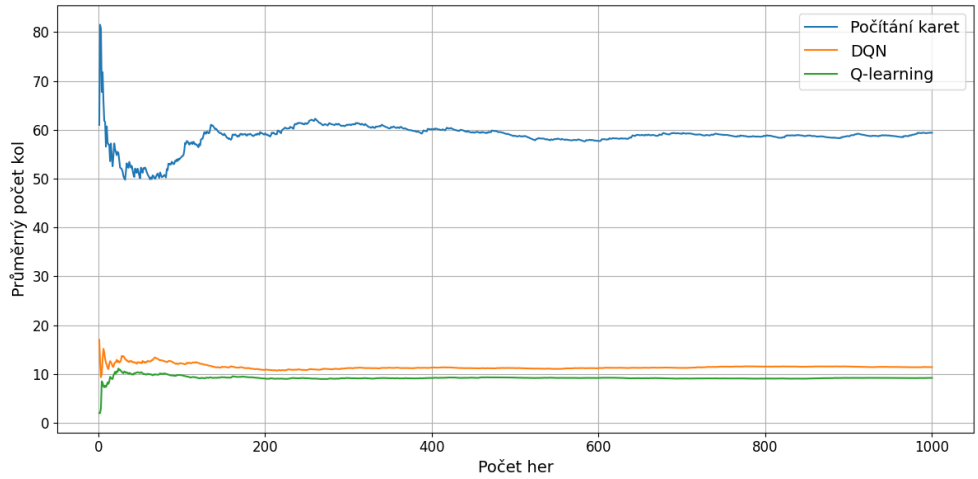
Obrázek 4.1 Srovnání průměrné odměny v závislosti na počtu her

Z grafu lze pozorovat srovnání úspěšnosti tří testovaných metod: počítání karet (modrá křivka), Deep Q-learningu (oranžová křivka) a klasického Q-learningu (zelená křivka). Metoda počítání karet vykazuje postupný nárůst výkonnosti, který se stabilizuje přibližně po 400 odehraných hrách. Její průměrná úspěšnost roste až k hranici 40 %, přičemž maximálně dosahuje přibližně 43 %.

DQN vykazuje rychlejší počáteční nárůst úspěšnosti a svého maxima dosahuje okolo 40 %. Přestože je tato metoda po většinu doby mírně méně úspěšná než počítání karet, v úseku mezi zhruba 130. a 200. hrou jej mírně překonává.

Klasický Q-learning vykazuje ze všech tří metod nejnižší výkonnost. Již od přibližně 100. hry se jeho výsledky stabilizují a zůstávají téměř konstantní, přičemž průměrná úspěšnost se pohybuje okolo 16 %.

Z těchto výsledků vyplývá, že nejlépe si vede metoda počítání karet, velmi těsně za ní následuje DQN, zatímco klasický Q-learning dosahuje výrazně horších výsledků.

