FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Quantum Reinforcement Learning applied to Games

Miguel Alexandre Brandão Teixeira

U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Ana Paula Rocha, PhD

Co-Supervisor: António Castro, PhD

July 19, 2021

Quantum Reinforcement Learning applied to Games

Miguel Alexandre Brandão Teixeira

Mestrado Integrado em Engenharia Informática e Computação

July 19, 2021

Abstract

Reinforcement learning (RL) is a machine learning (ML) paradigm where an agent learns how to optimize its behavior solely through its interaction with the environment, without any previous knowledge. It has been extensively studied and successfully applied to complex problems of many different domains in the past decades, i.e., robotics, games, scheduling. However, the performance of these algorithms becomes limited as the complexity and dimension of the state-action space increases. Different solutions have been devised to tackle this limitation, such as combining RL with deep learning techniques. Still, these solutions have their own drawbacks, especially the learning efficiency and the data required to train the agent.

Recent advances in quantum computing and quantum information have sparked interest in possible applications to machine learning. By taking advantage of quantum mechanics, it is possible to efficiently process immense quantities of information and improve computational speed. In RL, several studies have been made to assess how quantum computing can improve the performance of classical approaches. The results demonstrate that the quantum algorithms achieved a better balance between exploration and exploitation and can explore the state space more efficiently, improving its learning speed.

In this dissertation, we combined quantum computing with reinforcement learning and studied its application to a board game to assess the benefits that it can introduce, namely its impact on the learning efficiency of an agent. The domain of board games provided a deterministic environment with perfect information, where we could focus on the learning process. After implementing the quantum algorithm, we performed an analysis of its performance and compared its results against a classical approach. We concluded that the proposed quantum exploration policy improved the convergence rate of the agent and promoted a more efficient exploration of the state space.

Keywords: Reinforcement Learning, Quantum Computing, Board Games

Resumo

Aprendizagem por reforço é um paradigma de aprendizagem máquina onde um agente aprende a optimizar o seu comportamento através da interação com o ambiente envolvente, sem qualquer conhecimento prévio. Nas últimas décadas, este paradigma tem sido estudado extensivamente e aplicado com sucesso a problemas de diferentes domínios, tal como o controlo de robôs, jogos e problemas de escalonamento. No entanto, o desempenho destes algoritmos torna-se limitado à medida que a complexidade do problema e a dimensão do espaço de estados aumentam. Diferentes soluções foram desenvolvidas para lidar com estas limitações, tal como combinar aprendizagem por reforço com técnicas de *deep learning*. Contudo, estas soluções tem os seus próprios obstáculos, nomeadamente a eficiência da aprendizagem e a quantidade de dados necessária para treinar o agente.

Os avanços recentes em computação quântica e informação quântica despertaram o interesse em possíveis aplicações na aprendizagem máquina. Ao tirar partido das propriedades quânticas, é possível processar uma grande quantidade de informação de forma eficiente. Na área de aprendizagem por reforço, foram feitos vários estudos para avaliar como a computação quântica pode melhorar o desempenho dos algoritmos clássicos. Os resultados obtidos demonstram que os algoritmos quânticos alcançaram um melhor equilíbrio durante a fase de exploração e conseguem explorar o espaço de estados de forma mais eficiente, melhorando a velocidade de aprendizagem.

Nesta dissertação combinámos computação quântica com aprendizagem por reforço e estudamos a sua aplicação a um jogo de tabuleiro de forma a avaliar os potenciais benefícios que esta pode trazer, nomeadamente o seu impacto na eficiência da aprendizagem de um agente. O domínio dos jogos estratégicos proporcionou um ambiente determinístico com informação perfeita, onde podemos concentrar-nos no processo de aprendizagem. Após a implementação do algoritmo quântico, fizemos uma análise do desempenho e os resultados obtidos foram comparados com os de um algoritmo clássico. Concluímos que a política de exploração quântica proposta melhorou a taxa de convergência do agente e promoveu uma exploração mais eficiente do espaço de estados.

Keywords: Aprendizagem por reforço, Computação Quântica, Jogos de Tabuleiro

Acknowledgements

I am deeply grateful to my supervisors Ana Paula Rocha and António Castro, for their constant help and guidance throughout the development of this dissertation. They were always ready to answer any of my questions and were eager to help me whenever I had a problem. Their advice and feedback proved to be invaluable to the success of this work.

I would also like to thank my parents for their never-ending support and encouragement, as well as my friends for the moments we shared together that made the past five years fly like a breeze.

Thank you,
Miguel Teixeira

“We must know.
We will know.”

David Hilbert

Contents

List of Figures

List of Tables

Abbreviations

DP	Dynamic Programming
DQN	Deep Q-Network
DDQN	Double Deep Q-Network
MC	Monte Carlo
MDP	Markov Decision Process
ML	Machine Learning
NISQ	Noise Intermediate-Scale Quantum
PS	Projective Simulation
QC	Quantum Computing
QDK	Quantum Development Kit
QNN	Quantum Neural Network
QPU	Quantum Processing Unit
QRL	Quantum Reinforcement Learning
QVC	Quantum Variational Circuit
RL	Reinforcement Learning
SDK	Software Development Kit
TD	Temporal Difference
VQC	Variational Quantum Circuit

Chapter 1

Introduction

This chapter starts with a brief contextualization of the problem addressed in this dissertation, followed by a description of the motivation and objectives. Afterwards, the expected main contributions are presented and the structure of the document is outlined.

1.1 Context

Reinforcement learning (RL) is a machine learning paradigm, where a learner learns how to behave through its interaction with a dynamic environment and the feedback received. Each action is associated with a reward signal, either positive or negative. Through trial and error, the learner must converge to a policy that maximizes said reward signal and achieves the desired goal. RL has been applied to many problems of different domains, such as robotics and game theory.

In the last decades, this type of learning has been extensively applied to board games. In 1959, Samuel [36] studied the application of heuristic search methods in Checkers. Although this work only achieved an above-average performance against human opponents, it was an important step for self-learning programs and a significant achievement in machine learning. In games such as Chess, Go and Shogi, it is significantly harder to compete with human players due to the complexity of the games and the strategies involved. In order to tackle that issue, reinforcement learning has been combined with deep learning methods, leading to Deep Reinforcement Learning [29, 41]. It achieved impressive results against other RL algorithms and even managed to beat the world's top players.

Quantum computing (QC) introduces a new way for computers to efficiently process large amounts of information, offering theoretical quantum improvements in Machine Learning (ML) algorithms [6]. It has motivated numerous studies in this area and led to an increase of interest in quantum-enhanced ML algorithms. In the work of Paparo & al. [33], a quantum reinforcement learning (QRL) method is proposed, combining QC with RL. This algorithm has shown promising results by taking advantage of quantum properties, excelling classical RL algorithms in action selection policy, scalability and learning speed.

1.2 Motivation and objectives

Several challenges arise in RL problems. One of these challenges is the *exploration vs exploitation* dilemma [43]. To achieve its optimization goal, a learner not only has to exploit previously successful actions but also explore new actions to fully learn the environment and improve its future decisions. The dilemma is that the learner must choose actions that yield reward but can only do so after exploring new actions in the action space. Neither exploration nor exploitation can be ignored and, therefore, it is imperative to come up with a strategy that balances both points. Another challenge to overcome is the slow learning speed, due to the *curse of dimensionality* [5, 43]. As the complexity of the problem increases, so does its state and action space. This negatively influences the rate of convergence, making it more difficult for the learner to learn an optimal strategy efficiently.

These challenges are not trivial to solve. Several approaches have emerged to tackle them, such as Dynamic Programming, Monte-Carlo methods and Temporal-Difference learning. Each approach has its strengths and weaknesses and differs in terms of efficiency and speed of convergence. Although these approaches introduce several improvements to key areas of RL, there is still room for more.

With the increasing interest in quantum computing, several studies have been made to evaluate the impact that it can have on reinforcement learning [2, 9, 15, 18]. These studies have shown promising results when applying quantum-enhanced algorithms to a RL problem, describing major improvements in the performance and learning efficiency compared with classical RL algorithms.

With this dissertation, we pretend to assess if the same performance improvements can be achieved in the domain of board games. Therefore, we developed and applied a quantum-enhanced exploration policy to an off-policy RL algorithm, taking advantage of the state superposition principle and the amplitude amplification technique to deal with the convergence and the huge dimension of the state space. The game of Checkers was chosen due to its simple rules yet average computational complexity, allowing us to focus on the learning process while presenting an interesting problem to apply RL.

In order to better guide our work, we formulated several research questions that we aim to answer after applying the quantum algorithm to the problem at hand and analyzing the results obtained.

Those questions are:

RQ 1. How can quantum computing be applied to RL in the domain of board games?

RQ 2. How does the quantum approach impacts the learning efficiency of an agent [1]?

RQ 3. Is the quantum approach scalable for more complex board games?

[1] In the context of this dissertation, the agent assumes the role of a player.

1.3 Contributions

By the end of the dissertation, we expect to attain some contributions by:

- Developing and implementing a classical and a quantum RL algorithm
- Applying both algorithms to the game of Checkers with different board sizes
- Presenting a performance analysis between both approaches

1.4 Document structure

The dissertation is divided into seven chapters. The current chapter introduces the problem that we aim to solve, providing some context to it and delineating the motivations and goals behind our work. Additionally, the chapter also contains the research questions that we aim to answer. Following, chapter 2 provides a background on important concepts, such as reinforcement learning, quantum computing and quantum platforms. Chapter 3 presents a review of existing literature, focusing on the topics of quantum reinforcement learning, quantum computing and quantum walks. Chapter 4 defines the game of Checkers as an RL problem, detailing the representation used for the state space, the reward function and the opponent. Afterward, chapter 5 details the proposed solution, including the classical and quantum algorithms to be implemented and the challenges associated with each approach. Chapter 6 delineates the experiments conducted to assess the performance of both approaches in different environments. Furthermore, the chapter provides an analysis of the results and a comparison between the classical and the quantum agents. Finally, chapter 7 presents an overview of the main contributions and future work.

Chapter 2

Background

This chapter provides a background regarding reinforcement learning and quantum computing, covering key concepts required for the discussion of this dissertation. It begins with an introduction to reinforcement learning, including a formal definition, a description of its composing elements and popular approaches. Following, it presents a brief description of quantum mechanics and their application in the development of quantum algorithms, providing a more detailed explanation of Grover's search algorithm and the amplitude amplification technique. Lastly, it provides an analysis of the existing quantum frameworks and cloud-based quantum computing platforms.

2.1 Reinforcement learning

Reinforcement learning is a machine learning paradigm, where a learning agent starts with little to no knowledge and must learn which actions to take through iterative interactions with the environment. After each action, the environment provides feedback to the agent in the form of a scalar signal - a reward. Using this signal, the learner can evaluate and improve its behavior. Its goal is to learn a policy that maximizes the total reward received. In other words, RL considers a problem as a goal-directed agent that seeks to learn the sequence of decisions that optimize long-term rewards through its interaction with the environment. This type of learning is often applied to sequential decision-making problems, e.g., board games, robot control.

2.1.1 Agent and environment

A RL problem is composed of two entities: the learning agent and the environment. Both continuously interact with each other, as the learner selects actions and waits for feedback from the environment, which includes the resulting state and the associated reward. After each interaction, the learner evaluates and improves its decision-making based on the feedback received. This interaction is depicted in figure 2.1.

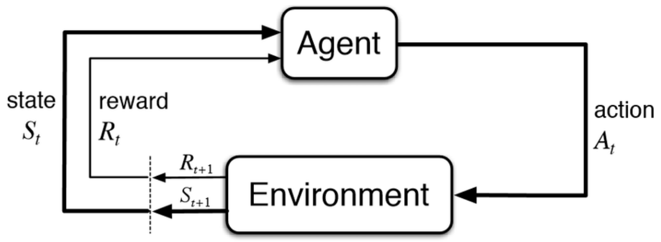


Figure 2.1: Interaction between the agent and the environment. Adapted from [43]

The interaction between the agent and the environment is a sequence of states, actions, and rewards over discrete time steps. At each time step t, the environment provides a representation of its state $S_t \in \mathcal{S}$ to the agent, which can then choose any of the available actions in that state, $A_t \in \mathcal{A}(s)$. In the next time step, the environment responds back with the new state, S_{t+1}, and the numerical reward associated with the action chosen, $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$.

2.1.2 Markov decision processes

Most RL problems can be formalized as a finite Markov decision processes (MDPs). An MDP is a mathematical framework used to formally describe the environment of a RL problem [48] and can be defined as the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where:

- $\mathcal{S}$ is the state space, a finite set containing every possible representation of the environment. Each state is unique and contains all the elements required to represent a state of the problem that is being modeled.

- $\mathcal{A}$ is the action space, a finite set containing all available actions that the decision-maker can take. The set of actions that can be taken when the system is in a state $s \in \mathcal{S}$ is denoted $\mathcal{A}(s)$, where $\mathcal{A}(s) \subseteq \mathcal{A}$.

- $\mathcal{P}(s,a,s') = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$ is the state-transition function, describing the probability of the system transitioning from state s to state s' after the agent takes the action a, where $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}(s)$.

- $\mathcal{R}(s,a) = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$ is the reward function, returning the immediate reward for taking the action a when the system is in the state s, where $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$. The function can also be defined as $\mathcal{R}(s,a,s') = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a, S_{t+1} = s']$, returning the immediate reward for taking the action a when the system is in the state s and transitions to the state s', where $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}(s)$.

- γ is the discount factor, where $\gamma \in [0,1]$. It influences and balances the weight of immediate rewards compared to future rewards and is used to ensure that the sum of rewards is finite [48].

In a MDP, every state has the Markov property. This property asserts that the transition between states is only dependent on the immediate previous state and action [43]. In other words, the result of an action is not dependent on the past interactions between the agent and the environment. Mathematically, a state $S_t \in \mathcal{S}$ has the Markov property if and only if the equality in equation 2.1 holds.

$$\mathbb{P}[S_{t+1} \mid S_t, A_t] = \mathbb{P}[S_{t+1} \mid S_0, A_0, \ldots, S_t, A_t] \tag{2.1}$$

If said property holds, then the current state captures all relevant information needed for the future. Thus, with only the latest state, the agent is capable of making the optimal decision.

Although a Markov decision process is not required to solve a RL problem, it provides a useful abstraction to model goal-oriented decision-making problems, helping an agent to learn about the dynamics of an environment and achieve its goal.

2.1.3 Goal

In reinforcement learning, the goal is to learn a policy that maximizes the total reward, taking into account both the immediate and future rewards. Each action yields a scalar reward signal, which can be either positive or negative. The environment uses this signal to provide feedback to the agent of the impact that a certain action had on the system. Using this signal, an agent can learn and adapt to optimize its actions: a positive reward encourages the agent's behavior, while a negative reward punishes it. Therefore, when defining a RL problem, it is important to properly set up the rewards depending on what one wants to achieve.

Formally, the goal of the agent is to maximize the expected return, denoted G_t, where the return is the cumulative sum of rewards [43], as described in equation 2.2:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \ldots + R_T = \sum_{k=0}^{T} R_{t+k+1} \tag{2.2}$$

where t is the current time step, such that $t \in [1, T[$ and $t, T \in \mathbb{R}$.

However, there are many problems that do not have a finite time step, i.e., $T = \infty$, which in turn causes the expected return to be infinite and prevents its optimization. Also, another aspect to consider is that it might be desirable to place a greater weight on immediate rewards than on future rewards, e.g., future rewards are uncertain or to prevent infinite returns in cyclic Markov processes. To solve both issues, a discount factor $\gamma \in [0, 1[$ is introduced to the equation 2.2. The discounted expected return is defined in equation 2.3.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.3}$$

The factor discounts future rewards and affects their significance on the expected return. If $\gamma = 0$, the agent becomes greedy and is only concerned about the immediate rewards. As γ approaches 1, the agent starts making more balanced decisions as it takes future rewards into consideration.

2.1.4 Policy

A policy, denoted π, defines the agent's behavior and determines which action is taken when in a certain state. Formally, a policy is a mapping from states to the actions [43]. A deterministic policy behaves as a lookup table, as shown in equation 2.4.

$$\pi(s) = a \tag{2.4}$$

On the other hand, a stochastic policy defines the probability of an action to be taken when the system is in a particular state, as defined in equation 2.5:

$$\pi(a \mid s) = \mathbb{P}[A_t = a \mid S_t = s] \tag{2.5}$$

where s is a state of the system and a is the action to be taken by the agent, for $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$. In a RL problem, the agent's goal is to find a policy that maximizes the expected rewards, i.e., an optimal policy.

2.1.5 Value functions and optimal policy

In most RL algorithms, the agent reaches an optimal policy by learning value functions. These functions estimate the value of being in a certain state or choosing a certain action in a given state when following a particular policy. Using these functions, the agent can assess and improve its behavior, thus converging to an optimal policy. The state-value function $V_\pi(s)$, described in equation 2.6, evaluates the expected return of starting in a state s and following the policy π.

$$V_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s] \tag{2.6}$$

Similarly, the action-value function $Q_\pi(s,a)$, described in equation 2.7, evaluates the expected return of starting in a state s, choosing an action a and following the policy π.

$$Q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a] \tag{2.7}$$

A fundamental property of value functions is that they satisfy recursive properties [43], similar to the expected return. Therefore, the state-value function can be expressed recursively as the immediate reward plus the discounted state-values of all the possible next states. The resulting expression is defined in equation 2.8 and is known as the Bellman Expectation Equation [5]:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(S_{t+1}) \mid S_t = s] \\ &= \sum_{a \in \mathcal{A}(s)} \pi(a \mid s) \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s')[\mathcal{R}(s,a,s') + \gamma V_\pi(s')] \end{aligned} \tag{2.8}$$

To solve a RL problem is the same as solving the underlying MDP. To solve an MDP, the agent must find an optimal policy, denoted π_*. A policy is defined optimal when it maximizes the state-value function, such that $V_{\pi_*}(s) \geq V_\pi(s)$ for all $s \in \mathcal{S}$ and for all policies π [48]. When following an optimal policy, it is possible to define the optimal state-value V_* and action-value Q_* functions. The optimal state-value is equal to the expected return for choosing the best action in a particular state s:

$$\begin{aligned} V_*(s) &= \max_{a \in \mathcal{A}(s)} \mathbb{E}_\pi[R_{t+1} + \gamma V_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_{a \in \mathcal{A}(s)} \sum \mathcal{P}(s,a,s')[\mathcal{R}(s,a,s') + \gamma V_*(s')] \end{aligned} \tag{2.9}$$

Similarly, the optimal action-value function Q_* is equal to the expected return for choosing an action a in a state s and then following the optimal policy:

$$\begin{aligned} Q_*(s,a) &= \mathbb{E}_\pi[R_{t+1} + \gamma \max_{a' \in \mathcal{A}(s')} Q_*(S_{t+1},a') \mid S_t = s, A_t = a] \\ &= \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s')[\mathcal{R}(s,a,s') + \gamma \max_{a' \in \mathcal{A}(s')} Q_*(s',a')] \end{aligned} \tag{2.10}$$

The expressions in equations 2.9 and 2.10 are known as the Bellman Optimality Equations [5].

2.1.6 Solving Markov decision processes

An MDP assumes that the dynamics of the environment are known, i.e., the transition function and the reward function are known. In such cases, the MDP can be solved through methods such as Dynamic Programming [5] and heuristic search. These methods are denoted *model-based*, as they require a model of the environment to compute the optimal policy. However, in most RL problems, a model may not be available to the agent. To solve such problems, *model-free* methods can be applied. Unlike the previous algorithms, *model-free* methods do not rely on the model of the MDP and instead learn by exploring the environment. These algorithms can be used to estimate the value functions of an unknown MDP (prediction problem) or optimize said value functions (control problem).

There are two main classes of *model-free* methods: Monte Carlo (MC) methods and Temporal Difference (TD) methods. Monte Carlo methods learn directly from experience. The experience comes from the simulation of complete episodes, starting at a random initial state until reaching a final state. These methods then learn and estimate the value functions based on the average sample returns [48]. Similar to MC methods, TD methods also learn from experience. However, instead of waiting until the end of an episode, these methods estimate and update the value functions after each time step. TD methods can learn from incomplete episodes by combining the sampling of MC methods with bootstrapping [1]. Unlike MC methods, these methods can also be used in non-episodic environments.

[1]Updates to the value functions rely on the observed returns along with existing estimates.

When learning an optimal policy, it is important to distinguish between *on-policy* and *off-policy* methods. An *on-policy* method, e.g., SARSA [35], evaluates and improves the policy currently being used for action selection. The agent chooses an action according to its current policy and uses the observed returns to improve it gradually. On the other hand, an *off-policy* method, e.g., Q-Learning [47], evaluates and improves a policy based on the experience obtained by following another policy. The agent uses a different policy to select an action, denoted exploration policy, and uses the observed returns to improve its target policy.

2.2 Quantum computing

Quantum computing studies the application of quantum mechanics to perform computation. By taking advantage of the superposition principle and quantum entanglement, quantum computing offers more efficient solutions and has the potential to tackle problems that are currently unfeasible in classical computers.

2.2.1 Qubit and its representation

In quantum computing, the quantum bit (qubit) is the basic unit of quantum information. A qubit is modeled as a two-level quantum system, where all its possible states can be represented using its basis states. The basis states, also called computational basis states, are two orthonormal vectors $\{|u\rangle, |u^{\perp}\rangle\}$ that define a two-dimensional complex vector space [31]. By convention, the standard basis is denoted $\{|0\rangle, |1\rangle\}$, corresponding to the classical states 0 and 1 respectively. Other basis are also commonly defined, such as $|+\rangle$ and $|-\rangle$, as shown in equation 2.11.

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad |+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \quad |-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \tag{2.11}$$

Besides its basis states, the state of a qubit can also be an arbitrary superposition of both. This quantum mechanic, denoted superposition principle, allows a qubit to be in multiple states simultaneously and is what distinguishes the qubit from the classical bit. Mathematically, a qubit $|q\rangle$ in superposition state is defined as a linear combination of its basis states [31], each associated with a complex constant. It is denoted by Dirac's notation [14] as:

$$|q\rangle = \alpha |0\rangle + \beta |1\rangle, \ \alpha, \beta \in \mathbb{C} \tag{2.12}$$

where α and β, denoted probability amplitudes, are complex numbers such that $|\alpha|^2 + |\beta|^2 = 1$. Due to this constraint, the equation 2.12 can be re-written [31] as in equation 2.13:

$$|q\rangle = cos\frac{\theta}{2} |0\rangle + e^{i\varphi} sin\frac{\theta}{2} |1\rangle), \ \theta, \varphi \in \mathbb{R} \tag{2.13}$$

where $\theta \in [0, \pi[$ and $\varphi \in [0, 2\pi[$ are coordinates in a unit three-dimensional sphere, denoted Bloch sphere. Using only the variables θ and φ, it is possible to define any qubit state.

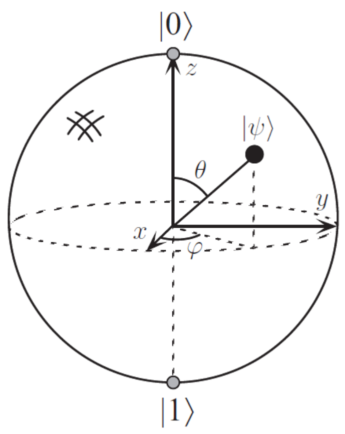


Figure 2.2: Bloch sphere. Adapted from [31]

The Bloch sphere is a geometric representation of a qubit state as a point on the sphere's surface, as depicted in figure 2.2. Each axis represents an orthonormal basis, e.g., the z-axis represents the basis $\{|0\rangle, |1\rangle\}$. The sphere helps visualize a qubit state and is useful to understand how a transformation affects it.

2.2.2 Measurement

In classical computing, it is possible to determine the state of a bit at any moment. However, the same does not apply in quantum computing. To retrieve any information from a qubit, it is necessary to measure it. When the qubit is in superposition state, it is impossible to determine its quantum state beforehand, as the result of the measurement is probabilistic.