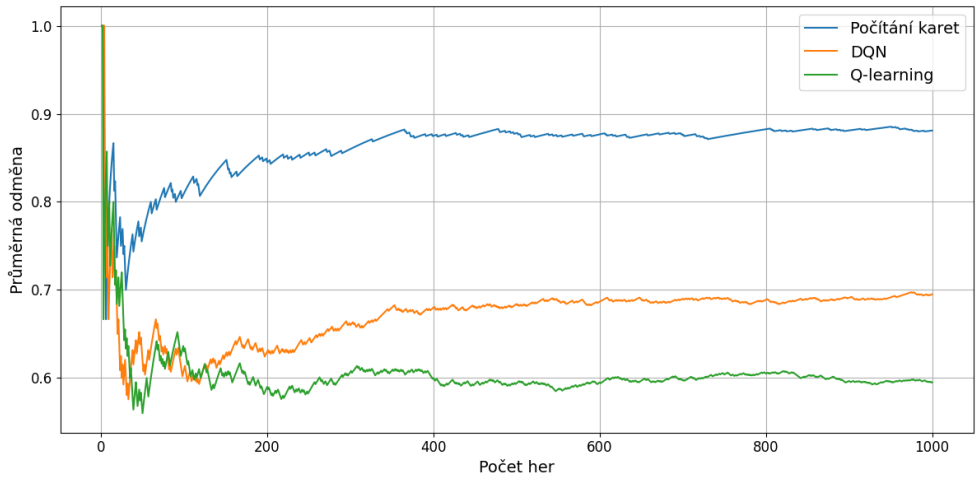


Obrázek 4.2 Srovnání průměrného počtu kol v závislosti na počtu her

Při analýze vývoje průměrného počtu kol na jednu hru je patrné, že všechny tři metody vykazují poměrně rychlou stabilizaci. Nejvýraznější počáteční výkyvy jsou vidět u metody počítání karet, která začíná s průměrným počtem 80 kol na hru, postupně klesá na přibližně 50 kol a po několika desítkách her se ustaluje kolem hodnoty 60 kol. Oproti tomu metody DQN a klasický Q-learning dosahují výrazně nižších hodnot. DQN se stabilizuje přibližně na 13 kolech na hru, zatímco Q-learning dosahuje průměrně okolo 10 kol. Tento výsledek naznačuje, že z hlediska délky jednotlivých her je nejrychlejší klasický Q-learning, těsně za ním následuje DQN a s výrazně delším průběhem her zaostává metoda počítání karet.

4.2 Porovnání metod po změně systému odměn

Po úvodním testování metod s původním systémem odměn jsme se rozhodli upravit pravidla pro přiřazování odměny tak, aby lépe odrážela úspěšnost agenta na počátku hry. V novém systému odměn agent obdrží hodnotu 1 v případě, že jeho počet žetonů během hry přesáhne 1000, čímž dosáhne pozitivního zůstatku. Tento přístup umožňuje, aby agent dostal odměnu 1 již v případě, že se mu podaří zlepšit svůj stav, i když nedosáhne dvojnásobku výchozího rozpočtu. Stále však platí, že pokud agent zbankrotuje, tedy přijde o všechny své žetony, obdrží odměnu 0. Tento upravený systém odměn poskytuje citlivější vyhodnocení výkonu metod, protože odměnu za úspěch získává agent již při dosažení pozitivního zůstatku, což umožňuje rychlejší identifikaci úspěšných her.

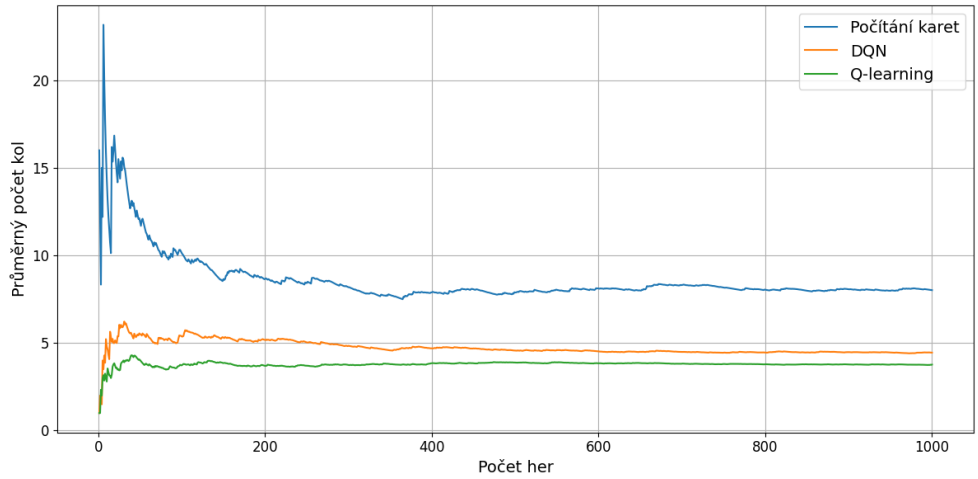


Obrázek 4.3 Srovnání průměrné odměny v závislosti na počtu her při změněném systému odměn