The measurement is performed in respect to a particular orthonormal basis, $\{|u\rangle, |u^{\perp}\rangle\}$, causing the qubit to collapse to one of its states [34]. Measuring a qubit $|q\rangle$ in respect to the basis $\{|0\rangle, |1\rangle\}$ will cause it to either collapse to the state $|0\rangle$ with a probability of $|\alpha|^2$ or to the state $|1\rangle$ with a probability of $|\beta|^2$, as described in equations 2.14 and 2.15:

$$\mathbb{P}(|0\rangle) = |\langle 0|q\rangle|^2 = |\alpha\langle 0|0\rangle + \beta\langle 0|1\rangle|^2 = |\alpha \cdot 1 + \beta \cdot 0|^2 = |\alpha|^2 \tag{2.14}$$
$$\mathbb{P}(|1\rangle) = |\langle 1|q\rangle|^2 = |\alpha\langle 1|0\rangle + \beta\langle 1|1\rangle|^2 = |\alpha \cdot 0 + \beta \cdot 1|^2 = |\beta|^2 \tag{2.15}$$

where $\langle x|y\rangle$ is the inner product between the states x and y. It is important to emphasize that a superposition is basis-dependent [34], as a qubit can be in a superposition state in respect to some basis and not others, e.g., a qubit can be in superposition state in respect to the basis $\{|0\rangle, |1\rangle\}$ but not in respect to $\{|+\rangle, |-\rangle\}$. As such, measuring a qubit in respect to a certain basis can give a deterministic result, while in others it can give a probabilistic result.

The measurement process collapses the superposition and thus changes the quantum state, making it impossible to revert a qubit back to its original quantum state. Any further measurements in respect to the same basis will return the same result with a 100% probability.

2.2.3 Multiple qubit system

A n-qubit system can be defined as the tensor product of n individual quantum states [34]. As presented in equation 2.16, a quantum system in superposition state has 2^n possible states:

$$|q\rangle = |q_1\rangle \otimes |q_2\rangle \otimes ... \otimes |q_n\rangle = \sum_{i}^{2^n} a_i \; |S_i\rangle \, , \forall i \; a_i \in \mathbb{C} \tag{2.16}$$

where a_i are the probability amplitudes such that $\sum_i^{2^n} |a_i|^2 = 1$ and each $|S_i\rangle$ is a quantum state of the system.

Similarly to a single qubit system, the probability of a quantum system $|q\rangle$ collapsing to a particular basis state $|S_i\rangle$ is proportional to the square of the magnitude of its associated probability amplitude, $|a_i|^2$. It is also possible to only measure certain qubits. This process affects the rest of the system, and thus the probability amplitudes update accordingly to satisfy the normalization constraint, $\sum_i^{2^n} |a_i|^2 = 1$.

2.2.4 Entanglement

A peculiar example of two-qubit systems are the Bell states [19], depicted in equation 2.17.

$$|\phi\rangle = \frac{|00\rangle \pm |11\rangle}{\sqrt{2}} \qquad |\psi\rangle = \frac{|01\rangle \pm |10\rangle}{\sqrt{2}} \tag{2.17}$$

Although these quantum systems are in superposition state, the measurement of one of its qubits will determine the measurement result of the other. Thus, these states are said to be entangled.

A system is entangled when its quantum state cannot be decomposed as the tensor product of its individual qubit states, i.e., the quantum state of each qubit cannot be described independently from the quantum states of the other qubits of the system. In an entangled system, the result of a measurement of one of its qubits is correlated to the measurement of all the others, collapsing the superposition [31].

2.2.5 Quantum gates

In classical computing, logical operations are expressed using classical logic gates. A logic gate implements a Boolean function, performing an operation on one or more binary inputs and outputting a single binary result.

Similarly, in quantum computing the quantum logic is expressed using quantum gates. A quantum gate describes a transformation that, when applied to a quantum system, manipulates and alters its state. Geometrically, a quantum gate defines a rotation of around the Bloch sphere. A single qubit quantum gate is represented by a 2×2 unitary matrix, as shown in equation 2.18:

$$U = \begin{pmatrix} U_1 & U_2 \\ U_3 & U_4 \end{pmatrix}, \; U_1, U_2, U_3, U_4 \in \mathbb{C} \tag{2.18}$$

As presented in equation 2.19, a single qubit state can be represented as a two-dimensional complex vector, comprised of its probability amplitudes:

$$|q\rangle = \alpha|0\rangle + \beta|1\rangle \rightarrow \begin{pmatrix} \alpha \\ \beta \end{pmatrix}, \ \alpha, \beta \in \mathbb{C} \tag{2.19}$$

Mathematically, the application of a quantum gate U to a quantum state $|q\rangle$ is performed through the multiplication of the respective matrices, as defined in equation 2.20:

$$|q'\rangle = U|q\rangle = \begin{pmatrix} U_1 & U_2 \\ U_3 & U_4 \end{pmatrix} \cdot \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \alpha \times U_1 + \beta \times U_2 \\ \alpha \times U_3 + \beta \times U_4 \end{pmatrix}, \ \alpha, \beta \in \mathbb{C} \tag{2.20}$$

Unlike most classical logical gates, quantum gates are reversible [34], as any transformation applied by a quantum state has a corresponding inverse transformation that restores said quantum state. This is an important property, as it ensures that a quantum state never loses information when it goes through a transformation.

The following paragraphs provide a brief description of some quantum gates, focusing on those most relevant to the scope of this dissertation.

Pauli gates

The Pauli gates are four single qubit gates represented by the Pauli matrices. The matrices for each gate are presented in equation 2.21.

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad Y = \begin{pmatrix} 0 & i \\ -i & 0 \end{pmatrix} \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2.21}$$

The I-gate, represented by the 2x2 identity matrix, acts trivially when applied on a qubit and thus has no effect on its quantum state. The X-gate, the Y-gate, and the Z-gate, represented by their respective Pauli matrix, transforms the quantum state by performing a rotation of π radians of the Bloch sphere around their corresponding axis, e.g., the X-gate defines a rotation of π radians around the x-axis.

Hadamard gate

The Hadamard gate is a single qubit gate that is fundamental for quantum computing. When applied to a quantum state $|0\rangle$ or $|1\rangle$, the gate creates a superposition of the states $|0\rangle$ and $|1\rangle$, as described in equation 2.22.

$$H|0\rangle = |+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \quad H|1\rangle = |-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \tag{2.22}$$

The gate transforms the quantum state by performing a rotation of $\frac{\pi}{2}$ radians around the y-axis, followed by a rotation of π radians around the x-axis. Its matrix is presented in equation 2.23:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2.23}$$

Phase shift gate

The phase shift gate is a single qubit gate that, when applied to a quantum state, modifies the relative phase of the state while maintaining the magnitude of the probability amplitudes, as described in equation 2.24.

$$R_\phi \left|q\right\rangle = \alpha \left|0\right\rangle + e^{i\phi} \beta \left|1\right\rangle \tag{2.24}$$

The gate has no effect on the measurement of the quantum state. However, the change to the relative phase is important and used in some quantum algorithms, as described later. The gate applies a rotation of ϕ radians of the sphere around the z-axis. Its matrix is presented in equation 2.25:

$$R_\phi = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix} \tag{2.25}$$

where $\phi \in [0, 2\pi]$ is the phase shift.

2.2.6 Quantum algorithms

A quantum algorithm uses a sequence of quantum gates and measurements to build a transformation, which performs a certain computation when applied to a quantum system. Quantum algorithms often exploit quantum mechanics by applying said transformations to a quantum system in superposition state [34]. When a quantum transformation, U_f, is applied to n-qubit system in superposition, $\left|x\right\rangle$, it acts on each individual quantum state, as depicted in equation 2.26.

$$U_f : \left|x, 0\right\rangle \rightarrow \left|x, f(x)\right\rangle \tag{2.26}$$

This effect is denoted quantum parallelism and can be used to speed up computation, as it enables the processing of 2^n states simultaneously in a single operation. In comparison, a classical algorithm would require 2^n computations to achieve the same result. However, this property also has a drawback: only one output can be observed, as it requires a measurement to extract information from a quantum state. Nevertheless, quantum parallelism allows for more efficient solutions of classical problems and is a crucial component of most quantum algorithms, such as Shor's integer factorization [40] algorithm and Grover's search algorithm [21].

Due to the scope of this dissertation, the Grover's algorithm will be analyzed in more detail in the following paragraphs.

Grover's search algorithm

The Grover's algorithm solves a search problem by querying a quantum oracle, a function f that recognizes solutions of the problem. Given an input x, it returns $f(x) = 1$ if x is a solution and $f(x) = 0$ otherwise. The algorithm begins by transforming a n-qubit system $|s\rangle$ into an even superposition of states and then repeatedly performs Grover iterations. Each iteration updates the probability amplitudes of the quantum state, amplifying the amplitudes of the states that are a solution to the problem while shrinking the amplitudes of the others.

In a Grover iteration, denoted G, two operators are applied sequentially to the quantum state: the oracle operator and the diffusion operator. The quantum oracle can be expressed as a unitary operator, U_ω, following the formula described in equation 2.27:

$$U_\omega : |x\rangle \rightarrow (-1)^{f(x)} |x\rangle \tag{2.27}$$

where $|x\rangle$ is a quantum state, and f is the oracle function. If the input state $|x\rangle$ is a solution to the problem, the operator negates its probability amplitude by performing a phase shift of π radians.

The diffusion operator, U_s, inverts the amplitudes about their mean, amplifying the amplitudes of the quantum states that are a solution to the problem. The diffusion operator can be expressed by the formula described in equation 2.28:

$$U_s = H^{\otimes n}(2|0\rangle\langle 0| - I)H^{\otimes n} = 2|s\rangle\langle s| - I \tag{2.28}$$

where $|x\rangle\langle y|$ is the outer product between the states x and y, $|s\rangle$ is an even superposition of states and I is the $2^n \times 2^n$ identity matrix.

By continuously applying Grover iterations, the algorithm increases the probability of measuring a solution to the problem. To be optimal, it requires $O(\sqrt{N})$ iterations, where $N = 2^n$. Thus, Grover's algorithm can solve a search problem over unstructured databases using only $O(\sqrt{N})$ calls to the oracle, providing a quadratic speedup over classical algorithms. The amplitude amplification technique used in this algorithm is useful in many different problems and is applied in several quantum algorithms.

2.3 Quantum frameworks

Existing quantum hardware is expensive and requires high-maintenance, which severely limits the research in quantum computing. Furthermore, state-of-the-art implementations have limited functionality and are still too unreliable due to noise introduced in gate operations, causing errors in the computations. Until better quantum error correction techniques are available, most prototype applications will employ quantum simulators to be built and tested. However, a classical simulation of a quantum computer is still limited, as the memory requirements increase exponentially with the number of qubits. All these factors have made quantum computing inaccessible to most developers and researchers.

In recent years, several companies started investing in this area and have made quantum computing available to general use by providing access to quantum computers and simulators through their cloud-based platforms. Each platform provides its own framework to interact with the underlying hardware and ease the development of quantum applications.

The remainder of this section provides an analysis of the platforms considered for this dissertation, weighting their advantages and disadvantages. The table 2.1 presents a comparison of the features offered by each platform.

IBM Quantum Experience is IBM's offer in the quantum computing market. Their platform provides access to many different quantum systems, classified as either *open* or *premium*. The *open* quantum systems are publicly accessible and enable developers and researchers to test and run their quantum applications in real quantum systems, offering up to 15 qubits. On the other hand, the *premium* quantum systems offer more qubits, up to 65 qubits, and can be used by companies and organizations to scale their projects. These systems are backed by prototype quantum hardware, thus it is important to take into account the noise introduced by the quantum gates. The platform also offers an OpenQASM [13] quantum simulator, supporting up to 32 qubits, which can be used to prototype quantum circuits in an error-free environment. The simulator also supports noise models for simulating a quantum circuit in the presence of errors, allowing the user to test and explore the performance of a program under simulated noisy devices. To create quantum circuits and queue their execution to a backend, the platform uses Qiskit [1], an open-source framework for quantum computing. This framework simplifies and accelerates the development of quantum applications by implementing commonly used algorithms and providing tools to interact with the quantum systems. The quantum circuits can also be created through a graphical interface.

Microsoft is also investing in scalable quantum computing, with the goal of supporting future large-scale quantum applications in current quantum hardware. Their Quantum Development Kit (QDK) comes with a new programming language Q#, several domain-specific libraries for areas such as Numerics, Chemistry and Machine Learning, as well as different types of quantum simulators. The QDK provides a higher layer of abstraction that simplifies the implementation of quantum circuits, combining classical and quantum computation. Any application written using the QDK can be targeted to execute in either a local quantum simulator, supporting up to 30 qubits, or in a quantum hardware backend in Azure Quantum. The latter option is an open ecosystem that offers access to both classical and quantum hardware and aims to be an entry point to the development of quantum applications. It is still in its early stages and intends to become publicly available later in 2021.

Another option to consider is Amazon Braket. Its SDK is a hardware-agnostic framework that enables the development of quantum applications that can be executed in any compatible quantum hardware. It supports hybrid computation, allowing the use of quantum computers as co-processors to classical computations. Braket provides easy access to a variety of quantum simulators and technologies through their cloud services and third-party providers. Regarding the simulators, a state vector (SV) and a tensor network (TN) simulator are available, supporting up to 34 and 50 qubits, respectively. It also supports a range of quantum processing units (QPUs), such as gate-based quantum computers and quantum annealing systems, provided by companies such as D-Wave, IonQ and Rigetti.

Google has also taken an interest in quantum computing and the development of noise intermediate-scale quantum (NISQ) processors. Their solution is Cirq [45], a framework to develop, run and optimize quantum circuits in quantum computers and simulators. It provides abstractions that help developers deal with the noise that occurs in state-of-the-art quantum hardware. The circuits can be executed in either a local simulator or in an external high-performance simulator. Google also offers access to quantum hardware through their Quantum Computing Service. Similar to Microsoft's offer, it is still in early access and only approved projects can run quantum applications in Google's Sycamore processor, a 53 qubit quantum processor.

Of the aforementioned platforms, IBM Quantum Experience is the most compelling option. It offers a complete and feature-rich framework capable of implementing all the required quantum circuits. It also includes extensive documentation and several code examples. Furthermore, IBM provides access to high-performance simulators, with a reasonable amount of qubits, that are able to simulate and test the program's performance under different noise models.

Table 2.1: Quantum platforms

Name	**Framework**	**Quantum Hardware**	**Quantum Simulator**
IBM Quantum Experience	Qiskit (Python)	Gate-based QPUs: - Open systems, up to 15 qubits - Premium systems, up to 65 qubits	Local SV simulator Local TN simulator Local density matrix simulator OpenQASM simulator (32 qubits)
Azure Quantum	QDK (Q#)	Gate-based QPUs (third-party) Quantum annealers (third-party)	Local simulator (30 qubits) Simulated annealing
Amazon Braket	Braket SDK (Python)	Gate-based QPUs (third-party) Quantum annealers (third-party)	Local SV simulator (25 qubits) SV simulator (34 qubits) TN simulator (50 qubits)
Google Computing Service	Cirq (Python)	Sycamore QPU (54 qubits) Bristlecone QPU (72 qubits)	Local SV simulator Local TN simulator Local density matrix simulator

2.4 Summary

Reinforcement learning is a machine learning paradigm that studies how an agent learns a policy that maximizes the reward given by the environment. After each action, the agent uses the associated reward signal to assess its previous actions and converge to an optimal behavior.

Quantum computing exploits quantum mechanics such as state superposition and entanglement in order to speed up the computation of certain problems. An application of quantum computing is Grover's search algorithm, which provides a quadratic speedup in unstructured search problems through the use of a technique known as amplitude amplification.

Over the last few years, quantum computing has become more and more accessible to developers and researchers. Several companies, such as Google and IBM, have started to invest in quantum computer systems and quantum frameworks, making them available through their cloud services. The availability of quantum computers and simulators has encouraged the development of quantum solutions capable of tackling problems previously unfeasible in classical computers.

Chapter 3

Literature review

This chapter presents an analysis of related work in quantum computing and quantum reinforcement learning (QRL). It begins with a review of the existing literature of QRL methods, briefly describing the contribution of each method and the results achieved. Afterward, it describes quantum walks and their application in search problems.

3.1 Quantum reinforcement learning

This section presents an analysis of existing RL frameworks that employ quantum computing to enhance the learning process. The goal is to provide an overview of which methods have been explored in the past and possible adaptations to the domain of strategic games. For each work, a brief description is given and the main contributions and results are highlighted. A comparison between each framework is presented in the table 3.1.

Quantum superposition state learning

Dong & al. [15] describe a novel QRL framework, taking advantage of the superposition principle and quantum parallelism. In the proposed method, the state and the action set are represented as quantum systems in superposition state, where each basis state corresponds to a possible state/action of the problem. The agent's policy in a given state is dictated by the measurement result of the quantum system, causing it to collapse to one of the actions according to their associated probability amplitude.

After each action, the QRL algorithm updates and amplifies the amplitude of said action through Grover iterations, depending on the value functions and the rewards received. Thus, the algorithm exploits the amplitude amplification technique to increase the probability of selecting a good performing action in the next time step.

Similarly to the state set, the state-value function is also represented as a superposition and updated by applying a unitary transformation that implements the TD(0) updating rule. This transformation makes use of quantum parallelism and thus is able to update all states simultaneously.

An analysis of the simulated results demonstrates that the QRL algorithm achieves a good balance of exploration and exploitation through its action selection policy, scaling efficiently as the dimension of the problem increases. The authors also studied the application of this algorithm to game theory [10], which proved its usefulness in said domain.

Another similar approach was explored in the work of Li & al. [24]. In their work, the authors propose new QRL models and apply them to decision-making tasks in the fields of neuroscience and neuroeconomics. These models do not rely on learning value functions. Instead, the quantum models encode their knowledge in the amplitude associated with each action of a given state. After an action, the models map the received reward to two phase angles, which are then used in a generalized amplitude amplification algorithm [22].

Projective simulation

Briegel & al. [9, 28] propose a different approach to solve RL problems. The framework, denoted projective simulation (PS), considers a quantum agent interacting with a classical environment. Due to its classical nature, the representation of the environment as a quantum superposition is restricted, thus prohibiting achieving a speedup through quantum queries. However, the internal program of an agent can still be enhanced using quantum mechanics.

The authors propose a quantum agent whose policy is determined by a simulation-based projection. The percepts and actions (or a sequence of both) are represented within the agent as clips, forming a weighted network of clips, as depicted in figure 3.1. The clip network represents the experience of the agent, either simulated or built through its past interactions with the environment.

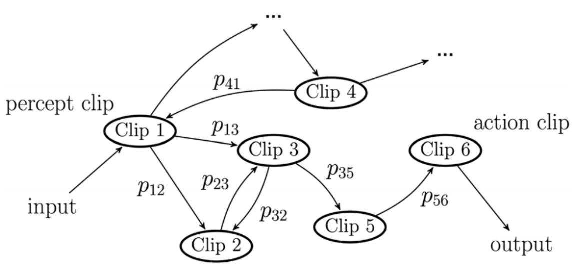


Figure 3.1: Clip network. Adapted from [28]

When the agent is given a percept, it triggers its past experience and causes a particular clip to be excited, resulting in a random walk through the network. The weights between clips determine the probability of a clip exciting an adjacent clip. Eventually, when an actuator clip is reached, the agent outputs an action. As the agent learns, the structure of the network and the weights between clips are updated according to the observed rewards, increasing the probability of performing better in the future. The framework also allows the use of tags, which are used to mark a transition between clips and allows the agent to "remember" a previously rewarded sequence of clips.

The framework was later extended in the work of Paparo & al. [33], which introduced the concept of reflecting projective simulation. Unlike a standard PS agent, a reflecting PS agent fully mixes a Markov chain and, once mixed, proceeds to sample from its stationary distribution until a tagged action clip is obtained. The authors also introduce a quantum implementation of the agent, where the action selection process is enhanced by employing a quantum search algorithm based on quantum walks, resembling a randomized Grover-like search algorithm [27]. The results demonstrate that the algorithm enables the agent to efficiently explore its internal memory, with a proven quadratic speedup over its classical counterpart.

In another work, Clausen & al. [12] took inspiration from the PS framework and enhanced it with a glow mechanism, similar to eligibility traces in RL and edge glow in PS. The agent was then applied to simple environments such as a 3x3 grid world and the invasion game, where it demonstrated a potential speedup in decision making.

Quantum-enhanced agent training

Dunjko & al. [18] extend the general agent-environment framework of RL, aiming to assist in the exploration phase. It works under the assumption that the more successful the exploration is, the better the performance in the future.

An agent is enhanced by using quantum access to the environment to find a rewarding sequence faster than a classical agent. This improvement is achieved through queries to a quantum oracle, using Grover's search algorithm. After finding a rewarding sequence, the agent interacts with the classical environment to obtain the actual rewards of the sequence, as the oracle is unable to provide them. Afterward, the sequence of actions and rewards is used to train an internal simulation of the classical agent, repeatedly simulating the interaction between the said agent and the environment until it learns to choose the same sequence of actions as in the rewarding sequence.

By combining fast searching via a quantum search algorithm and a classical learning model, the enhanced agent can achieve a quadratic improvement in learning efficiency.

Measurement-based reinforcement learning

Albarrán-Arriagada & al. [2] study the application of a reinforcement learning algorithm in a measurement-based adaptation protocol, as shown in figure 3.2. In the work presented, an agent interacts and learns the model of an unknown environment through successive measurements, using an auxiliary register. After each iteration, the agent adapts its quantum state depending on the feedback received from the measurement of the register, converging to the quantum state of the environment.

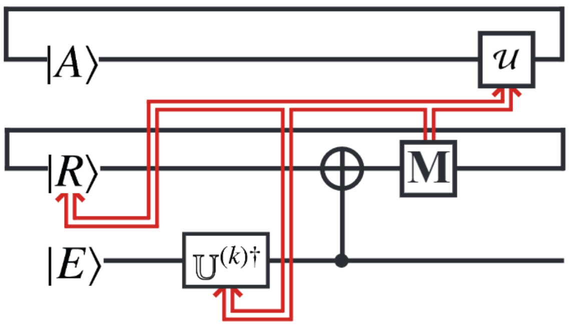


Figure 3.2: Quantum circuit for the protocol. Adapted from [2]

The algorithm was tested under different parameters and was capable of cloning an arbitrary quantum state with a high degree of fidelity (over 90%) and a small number of iterations, achieving a better performance than quantum tomography. An experimental implementation of the protocol in a Rigetti quantum computer and an IBM Q quantum computer is presented in the work of Olivares-Sánchez & al. [32] and Shenoy & al. [38], respectively. The results obtained in both experiments achieved a high degree of fidelity, confirming the expected theoretical results.

Quantum neural networks

Chen & al. [11] explore the application of variational quantum circuits (VQC) to approximate the Q value function in deep reinforcement learning. In their work, the authors follow the same principles as classical deep reinforcement learning, but replaced the network with a parameterized quantum circuit. The circuit is composed of several layers, each consisting of several single-qubit rotations that produce a highly entangled quantum state. The results demonstrate that the quantum network requires fewer parameters when compared to classical approaches.

Similarly, Lockwood & al. [26] employed variational quantum circuits to solve simple RL problems. In their work, the authors present new techniques to encode classical data into quantum data that can later be fed into quantum circuits. Furthermore, the authors expand the previous work and propose a pure and a hybrid VQC model, replacing the classical neural networks in Deep Q-Network (DQN) and Double Deep Q-Network (DDQN) algorithms. The models were tested in the CartPole and the Blackjack environments and were later compared with their classical versions. The results obtained demonstrated that both pure and hybrid models achieved a better policy and converged faster than classical models while using a smaller number of trainable parameters. In order to test how the model scales in more complex environments, the authors also applied their work to Atari games [25].

Wu & al. [49] set out to study the application of quantum algorithms to RL problems with a continuous action space. The authors proposed a quantum version of the Deep Deterministic Policy Gradient method based on quantum neural networks (QNNs), which encodes the state information into quantum state amplitude to address the dimensionality problem. The simulated results demonstrated that the method could efficiently solve quantum control tasks, such as quantum state cloning and quantum state transfer.