V grafu znázorňujícím průměrnou odměnu dosahuje metoda počítání karet nejvyšší výkonnosti, kdy její průměrná úspěšnost činí téměř 90 %. Druhou nejúspěšnější metodou je DQN, která dosahuje přibližně 70 % průměrné odměny. Nejhůře si opět vede metoda Q-learning, jejíž hodnota se pohybuje okolo 60 %. Ve srovnání s předchozími výsledky lze v grafu úspěšnosti pozorovat výraznější rozdíl mezi DQN a metodou počítání karet, což naznačuje, že právě počítání karet vykazuje v tomto experimentálním nastavení výrazně lepší výkon.

Z předchozího experimentu rovněž vyplývá, že metoda počítání karet sice často končí v pozitivním zisku, avšak jen zřídka dosahuje dvojnásobku původního vkladu. Oproti tomu metoda DQN má při dosažení kladného výsledku vyšší pravděpodobnost, že se jí podaří dosáhnout i této hranice.

Podobně jako v předchozí sekci se i v této části zaměříme na průměrný počet kol, která agent odehraje během jedné hry. Oproti předchozímu nastavení zde očekáváme výrazně kratší dobu trvání jednotlivých her, což by se mělo projevit nižším počtem odehraných kol na hru.



Obrázek 4.4 Srovnání průměrného počtu kol v závislosti na počtu her při změněném systému odměn

Jak bylo očekáváno, došlo k poklesu průměrného počtu kol. Přesto však metoda počítání karet zůstává méně efektivní než zbývající dvě metody.

Konkrétně lze u metody počítání karet pozorovat, že na začátku tréninku se průměrný počet kol pohybuje okolo 20, avšak velmi rychle dochází k jeho snížení na přibližně 8 kol, kde se tato hodnota stabilizuje po zbytek tréninku. Naproti tomu metoda DQN vykazuje konzistentně nižší počet kol již od počátku, přičemž se ustaluje v rozmezí 4 až 5 kol. Ještě efektivněji si v tomto ohledu vede klasický Q-learning, jehož průměrný počet kol se pohybuje v rozmezí 3 až 4, a jeho vývojová křivka má obdobný tvar jako u DQN, jen s nižšími hodnotami.

Tato data naznačují, že agenti řízení metodami založenými na Q-learningu jsou schopni dosahovat cílového stavu výrazně rychleji než agent využívající počítání karet, a to již od rané fáze tréninku.

4.3 Analýza strategií sázení a akcí

V této sekci jsme se zaměřili na podrobnou analýzu herního chování a sázejících strategií jednotlivých agentů prostřednictvím série experimentů. Cílem bylo komplexně zhodnotit, jak různé přístupy k rozhodování ovlivňují efektivitu hry a preference při volbě sázek a herních akcí. V úvodu jsme se soustředili na srovnání základních metrik, které poskytují kvantitativní vyjádření výkonnosti každé ze zkoumaných strategií. Tyto metriky tvoří základní rámec pro objektivní posouzení rozdílů mezi metodami a slouží jako výchozí bod pro následnou interpretaci výsledků.

Následně jsme se podrobněji věnovali chování agentů v průběhu jednotlivých herních epizod, s cílem identifikovat rozdíly ve výběru akcí a strategickém

přístupu v různých fázích hry. Tato analýza umožňuje lépe porozumět tomu, jak zvolená strategie ovlivňuje herní rozhodnutí a jakým způsobem se projevuje v celkovém výkonu agenta.

Pro ilustraci rozdílů mezi strategiemi bylo provedeno kvantitativní srovnání klíčových ukazatelů vztahujících se k sázení a volbě akcí. V následující tabulce jsou prezentovány výsledky pro tři vybrané přístupy: metodu počítání karet, DQN a klasický Q-learning. Hodnocené metriky zahrnují průměrnou, nejčastější, minimální a maximální výši sázky, stejně jako nejčastěji zvolenou akci během hry.