Table 3.1: Comparison of QRL frameworks

Reference	Environment-Agent	Method	Remarks
[15, 24]	Classical-Quantum	Amplitude Amplification	- Action selection policy is dictated by quantum measurement - Rewarding actions are amplified using Grover iterations
[9]	Classical-Quantum	Projective Simulation	- Quantum walk over a directed Markov chain (clip network) - Transition probabilities are updated according to rewards
[33]	Classical-Quantum	Reflecting Projective Simulation	- Reflecting PS agents fully mix the Markov chain and sample from its stationary distribution
[18]	Quantum-Quantum	Fast training of classical agents	- Find a winning sequence faster than a classical agent - Simulate episodes and train a classical agent
[2]	Quantum-Classical	Measurement-based Quantum Computing	- Explores the application of RL in quantum control - Used to approximate the agent's state to the target state
[11, 26]	Classical-Quantum	Quantum Deep Reinforcement Learning	- Replaces neural network with a variational quantum circuit - Applied VQCs to DQN and DDQN algorithms
[49]	Quantum-Quantum	Quantum Deep Deterministic Policy Gradient	- Deals with the discretization problem of continuous action space

3.2 Quantum walks

A quantum walk is the quantum analogue of a classical random walk and is used by several quantum algorithms to achieve a speedup in certain class of problems, such as search and element distinctness problems [3]. A quantum walk can be classified as either discrete or continuous, however this section will focus on the former.

In a classical random walk, two elements can be defined: the coin and the walker. The outcome of the coin flip determines the movement of the walker on a line, which has a 50% probability of moving either to the left or to the right, as presented in figure 3.3. A random walk can also be applied to a graph, where the movement of the walker between vertices is determined by a probabilistic event, e.g., a dice.

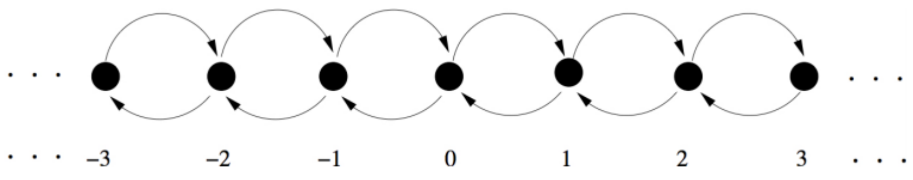


Figure 3.3: Random walk on a line. Adapted from [46]

Similar to its classical counterpart, a quantum walk is composed by two unitary transformations: the coin flip transformation and the shift transformation. The former performs a coin flip which determines the direction of the movement, while the latter moves the walker depending on the outcome. These transformations are applied sequentially to the walker for a certain number of steps and, in the end, the system is measured.

The application of quantum walks to search problems has been extensively studied in the past [4, 16, 39, 42]. In 2004, Szegedy [44] proposed a quantum walk operator, denoted W_P, and presented a quantum search algorithm capable of finding multiple marked states on any symmetric and ergodic Markov chain. The quantum algorithm demonstrated a quadratic improvement over classical algorithms, requiring only $O(\sqrt{N})$ calls to the walk operator. Later, Magniez & al. [27] expanded this work and devised a simple and more efficient implementation of the algorithm, allowing its application to a larger class of Markov chains. Their work served as a base for this dissertation and, thus, we will provide a brief overview of the algorithm in the remainder of this section.

Quantum search over a Markov chain

The quantum algorithm acts on an ergodic and reversible Markov chain, described by its transition matrix P, the set of all states X and the set of all marked states M. The algorithm acts on two separate quantum registers, representing the current and previous states of the quantum walk, respectively. Initially, the quantum state encodes the stationary distribution of P, denoted π, in a quantum superposition, as described in equation 3.1:

$$|\pi\rangle = \sum_{x\in X} \sqrt{\pi_x}\,|x\rangle\,|p_x\rangle = \sum_{y\in X} \sqrt{\pi_y}\,\left|p_y^*\right\rangle |y\rangle \tag{3.1}$$

where $|p_x\rangle = \sum_{y\in X} \sqrt{P_{xy}}\,|y\rangle$ and $\left|p_y^*\right\rangle = \sum_{x\in X} \sqrt{P_{yx}^*}\,|x\rangle$. Here, π_x represents the probability of the state x under the stationary distribution π, P_{xy} denotes the probability of transitioning from the state x to the state y and P^* denotes the time-reversed Markov chain.

The goal of the algorithm is to transform the initial state $|\pi\rangle$ into the state $|\widetilde{\pi}\rangle$, which represents the re-normalized stationary distribution π with support only over the marked states, as defined in equation 3.2:

$$|\widetilde{\pi}\rangle = \frac{1}{\sqrt{\varepsilon}} \sum_{x\in M} \sqrt{\pi_x}\,|x\rangle\,|p_x\rangle \tag{3.2}$$

where ε represents the total probability of the marked states M under the stationary distribution π. The algorithm achieves this by sequentially applying two operators to the initial quantum state, similar to an iteration of Grover's search algorithm. The first operator acts as an oracle and performs a reflection over the space span of the marked states, while the second operator performs a reflection around the initial state $|\pi\rangle$. As the second reflection is computationally expensive to perform, the authors applied a modified Kitaev's phase estimation algorithm [23] to exploit a property of Szegedy's walk operator, allowing them to efficiently construct an approximation of the diffusion operator.

After constructing both operators, the algorithm itself is quite simple. It starts by sampling from the stationary distribution a few times to accommodate the case when $\varepsilon \geq 0.25$. If no marked state is obtained, the algorithm repeatedly applies the two operators on the initial state $|\pi\rangle$ a total of T number of times, where T is chosen uniformly at random in $[0, 1/\sqrt{\varepsilon}]$ [7]. In the end, the resulting quantum state is measured. The pseudo-code for the quantum algorithm is summarized in algorithm 1.

Algorithm 1 Quantum search in a Markov chain

```
1: procedure QUANTUM_SEARCH($P$, $\varepsilon$)
2:     repeat 5 times
3:         $state \leftarrow$ sample from the stationary distribution $\pi$ of $P$
4:         if $state \in M$ then
5:             return $state$
6:     $T \leftarrow$ uniformly random in $[0, 1/\sqrt{\varepsilon}]$
7:     Prepare the initial state $|\pi\rangle$ and the required ancilla qubits
8:     repeat $T$ times
9:         Apply the oracle operator
10:        Apply the approximate diffuse operator using a fresh set of ancilla qubits
11:    $state \leftarrow$ measure the first register
12:    if $state \in M$ then
13:        return $state$
14:    else
15:        return 'no marked state exists'
```

3.3 Summary

This chapter explored different approaches that combined quantum computing with existing classical frameworks and applied them to classical and quantum tasks. These quantum approaches demonstrate an advantage over their classical counterparts, either in computational complexity or scalability. Furthermore, the chapter also provides insight into quantum walks, detailing a quantum algorithm capable of finding a marked state in a Markov chain. The algorithm performs a coherent quantum walk over the network, providing a quadratic speedup over existing classical algorithms.

Chapter 4

Checkers as an RL problem

This chapter details the modeling of the game of Checkers as an RL problem. It begins with a brief introduction to the game and its rules and presents the different board sizes used in this dissertation. Afterward, it explores possible representations of the state space and how it influences the agent's ability to generalize its knowledge. Then, the chapter discusses the design of the reward function and the application of reward shaping to enhance said function using domain knowledge. Finally, it details the implementation of the opponent and the evaluation function used by the Negamax search algorithm.

4.1 Context

Checkers, also known as English draughts, is a two-player zero-sum game with perfect information, as the players know all the moves made previously at any given moment. Although less complex than other board games such as Chess or Go, Checkers stills poses a challenge with its high decision complexity and considerate space complexity [37]. It is an easy game to learn but requires skill to play it at a master level. As such, it is an exemplar game to study, striking a balance between its simple rules and complex strategies, which allow us to focus on the learning process of the agent while still presenting an interesting problem for reinforcement learning.

In this dissertation, we considered the 8x8 variant of Checkers where each player starts with 12 pieces on a 8x8 checkered board, as shown in figure 4.1. However, during the development process, it was necessary to repeatedly test numerous changes in different components of the environment and the agent, such as the state representation and the reward function. In order to speed up the iteration process, we employed a simpler version of the game using a 6x6 board, where each player starts with 6 pieces, as shown in figure 4.2.

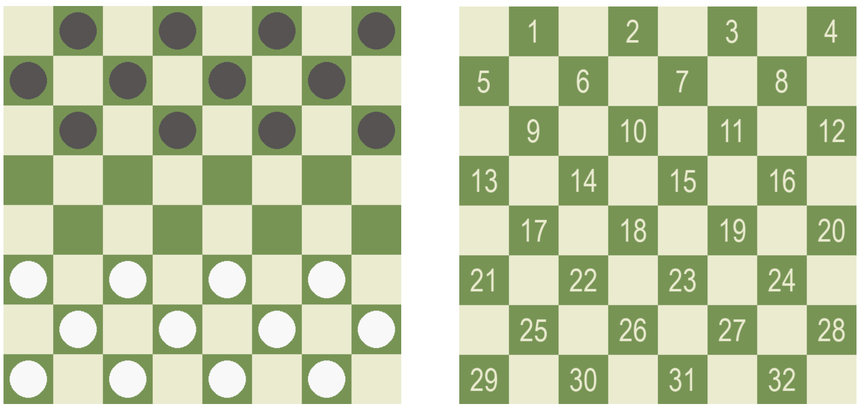


Figure 4.1: The 8x8 board and the notation used to number the squares

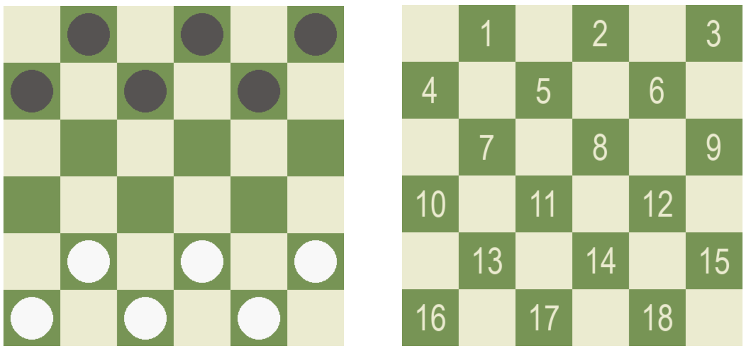


Figure 4.2: The 6x6 board and the notation used to number the squares

This scaled-down version offered us a more elementary environment with a much smaller state space complexity, all while following the same rules as the original version. Later, it also served as a benchmark to compare the agent's performance in different board sizes.

4.2 Rules

The rules of the game are quite simple. The game is played in a 8x8 board, where each player starts with 12 pieces on opposite sides of the board and take turns moving their pieces. During the entirety of the game, every piece stays in one of the dark squares of the board. There are two types of pieces: *men* (uncrowned pieces) and *kings* (crowned pieces). A *man* can move diagonally forward towards the opponent's back row. Meanwhile, a *king* (crowned piece) can move both forward and backward. A *man* is promoted to a *king* when it reaches its opponent's back row.

There are two ways of moving a piece: a simple move and a jump. A simple move consists of moving a piece diagonally into an adjacent empty square. If an adjacent square is occupied by an opponent's piece and the square immediately beyond it in the same direction is empty, the player can capture and remove the opponent's piece from the board by jumping over it. A player must perform a jump whenever possible and is allowed to jump and capture multiple pieces in its turn.

If more than one jump is available, the player is free to choose which piece(s) to capture. The chosen jump sequence does not have to be the one that maximizes the number of captured pieces.

The game ends when one of the players either has no pieces left or no legal moves available. In order to prevent extending a game indefinitely, the game is considered a draw after 20 moves without any player capturing a piece.

4.3 State representation

One of the biggest obstacles in strategic games is the complexity of the state space. In the game of Checkers, considering that the board has 32 positions and that each position can be empty or have a piece (white/black *man*, white/black *king*), a rough estimate gives us a total of 5^{32} possible states. However, most of these states are either illegal or impossible to reach, such as a state with no pieces or a state with 24 *kings*. A more realistic estimate, removing most of these impossible states, reduces the state space to around 5×10^{20} states [37]. Although the new estimate is significantly smaller than the previous, the state space is still considerably large and requires special attention to how it is represented.

A naive approach is to consider each board setting as a unique state of the problem. As the agent learns, it will eventually identify the optimal moves for the states that it has previously visited. Although this approach is simple to implement, it often leads to poor results due to the high dimensionality of the game. There are simply too many possible states to consider, which makes it intractable for the agent to visit each and every one of them due to both time and computational constraints. No matter how many states the agent explores, there are many more left to be explored. It is necessary to reduce the state space complexity using a more compact representation.

A solution is to allow the agent to generalize from its past experience. Instead of treating each state of the problem as completely unique, the agent could use what it has learned from previously visited states and apply that knowledge to new and unseen states. To achieve this, we decided to employ a different representation for the state space that aggregates similar states together. In Checkers, many different states share certain similarities, such as the number of pieces defending the back row or the number of pieces near an edge. We decided to represent a state using a vector of these features, thus reducing the number of possible states and allowing the agent to explore and learn an optimal policy from a smaller subset of the original state space.

The state representation influences how the agent perceives the environment and, as such, the choice of the features used in this representation has a significant impact on the agent's performance. The selected features must summarize the most important details of the board and have a strong correlation with how good or bad each player is performing. Another essential factor to be aware of is the number of features used to describe a state, as more features imply less generalization. Therefore, it is necessary to balance not only the quality of the features used but also their number.

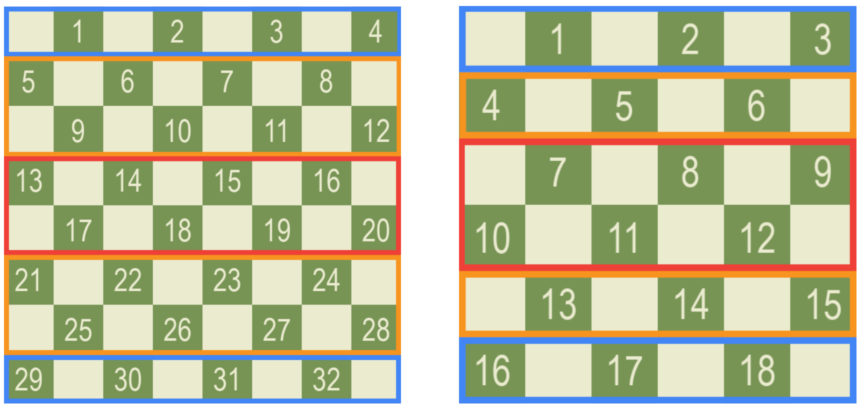
In this dissertation, we considered a combination of two features for the representation of the state space:

i) the material of each player, i.e., the pieces of each player, and

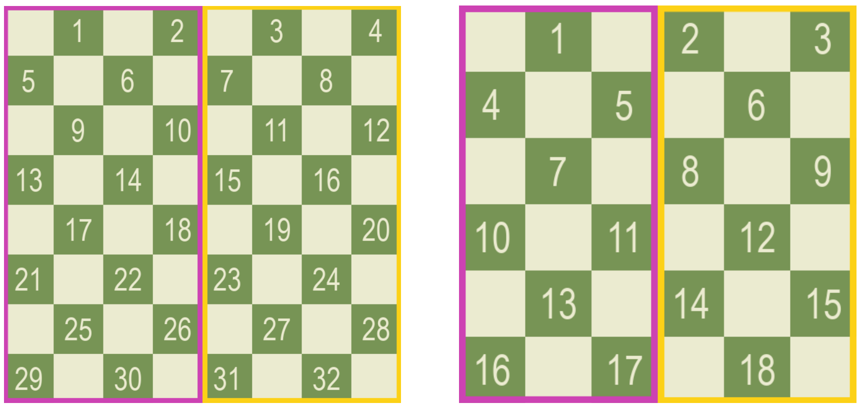
ii) how that material is distributed across the board

The first feature is an obvious choice, as it is directly related to one of the goals of the game and allows us to determine which player has an advantage at that point of the game. However, it is also necessary to take into account the position of the pieces. It is important to distinguish between a piece protecting the back row and a piece in the center, as it impacts the flow of the game. Thus, we need to consider both the material advantage and the control of different areas of the board.

In order to implement this second feature, we took inspiration from heat maps where an area can be hotter or colder depending on the value of a certain variable. As such, we divided the board into different areas, for a total of 7 areas, as shown in figure 4.3.



(a) Horizontal areas



(b) Vertical areas

Figure 4.3: The boards are divided in 7 areas. Horizontally: the blue areas represent the back row; orange areas represent the back; red area represents the center of the board. Vertically: the pink and the yellow areas represent the left and the right of the board, respectively.

The horizontal areas were chosen to help the agent recognize both offensive and defensive moves from its opponent, e.g., the opponent moving a piece from its back row or trying to make a push towards the center. The representation places greater emphasis on the center of the board and the back rows of both players, as control over those areas is the key to win the game. The representation also distinguishes between the left and right sides of the board, thereby providing more information to the agent. For each area of the board, we calculate the material advantage that a player has over the other. We explored two different approaches, one using the absolute material advantage and another using the relative material advantage, according to equations 4.1 and 4.2:

$$\text{AbsoluteMaterialAdvantage} = \text{AreaMaterialScore}_{Agent} - \text{AreaMaterialScore}_{Opp} \tag{4.1}$$

$$\text{RelativeMaterialAdvantage} = \frac{\text{AreaMaterialScore}_{Agent} - \text{AreaMaterialScore}_{Opp}}{\text{AreaMaterialScore}_{Agent} + \text{AreaMaterialScore}_{Opp}} \tag{4.2}$$

where $\text{AreaMaterialScore}_{Agent}$ and $\text{AreaMaterialScore}_{Opp}$ represents the material score of the agent and the opponent in a certain area, respectively. The absolute material advantage is calculated from the difference between the material score of the players in the area. The relative material advantage is similar, except the material advantage is further divided by the total material score in the area, resulting in a value between $[-1.0, 1.0]$. For both approaches, a player's material score is given by the weighted sum of the number of *men* and *kings* that are present in the area, following the formula depicted in equation 4.3. We chose to give a weight of 1.0 to each *men* and a weight of 2.0 to each *king*, as a *king* can move in both directions.

$$\text{MaterialScore} = 1.0 \times \#\ \text{Men} + 2.0 \times \#\ \text{Kings} \tag{4.3}$$

After testing both options, we noticed that the absolute version achieved slightly better results than the relative version. As the latter version returns a decimal value, we assume that the lack of discretization negatively impacted the generalization ability of the agent, thus decreasing its performance. In the end, we opted to use the absolute version for the rest of our work.

Using the aforementioned representation, each state is represented by a vector of 7 values (one for each area) and provides the agent with a clear overview of how the material is distributed across key areas of the board. By exploiting the similarities between different states, the agent is able to generalize its game knowledge and perform better in board settings that it has never faced before. This generalization of the state space enables the agent to make better decisions throughout its exploration, at the cost of losing some information.

4.4 Reward function

An important component of an RL problem is the reward signal that the agent receives at each time step, which is used to express the task that the designer wants the agent to learn. Using this signal as feedback, the agent can assess the outcome of its actions and learn which ones bring it closer to the intended goal.

It is critically important to formulate a good reward structure that encourages the agent to complete its goal, as it directly impacts its performance. In a zero-sum game, the goal is quite simple: learn a strategy that allows the agent to win against its opponent. Thus, a straightforward reward function would be to reward or punish the agent depending on whether it wins or loses the game, respectively. Equation 4.4 is an example of it:

$$\mathcal{R}(s,a,s') = \begin{cases} 2.0, & \text{if the agent wins} \\ -2.0, & \text{if the agent loses} \\ 0.0, & \text{otherwise} \end{cases} \tag{4.4}$$

where $\mathcal{R}$ represents the reward value that the agent receives after performing action a and transitioning from state s to state s'.

A reward signal as simple as this introduces a new problem: the agent only receives a non-zero feedback after his last action. During the rest of the game, the agent receives a zero reward and is unable to correctly assess its behavior and how it affected the outcome of the game. Thus, a sparse reward function can hinder the agent's capability to learn, as it requires the agent to explore the state space extensively to receive a positive feedback from the environment. In environments with large state spaces, such as in Checkers, the agent may need to play countless games to find a single winning sequence.

A possible solution is to use a denser reward function that returns a non-zero reward to the agent after each action taken, thus aiding it in recognizing and distinguishing good states from bad states. Designing such a function is often not trivial, as it requires domain knowledge about the problem at hand to know when to reward or punish the agent. Nevertheless, one could design a heuristic function that evaluates the state of the environment and use it to provide additional information, encouraging the agent to achieve certain sub-goals that might later help it reach the main goal. This approach is denoted reward shaping [43], as it modifies the original reward function by incorporating domain knowledge into it in order to help guide the agent in its exploration of the state space.

However, some problems might arise when applying reward shaping. If the new reward function is ill-formed, it might introduce bias to the agent's behavior and cause it to stray further away from its main goal, e.g., the agent might prefer capturing the opponent's pieces instead of winning the game. As the reward function directly affects how the agent explores and learns the environment, it can also cause the agent to converge to a sub-optimal policy. To address these issues, Ng & al. [30] introduced a formal framework to shape the reward function of a problem without changing the optimal policy. The framework transforms a sparse reward function $\mathcal{R}$ into a denser reward function $\mathcal{R}'$, as shown in equation 4.5:

$$\mathcal{R}'(s,a,s') = \mathcal{R}(s,a,s') + \mathcal{F}(s,a,s') \tag{4.5}$$

where $\mathcal{F}$, denoted the shaping reward function, is a real-valued function that provides additional information to the reward signal. In order to prevent the agent from getting distracted by sub-goals, the authors propose the use of a potential-based shaping reward function, according to equation 4.6:

$$\mathcal{F}(s,a,s') = \gamma\,\Phi(s') - \Phi(s) \tag{4.6}$$

where γ is the discounting factor and Φ is a function that evaluates and determines how good a certain state is. By calculating the difference between the values of Φ for the current and the last state, $\mathcal{F}$ provides information regarding whether the transition was beneficial for the agent and if the action taken drove it closer to its goal. The authors proved that the optimal policy associated with the shaped, denser reward function $\mathcal{R}'$ is equivalent to the optimal policy associated with the original, sparse reward function $\mathcal{R}$. Thus, if applied correctly, reward shaping can increase the learning speed without affecting the optimal policy learned by the agent.

In this work, we applied reward shaping to enhance the reward function in equation 4.4. In Checkers, the material advantage of a player has a strong correlation with winning or losing the game [36]. Thus, we devised a simple heuristic function that evaluates the board by calculating the relative material advantage, as shown in equation 4.7:

$$\Phi(s) = \frac{\text{MaterialScore}_{Agent} - \text{MaterialScore}_{Opp}}{\text{MaterialScore}_{Agent} + \text{MaterialScore}_{Opp}} \tag{4.7}$$

where $\text{MaterialScore}_{Agent}$ and $\text{MaterialScore}_{Opp}$ represents the material score of the agent and the opponent, respectively. A player's material score (MaterialScore) is calculated from the number and type of pieces that are currently on the board, as previously described in equation 4.3.

The heuristic function in equation 4.7 returns a real-valued number between $[-1.0, 1.0]$, representing the relative material advantage that a player has over its opponent. A positive number denotes that the agent has more material than its opponent, while a negative number signifies that the opponent has an advantage. By imbuing the heuristic function into $\mathcal{R}'$, the new reward function will encourage the agent to consider how the total material is distributed between both players and how its actions affect that distribution. Thus, the agent will prioritize the states where it has more material than its opponent.

We could have also incorporated more heuristics in the Φ value (equation 4.7), by rewarding other sub-goals such as trying to crown a *man* or threatening the opponent's material. However, we decided not to over-engineer the heuristic function, as we wanted to introduce as less as bias as possible to the agent's behavior and because the simplest version accomplished its goal in guiding the agent's exploration.

4.5 Opponent

In this dissertation, we considered the opponent as part of the environment that the agent is trying to learn. The environment is thus stochastic, as the transition between states depends not only on the agent's action but also on the opponent's action. The agent only perceives the resulting board after both its move and its opponent's move.

For the opponent, we decided to pit the agent against an adversary with some domain knowledge, as it provides more interesting results than playing against a random opponent with no reasoning behind its choice of actions. The opponent was implemented using Negamax, a recursive depth-first search algorithm that searches the game tree for the next best move. The algorithm is a simplified version of Minimax that relies on the zero-sum property of a two-player game, depicted in equation 4.8:

$$\max(\text{Score}_\text{A}, \text{Score}_\text{B}) = -\min(-\text{Score}_\text{A}, -\text{Score}_\text{B}) \tag{4.8}$$

where the score of one player A is the negation of the score of its adversary B.