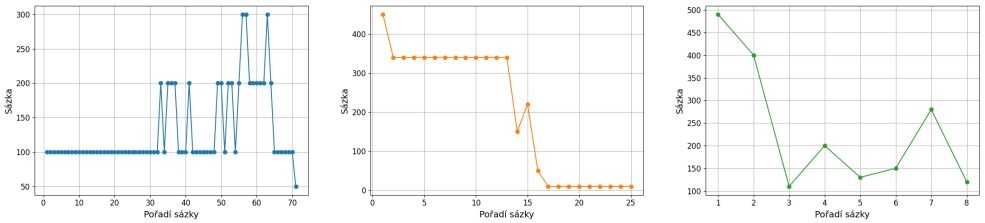
Tabulka 4.1 Tabulka srovnání strategií sázení a akcí

Metrika	Počítání karet	Deep Q-learning	Q-learning
Průměrná sázka	110.97	251.44	265.76
Nejčastější sázka	100.00	330.00	220.00
Minimální sázka	20.00	10.00	10.00
Maximální sázka	500.00	500.00	500.00
Nejčastější akce	Hit	Stand	Stand

Z výsledků vyplývá, že agent využívající počítání karet má nejnižší průměrnou sázku (110.97) a nejčastěji sází částku 100.00. Naopak agenti využívající Q-learning a DQN mají vyšší průměrné i nejčastější sázky, přičemž u DQN se průměrná sázka pohybuje kolem 251.44 a nejčastější sázka je 330.00. Minimální a maximální sázky jsou pro všechny strategie v podobném rozsahu (od 10 do 500).

Pokud jde o herní akce, agent využívající počítání karet preferuje akci *Hit*, zatímco agenti založení na Q-learningu a Deep Q-learningu častěji volí akci *Stand*. To naznačuje odlišné přístupy k řízení rizika a rozhodování o dalším postupu v závislosti na metodě, kterou agent využívá.

Dále jsme se zaměřili na podrobnou analýzu sázení jednotlivých agentů v průběhu jedné konkrétní hry, abychom odhalili případné rozdíly v jejich strategii.



(a) Počítání karet **(b)** Deep Q-learning **(c)** Q-learning

Obrázek 4.5 Srovnání strategie sázení

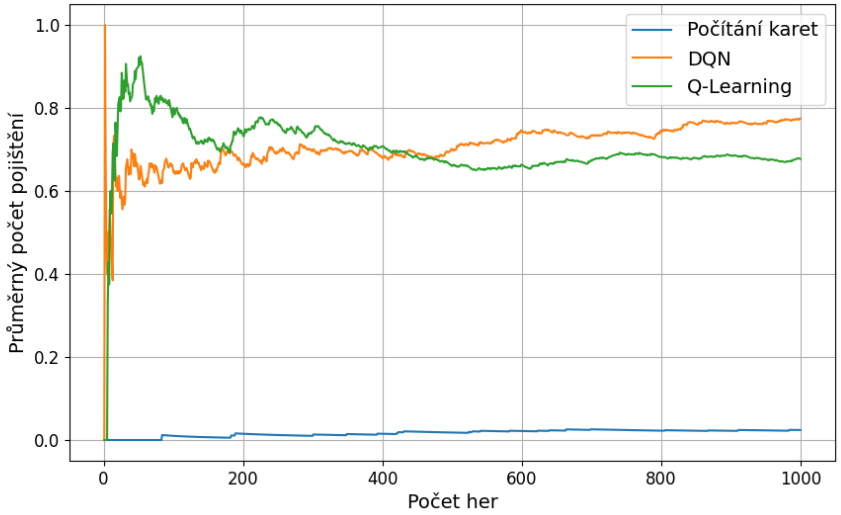
Z grafu je kromě rozdílů v délce jednotlivých kol patrná také odlišná dyna-

mika sázek mezi jednotlivými strategiemi. Agent využívající strategii počítání karet ve většině případů sází konstantní částku ve výši 100, tedy jednu základní sázkovou jednotku. Vyšší sázky, jako například 200 nebo 300, se vyskytují jen zřídka a spíše výjimečně.