A game can be thought of as a tree, named game tree, whose nodes are possible future game states. The root node of the game tree represents the current state of the game and each depth level denotes the consequent states that are reachable in a player's turn, alternating between both players. Each node has a value associated with it, representing how good the state is for a player. Then, the algorithm minimizes the worst-case potential loss, i.e., maximizes the chances of a player winning, assuming that its opponent is maximizing its own chances of winning. In contrast to Minimax, in Negamax both players try to maximize their score, where one of the players (opponent) maximizes the negated score.

In Checkers, the game tree has a branching factor of 2.8 and an average game length of 70 moves [37]. The number of nodes that need to be explored is given by the branching factor raised to the power of the number of turns, increasing exponentially with the depth of the tree. Thus, it is impractical to analyze the game tree completely. To deal with these limitations, we limited the opponent by only allowing it to look ahead a certain number of moves. When the algorithm reaches the maximum search depth allowed, it employs a heuristic function that evaluates the state. We further improved its performance by enhancing the algorithm with alpha-beta pruning, dramatically decreasing the number of nodes that need to be evaluated. The pseudo-code of the search algorithm is summarized in algorithm 2.

Algorithm 2 Negamax search

```
1:  procedure NEGAMAX(node, depth, $\alpha$, $\beta$, color)
2:      if depth = 0 or node is terminal then
3:          return evaluate(node)
4:      childNodes = getChildNodes(node)
5:      value = -∞
6:      for each child in childNodes do
7:          value = max(value, −negamax(child, depth − 1, $-\beta$, $-\alpha$, −color))
8:          $\alpha$ = max($\alpha$, value)
9:          if $\alpha \geq \beta$ then
10:             break
11:     return value
```

The evaluation of a game state (line 3 of algorithm 2) is done by an heuristic function based on a linear combination of independent weighted features of the board, as shown in equation 4.9:

$$\text{Evaluation} = turn \times \sum_i w_i \times f_i, \tag{4.9}$$

where each feature f_i is associated with a weight w_i, representing its importance in relation to the other features. Negamax requires a symmetric evaluation function, which returns a score relative to the player that is being evaluated. To do so, the score of each feature f_i is given by calculating the difference between the feature score of each player, as shown in equation 4.10. The result is then multiplied by a factor $turn$, where $turn = 1$ for the agent and $turn = -1$ for the opponent.

$$f_i = f_i(Agent) - f_i(Opp) \tag{4.10}$$

The choice of the features and the weights used in the evaluation determines how the opponent behaves. We considered four features of the board: the material of each player, the pieces with an unimpeded path to the back row, the pieces at risk of being captured and the position of the pieces. These features encourage the opponent to capture its adversary's pieces while defending its own, all while moving its pieces forward towards the center and the back row. The weights associated with each feature were chosen to prioritize certain features over others, e.g., the opponent prioritizes protecting a piece under attack rather than promoting a piece. The weights were hand-tuned after testing with different values and, as such, it is not expected that the opponent makes no mistakes or plays optimally. Nevertheless, the opponent plays smart and offers a fair challenge to the learning agent.

4.6 Summary

In this chapter, we defined the game of Checkers as a reinforcement learning problem. In order to address the high dimensionality of the problem, we explored possible representations for the state space and discussed their impact on how the agent perceives and learns the environment. Then, we explained how a simple reward function can be enhanced with domain knowledge without affecting the original optimal policy that the agent learns. Finally, we discussed the implementation of the opponent as part of the environment and provided a brief explanation regarding the Negamax algorithm and the evaluation function used.

Chapter 5

Improving exploration with QC

This chapter presents an exploration policy that aims to aid the agent in its exploration of the state space by improving the soft-max policy through the use of flags. First, it starts with a discussion about the possible applications of quantum computing to reinforcement learning in the domain of board games, providing an answer to the research question RQ1. Afterward, it describes the classical approach of the exploration policy, including the action selection process and the flag update mechanism. Then, the chapter highlights a few problems that might arise in the classical approach and details how the policy can be further enhanced using quantum computing. Finally, the chapter provides insight regarding the implementation of the operators used by the quantum algorithm and addresses its scalability, thus answering the research question RQ3.

5.1 Context

When applying quantum computing to a reinforcement learning problem, the first step is to determine which component will be enhanced, as it determines how the agent interacts with the environment. Both the agent and the environment can be classified as either classical or quantum, depending on how information is processed internally.

An option is to consider a quantum-accessible environment, as depicted in the work of Dunjko & al. [18]. Assuming the existence of a quantum oracle that is able to identify a winning sequence of moves, the agent could create a superposition of all possible sequences of moves and employ a Grover's search algorithm to amplify the probability of finding a sequence that wins the game. Using an oracle to mark the winning sequences, the agent could achieve a quadratic improvement over classical algorithms and speed up the learning process.

However, more often than not, designing such an oracle is not trivial. We could encode the entire environment and its state transitions using quantum transformations, but we would be limited by the number of qubits available and the complexity of the resulting quantum circuit. For a simpler game such as Connect4, which has a relatively small state space and a well-defined upper bound on the number of moves per game, it would be possible to pre-determine all winning sequences and directly encode them in an oracle. Nevertheless, such implementation reduces the agent's task to a simple search problem, where reinforcement learning is unnecessary. When considering more complex games such as Chess or Checkers, it is simply not possible for the agent to create a superposition of all possible sequences, nor is it efficient to encode all the solutions in the oracle, due to both the size of the state space and the high number of moves per game. Besides the complexity of the construction of the oracle, this approach also requires the environment to be deterministic, i.e., the opponent follows a fixed strategy, which may not be desirable.

Thus, we decided to divert our attention to quantum-enhanced agents and how quantum computing can aid in the decision-making of the agent throughout the learning process.

5.2 An improved exploration policy

In board games, the first and the most important step is to find a sequence of moves that results in a win. Only after learning how to beat its opponent can the agent start optimizing its strategy. However, finding that winning sequence is often not an easy task. It is necessary to employ a policy that helps guide the agent in the exploration of the environment, allowing it to make more informed decisions.

A common exploration policy is ε-greedy, where the agent chooses a random action with a probability ε and the current best action with a probability $1-\varepsilon$. The agent starts with a high probability of randomly choosing its actions and, over time, converges to a more greedy behavior. Its main drawback is that the agent chooses equally among all actions while exploring, even if some actions are known not to be optimal. In problems with huge state spaces, it often leads to longer convergence times. Another commonly used exploration policy is soft-max, or Boltzmann distribution, where the probability of choosing an action is proportional to its action value (Q value). The agent can adjust a temperature parameter to promote either exploration or exploitation. This policy allows the agent to focus on potentially promising actions and ignore previously bad actions. However, it requires knowledge of the domain of the action values in order to correctly choose the initial temperature parameter and how it decays.

In order to address the previous issues, we propose an exploration policy based on the soft-max action selection and further enhanced through the use of flags. A flag is a simple structure that serves as the short-term memory of the agent and is used to identify actions that it believes worth exploring. Its use was first proposed in the work of Briegel & al. [9], where it was demonstrated that it could significantly improve the performance of the agent at certain tasks. This exploration policy allows the agent to reflect on its choice of actions and helps it select more useful actions while exploring the environment. Thus, it makes the agent more efficient in its decision-making at the cost of spending more time in action selection.

5.2.1 Action selection

When reaching a certain state, the agent starts by determining how likely it is to choose each of the available actions based on its knowledge of how good they are. To do so, the agent employs a Boltzmann distribution and converts the Q value of each action into the probability of selecting said action, as described in equation 5.1:

$$\mathbb{P}(a \mid s) = \frac{e^{\beta Q(s,a)}}{\sum_{a_i \in A(s)} e^{\beta Q(s,a_i)}} \tag{5.1}$$

where s represents a state, $\mathcal{A}(s)$ represents the set of possible actions available to the agent when in state s, a is an action such that $a \in \mathcal{A}(s)$ and β denotes the inverse temperature. Using this distribution, the agent directly incorporates its knowledge into the action selection process, thus leading to a more informed decision. The inverse temperature parameter controls how greedy the agent behaves. A lower value results in an even probability distribution, while a higher value further stresses the differences between actions.

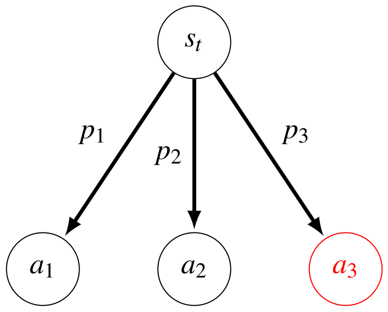


Figure 5.1: The action selection process can be visualized as a directed two-layered network, where the root node is the current state and the leaf nodes are all the available actions. The probability of transitioning from the root node to each leaf node is given by the Boltzmann distribution. In this example, the action a_3 is the only flagged action.

In the next step of the process, the agent repeatedly samples an action based on the probabilities calculated earlier until it obtains a flagged action (see line 3 of algorithm 3). Such procedure can be thought of as a simple random walk on a directed two-layered network, as depicted in figure 5.1, where the agent repeatedly hops from the root node to a leaf node until a flagged action is reached. The probability of choosing a flagged action may become low during the learning process, e.g., the sum of the probability of choosing a flagged action is less than 10%. Thus, we introduced a fixed parameter R, which denotes the maximum number of available iterations to the agent. If no flagged action is obtained before reaching the maximum number of tries, the agent chooses the last sampled action. The action selection process is summarized in algorithm 3.

5.2.2 Flag update

Initially, all actions are flagged, as the agent considers all transitions to be equally good. During the learning process, the agent continuously assesses the impact of an action on the environment

Algorithm 3 Action selection process

```
1: procedure ACTION_SELECTION($R$)
2:     repeat for a maximum of R times
3:         $selectedAction \leftarrow$ sample based on Boltzmann distribution
4:         if $selectedAction$ is flagged then
5:             break
6:     return $selectedAction$
```

and either adds or removes its flag, depending on whether the outcome was considered good or bad, respectively. When the last remaining flag is removed, the process restarts and all the other actions except the last one are flagged again (see lines 6 and 7 of algorithm 4). Therefore, for any given state, there will always be at least one flagged action.

When following this exploration policy, a problem that arises is how to evaluate the outcome of an action and when is the action considered good. To address this problem, we explored several different approaches. The most naive approach is only to consider the immediate reward associated with the action. The flag is either added or removed depending on whether the perceived reward was positive or negative, respectively. This approach works in simple environments where the agent is only concerned about the immediate reward, e.g., the agent either accomplishes or fails its task after an action. However, board games often require the agent to deal with delayed rewards, where it might have to choose bad actions to gain the upper hand over its opponent, e.g., sacrifice a piece to later promote another piece. The agent might also face a situation where all actions have a negative reward associated, which will cause it to remove and reset the flags constantly. Thus, when updating the flags, it is necessary to consider both the immediate and the delayed rewards of an action.

A more knowledgeable approach is to evaluate the outcome of an action using either the TD-Error or the TD-Target from the Q-learning's update rule, according to equation 5.2:

$$Q(S_t,A_t) = Q(S_t,A_t) + \alpha \left(\overbrace{\underbrace{R_t + \gamma \max_a Q(S_{t+1},a)}_{\text{TD-Target}} - Q(S_t,A_t)}^{\text{TD-Error}}\right) \tag{5.2}$$

where $Q(S_t,A_t)$ denotes the current Q value of the action A_t in the state S_t, R_t is the reward received, α is the learning rate and γ is the discounting factor. The TD-Error determines how close (or far) the new Q value estimate is from the previous estimate. However, this metric is highly unstable, as it can greatly vary when the agent explores new states and recalculates the estimate using the latest information. Also, a negative TD-Error does not imply that the action had negative consequences, only that the agent overestimated the actual Q value. On the other hand, the TD-Target denotes the most recent estimate of the Q value and represents the agent's current knowledge regarding a certain action. It is a good metric for the flag update mechanism, but it ignores the learning rate and can thus also suffer from the same stability issues as the TD-Error, especially at the beginning of the exploration phase. Therefore, we decided to use the actual Q value (see line 2 of algorithm 4), as it has the same advantages as the TD-Target while also considering the agent's learning rate. The flag update process is described in pseudo-code 4.