Strategie založená na DQN se vyznačuje výrazně odlišným průběhem. Na začátku epizody agent sází částky kolem 450, následně po několik kol udržuje sázky na úrovni 350 a poté postupně snižuje jejich výši až na minimální možnou hodnotu 10.

Podobný vývoj lze pozorovat i u agenta využívajícího klasický Q-learning. Počáteční sázky jsou relativně vysoké, následně dochází k jejich poklesu na hodnotu přibližně 100. V závěrečné fázi epizody se pak sázky stabilizují a pohybují se převážně v rozmezí mezi 100 a 200.

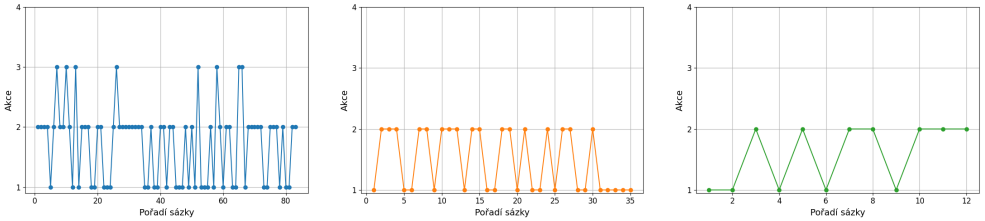
V následujícím testu se zaměříme na porovnání průměrného počtu zahrání pojištění jednotlivými agenty v průběhu 1000 her.



Obrázek 4.6 Srovnání průměrného počtu pojištění v závislosti na počtu her

Z grafu vyplývá, že žádný z agentů nezahrává pojištění příliš často během jedné hry. Nejčastější výskyt této akce lze pozorovat u agentů využívajících Q-learning a DQN, kteří ji zahrávají v průměru přibližně 0,7krát za hru. Naopak agent využívající počítání karet pojištění téměř vůbec nezahrává, přičemž jeho hodnota zůstává blízko nule.

V posledním experimentu se budeme věnovat porovnání způsobu zahrávání jednotlivých akcí agenty v průběhu jedné hry.



(a) Počítání karet **(b)** Deep Q-learning **(c)** Q-learning

Obrázek 4.7 Srovnání strategie akcí

Z výsledných grafů lze pozorovat, že agent využívající počítání karet na rozdíl od zbývajících dvou agentů aktivně využívá akci *double down*. Zároveň u něj dochází k přibližně vyrovnanému poměru mezi akcemi *hit* a *stand*. Naproti tomu agenti využívající metody DQN a klasický Q-learning používají téměř výhradně pouze akce *hit* a *stand*, přičemž s mírnou převahou častěji volí *stand*.

Kapitola 5
Závěr

V této práci jsme se zaměřili na porovnání různých metod hraní blackjacku z hlediska jejich efektivity a schopnosti dosáhnout dlouhodobé výhody nad kasinem. Pomocí simulací jsme objektivně vyhodnotili výkonnost jednotlivých metod.

Z experimentů vyplynulo, že tradiční metoda počítání karet vykazuje nejvyšší čistou úspěšnost. Na druhou stranu, model založený na DQN, i přes mírně nižší úspěšnost, vyniká výrazně vyšší rychlostí zpracování jednotlivých herních kol, což v praktickém nasazení představuje významnou výhodu. Z hlediska efektivity, tedy poměru mezi časem a dosaženým ziskem, se tak DQN jeví jako velmi konkurenceschopný přístup.

Metody založené na strojovém učení navíc vykazují slibný potenciál z hlediska schopnosti přizpůsobovat se prostředí, ve kterém operují. Tato adaptabilita je důsledkem samotné povahy strojového učení, jež umožňuje modelům zlepšovat své rozhodování na základě interakce s herním prostředím a průběžného učení ze specifických situací.

Ačkoli agenti neprokázali schopnost trvale překonávat hranici 50 % úspěšnosti v dosažení dvojnásobku původního vkladu, při zaměření pouze na dosažení pozitivního zisku tuto hranici výrazně překonali. To znamená, že vyvinuté metody jsou v principu funkční a při vhodném nastavení a optimalizaci mohou vést k výhodě nad kasinem.

Tyto výsledky ukazují potenciál dalšího výzkumu v této oblasti a možnosti zlepšení metod například pomocí kombinace více přístupů nebo jemnějšího ladění parametrů u modelů s hlubokým učením.

Bibliografie

1. BALDWIN, Roger R.; CANTEY, Wilbert E.; MAISEL, Herbert; MCDERMOTT, James P. The Optimum Strategy in Blackjack. 1956, roč. 51, č. 275, s. 429–439. Dostupné také z: https://doi.org/10.1080/01621459.1956.10501334. [online]. [visited on 2025-04-14].
2. MILLMAN, Martin H. A Statistical Analysis of Casino Blackjack. *The American Mathematical Monthly*. 1983, roč. 90, č. 9, s. 644–648. Dostupné z DOI: 10.1080/00029890.1983.11971251. [online]. [visited on 2025-04-20].
3. GOTTLIEB, Gary. An Analytic Derivation of Blackjack Win Rates. *Operations Research*. 1985, roč. 33, č. 5, s. 971–974. Dostupné z DOI: 10.1287/opre.33.5.971. [online]. [visited on 2025-04-20].
4. WERTHAMER, N. Richard. *Risk and Reward: The Science of Casino Blackjack*. Springer, 2009. Dostupné z DOI: 10.1007/978-1-4419-0253-5.
5. THORP, Edward O. *Beat the Dealer: A Winning Strategy for the Game of Twenty-One*. New York: Random House, 1962. ISBN 0-394-70310-3.
6. THORP, Edward O. *Beat the Dealer: A Winning Strategy for the Game of Twenty-One*. New York: Random House, 1962. ISBN 0-394-70310-3.
7. THORP, Edward O. *Beat the Dealer: A Winning Strategy for the Game of Twenty-One*. New York: Random House, 1962. ISBN 0-394-70310-3.
8. ZUTIS, K.; HOEY, J. Who's counting? Real-time blackjack monitoring for card counting detection. In: *Proceedings of the International Conference on Computer Vision Systems*. Springer, 2009. Dostupné z DOI: 10.1007/978-3-642-04667-4_36. [online]. [visited on 2025-04-27].
9. CHEN, Billy. *Visualizing the Effectiveness of the Hi-Low Card Counting System* [https://graphics.stanford.edu/~billyc/class/vis_win0304/as2/]. 2025. [online]. [visited on 2025-04-25].

10. THORP, Edward O. *Beat the Dealer: A Winning Strategy for the Game of Twenty-One*. New York: Random House, 1962. ISBN 0-394-70310-3.
11. VIDÁMI, M.; SZILÁGYI, L.; ICLANZAN, D. Real Valued Card Counting Strategies for the Game of Blackjack. *CORE Research Archive*. 2020, s. 23–27. Dostupné také z: `https://core.ac.uk/download/pdf/334425979.pdf`. [online]. [visited on 2025-04-27].
12. THORP, Edward O. *Beat the Dealer: A Winning Strategy for the Game of Twenty-One*. New York: Random House, 1962. ISBN 0-394-70310-3.
13. VANCURA, Olaf; FUCHS, Ken. *Knock-out Blackjack: The Easiest Card-counting System Ever Devised*. 3687 South Procyon Avenue, Las Vegas, Nevada 89103, USA: Huntington Press, 1998. ISBN 0-929712-31-5.
14. BURAMDOYAL, A.; GEBBIE, T. Variations on the Reinforcement Learning performance of Blackjack. *arXiv preprint arXiv:2308.07329*. 2023. Dostupné také z: `https://arxiv.org/abs/2308.07329`. [online]. [visited on 2025-04-27].
15. KAELBLING, Leslie P.; LITTMAN, Michael L.; MOORE, Andrew W. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*. 1996, roč. 4, s. 237–285. Dostupné také z: `https://www.jair.org/index.php/jair/article/view/10166/24110`. [online]. [visited on 2025-04-27].
16. SUTTON, Richard S.; BARTO, Andrew G. *Reinforcement Learning: An Introduction*. 2. vyd. MIT press, 2018. Dostupné také z: `http://incompleteideas.net/book/the-book-2nd.html`. [online]. [visited on 2025-04-27].
17. WATKINS, Christopher John Cornish Hellaby. *Learning from delayed rewards*. 1989. Dostupné také z: `https://www.academia.edu/3294050/Learning_from_delayed_rewards`. Dis. pr. [online]. [visited on 2025-04-28].
18. WATKINS, Christopher JCH; DAYAN, Peter. Q-learning. *Machine Learning*. 1992, roč. 8, s. 279–292. Dostupné také z: `https://link.springer.com/content/pdf/10.1007/BF00992698.pdf`. [online]. [visited on 2025-04-28].
19. MNIH, Volodymyr; KAVUKCUOGLU, Koray; SILVER, David; RUSU, Andrei A; VENESS, Joel; BELLEMARE, Marc G; GRAVES, Alex; RIEDMILLER, Martin; FIDJELAND, Andreas K; OSTROVSKI, Georg et al. Human-level control through deep reinforcement learning. *Nature*. 2015. Dostupné z DOI: `10.1038/nature14236`. [online]. [visited on 2025-04-28].

20. HESTER, Todd; VECERIK, Matej; PIETQUIN, Olivier; LANCTOT, Marc; SCHAUL, Tom; PIOT, Bilal; HESSEL, Matteo; ASLANIDES, John; SILVER, David. Deep Q-learning from Demonstrations. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. AAAI, 2018. Dostupné také z: `https://ojs.aaai.org/index.php/AAAI/article/view/11757`. [online]. [visited on 2025-04-30].

21. FEDUS, William; RAMACHANDRAN, Prajit; AGARWAL, Rishabh; BENGIO, Yoshua; LAROCHELLE, Hugo; ROWLAND, Mark; DABNEY, Will. Revisiting Fundamentals of Experience Replay. *arXiv preprint arXiv:2007.06700*. 2020. Dostupné také z: `https://proceedings.mlr.press/v119/fedus20a/fedus20a.pdf`. [online]. [visited on 2025-04-30].

22. MNIH, Volodymyr; KAVUKCUOGLU, Koray; SILVER, David; GRAVES, Alex; ANTONOGLOU, Ioannis; WIERSTRA, Daan; RIEDMILLER, Martin. Playing Atari with Deep Reinforcement Learning. 2013. Dostupné také z: `https://arxiv.org/abs/1312.5602`. [online]. [visited on 2025-04-28].

23. SCHAUL, Tom; QUAN, John; ANTONOGLOU, Ioannis; SILVER, David. Prioritized Experience Replay. *arXiv preprint arXiv:1511.05952*. 2015. Dostupné také z: `https://arxiv.org/abs/1511.05952`. [online]. [visited on 2025-05-01].

24. GOLDBERGER, Jacob; BEN-REUVEN, Ehud. Training deep neural-networks using a noise adaptation layer. In: 2017. Dostupné také z: `https://openreview.net/forum?id=H12GRgcxg`. [online]. [visited on 2025-05-01].

25. GU, Shixiang; LILLICRAP, Timothy; SUTSKEVER, Ilya; LEVINE, Sergey. Continuous deep Q-learning with model-based acceleration. In: *Proceedings of the 33rd International Conference on Machine Learning (ICML)*. New York, NY, USA: JMLR: W&CP, PMLR, 2016, sv. 48. Dostupné také z: `https://proceedings.mlr.press/v48/gu16.pdf`. [online]. [visited on 2025-04-28].