Algorithm 4 Flag update process

```
1: procedure FLAG_UPDATE($qValues$, $flaggedActions$, $currentState$, $selectedAction$)
2:     if $qValues[currentState, selectedAction] < 0.0$ then
3:         Remove $selectedAction$ from $flaggedActions$
4:     else
5:         Add $selectedAction$ to $flaggedActions$
6:     if $flaggedActions$ is empty then
7:         Add all actions to $flaggedActions$ except $selectedAction$
```

5.3 A quantum approach

Although the previous approach offers several advantages over other exploration policies, it has its flaws. First, the action selection process is directly dependent on the probability of obtaining a flagged action. If such probability is relatively small, either due to a high number of possible actions or a high value on the temperature parameter, the agent might constantly reach the maximum number of tries and not be able to output a suitable action. Furthermore, the agent might also reacts slowly to changes in the environment, e.g., when the opponent changes its strategy. In such cases, the agent might find itself choosing non-flagged actions for an extended period while the flags slowly update accordingly.

In order to solve these issues, we enhanced the action selection process using a quantum algorithm, inspired by the work of Paparo & al. [33] on reflecting projective simulation. The goal is to amplify the probability of choosing a flagged action to near unity (100%) while maintaining the relative probabilities between said actions, thus achieving the distribution presented in equation 5.3:

$$\widetilde{\mathbb{P}}(a \mid s) = \begin{cases} \dfrac{\mathbb{P}(a \mid s)}{\sum_{a_i \in \mathcal{F}(s)} \mathbb{P}(a_i \mid s)}, & \text{if } a \text{ is a flagged action} \\ 0, & \text{otherwise} \end{cases} \tag{5.3}$$

where $\widetilde{\mathbb{P}}(a \mid s)$ is the re-normalized probability distribution with support only over the flagged actions and $\mathcal{F}(s)$ denotes the set of flagged actions.

It is important to note that the algorithm we employ in this work is not optimal in searching for a flagged action. Instead, its task is to achieve a good approximation of the distribution mentioned above and then sample an action based on it, for which the algorithm is optimal [33].

5.3.1 Action selection

Upon reaching a certain state, the quantum agent internally constructs an ergodic and reversible Markov chain P based on a two-layered network, as shown in figure 5.2. The states of the chain correspond to the available actions and the probability of transitioning to each state is the same as the original probability of choosing the corresponding action. Any state associated with a flagged action remains flagged in the Markov chain.

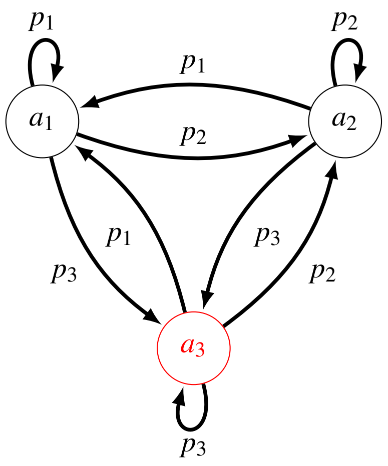


Figure 5.2: An ergodic and reversible Markov chain, built from the two-layered network depicted in figure 5.1.

During the action selection process, the quantum agent performs a quantum walk over the Markov chain. The agent starts by preparing and encoding the stationary distribution π of P in its quantum state, according to equation 5.4:

$$|\pi\rangle = \sum_{a\in\mathcal{A}(s)} \sqrt{\pi_a}\,|a\rangle \tag{5.4}$$

where π_a denotes the probability of the action a under the stationary distribution π. Each action a is represented by its respective basis state $|a\rangle$ in a N-dimensional Hilbert space $\mathcal{H}$, where N is the number of states in the Markov chain. Then, the agent sequentially applies two operators to the quantum state. The first operator performs a reflection over the set of flagged actions and shifts the phase of the basis states that correspond to a flagged action, as described in equation 5.5.

$$\text{ref}(\mathcal{F}) : |a\rangle \to \begin{cases} -|a\rangle, & \text{if } a \in \mathcal{F}(s) \\ |a\rangle, & \text{otherwise} \end{cases} \tag{5.5}$$

The second operator performs a reflection over the initial stationary distribution, thus amplifying the amplitude of the basis states that correspond to a flagged action. Said reflection is defined in equation 5.6:

$$\text{ref}(\pi) = 2|\pi\rangle\langle\pi| - I \tag{5.6}$$

where $|\pi\rangle$ denotes the encoded stationary distribution of P. The process closely resembles Grover's search algorithm, as each iteration rotates the quantum state closer to a state with support only over the flagged actions. After applying the required number of iterations, the resulting quantum state approximately encodes the distribution described in equation 5.3, following equation 5.7:

$$Q^T |\pi\rangle = \sin((2T+1)\theta) \sum_{x \in \mathcal{F}(s)} \sqrt{\frac{\pi_x}{\varepsilon}} |x\rangle + \cos((2T+1)\theta) \sum_{y \notin \mathcal{F}(s)} \sqrt{\frac{\pi_y}{1-\varepsilon}} |y\rangle \tag{5.7}$$

where T is the number of iterations, ε is the probability of the flagged actions under the stationary distribution π, such that $\varepsilon = \sum_{a \in \mathcal{F}(s)} \pi_a$, and $\theta = \arcsin(\sqrt{\varepsilon})$ [8]. Finally, the agent measures its quantum register and, if flagged, outputs the measured action. Otherwise, the process restarts.

The quantum algorithm requires an average of $O(1/\sqrt{\varepsilon})$ iterations, providing a quadratic speedup over the classical version. The process is summarized in algorithm 5.

Algorithm 5 Quantum action selection process

```
 1: procedure QUANTUM_ACTION_SELECTION($P$, $\varepsilon$, $R$)
 2:     Encode the stationary distribution $\pi$ of $P$ in the circuit
 3:     repeat for a maximum of $R$ times
 4:         $T \leftarrow$ uniformly random in $[0, 1/\sqrt{\varepsilon}]$
 5:         repeat $T$ times
 6:             Apply the oracle operator to the circuit
 7:             Apply the diffusion operator to the circuit
 8:         $action \leftarrow$ measure the quantum register
 9:         if $action$ is flagged then
10:             break
11:     return $action$
```

5.3.2 Implementation

In a Szegedy-based quantum walk such as the one described in subsection 3.2, the quantum algorithm acts on two copies of a N-dimensional Hilbert space $\mathcal{H}$. These two copies are used to encode all possible transitions from each state of the chain, as a unitary transformation acting on a single copy cannot do so. As such, the algorithm requires $k = 2 \times \text{int}(\log_2(N))$ qubits for a Markov chain with N states.

In our work, the Markov chains are always built from a two-layered network. The resulting chain is denoted rank-one, as all the columns of its transition matrix are the same and equal to the stationary distribution. As such, the transition probabilities are identical for every state of the chain, making it possible to encode them onto a quantum state using the same unitary transformation. This is an important property of the chain, as it dramatically simplifies the implementation of the original algorithm. The simplified version requires less controlled transformations and half the number of qubits, enabling a more efficient construction of the quantum circuit while maintaining all the benefits of the original algorithm. Furthermore, it also decreases the complexity associated with the implementation of the operators used in the algorithm, as we will discuss in the rest of the subsection. All in all, these simplifications allow us to scale the quantum algorithm to board games with bigger action spaces.

Probability encoding

One of the most important building blocks of the algorithm is the transformation U_π, which encodes the stationary distribution π onto a quantum state, as shown in equation 5.8. This is the process identified in line 2 of algorithm 5.

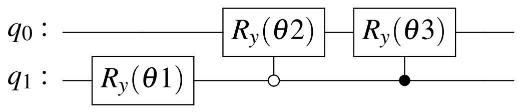
$$U_\pi : |0\rangle \rightarrow |\pi\rangle \tag{5.8}$$

The transformation can be constructed using a sequence of controlled single-qubit rotations around the y-axis, through a process denoted coherent controlization [17]. By following this process, we are able to construct a probability unitary that encodes any given probability distribution. The angles used in the rotations are calculated recursively from the probability distribution that we wish to encode. This process is described in algorithm 6.

Algorithm 6 Calculate the rotation angles from a probability distribution

```
1: procedure PROB_TO_ANGLES($distr$, $previous = 1.0$)
2:     if len($distr$) = 2 then
3:         return [$distr$[0] / $previous$]
4:     $lhs$, $rhs \leftarrow$ split($distr$, 2)
5:     $angles \leftarrow$ [sum($lhs$) / $previous$]
6:     Append prob_to_angles($lhs$, sum($lhs$)) to $angles$
7:     Append prob_to_angles($rhs$, sum($rhs$)) to $angles$
8:     return $angles$
```

For larger Markov chains with more states, the unitary U_π can be recursively decomposed into smaller, controlled unitaries that act on a subset of the qubits. An example can be seen in figure 5.3, where the circuits presented are used to encode the probability distribution of a Markov chain with up to 4 and 8 states, respectively. As it can be seen, the second circuit is built through recursive applications of the first circuit.



(a) Circuit implementing the operator U_π for a Markov chain with up to 4 states.

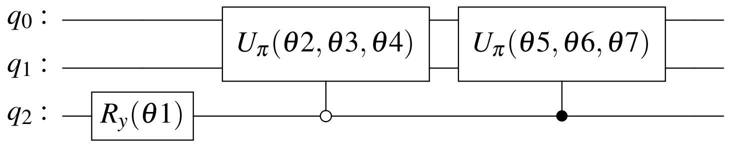

q_0 :
q_1 :
q_2 :
$R_y(\theta1)$
$U_\pi(\theta2,\theta3,\theta4)$
$U_\pi(\theta5,\theta6,\theta7)$
(b) Circuit implementing the operator U_π for a Markov chain with up to 8 states.

Figure 5.3: Coherent controlization. The filled dot denotes that the transformation is applied to the target qubits when the control qubit is $|1\rangle$.

Oracle

The reflection $\mathrm{ref}(\mathcal{F})$ is implemented as an oracle that marks the basis states that correspond to a flagged action (see line 6 of algorithm 5). The reflection can be described by the diagonal matrix depicted in equation 5.9:

$$\mathrm{ref}(\mathcal{F}) = \begin{pmatrix} (-1)^{f(a_0)} & 0 & \cdots & 0 \\ 0 & (-1)^{f(a_1)} & \cdots & 0 \\ \vdots & 0 & \ddots & \vdots \\ 0 & 0 & \cdots & (-1)^{f(a_n)} \end{pmatrix} \tag{5.9}$$

where $f(a) = 1$ if $a \in \mathcal{F}(s)$ and $f(a) = 0$ otherwise. As the matrix is dependent on which actions are flagged, the implementation of the reflection differs from case to case. A generic way to do it is to construct a classical Boolean oracle and then convert it to a phase shift oracle by encapsulating the last qubit between X and H gates. Although simple to implement, this approach requires an extra qubit and, thus, is less efficient than more specialized implementations [20].

Walk operator

Another important operator required by the algorithm is the Szegedy's walk operator W_P, which acts on a superposition of all states of the chain and performs a coherent quantum walk (see line 7 of algorithm 5). In the original algorithm, a special property of this operator was exploited, using a phase estimation algorithm to construct an approximation of the diffusion operator $\mathrm{ref}(\pi)$. However, due to the simplifications mentioned earlier, the W_P can be defined as described in equation 5.10:

$$W_P = U_\pi \, D_0 \, U_\pi^\dagger \tag{5.10}$$

where D_0 denotes a reflection over the quantum state $|0\rangle$ and $U_\pi^\dagger$ is the conjugate of the operator U_π. Together, these transformations reverse the initial state preparation, perform a reflection over the state $|0\rangle$ and then recompute the state preparation, achieving an exact reflection over the stationary distribution $|\pi\rangle$, i.e, $\mathrm{ref}(\pi) = W_P$. Therefore, it is no longer necessary to approximate the diffuse operator as in the original algorithm, reducing the need for ancilla qubits and controlled operations.

5.4 Summary

This chapter started with a brief discussion on the applications of quantum computing to reinforcement learning, where we distinguish two possible approaches: oracularized environments or quantum-enhanced agents, thus answering the research question RQ1. Then, we introduced an improved exploration policy that aims to address some issues that other common policies have. The presented policy enhances the soft-max action selection through the use of flags, which identify actions that are worth exploring. We described its action selection process and the flag update mechanism, providing some insights about how the agent assesses the outcome of an action.

Following, we discussed some shortcomings of the classical approach and proposed a quantum algorithm to further improve the policy. The employed quantum algorithm replaces the previous action selection with a quantum walk over a Markov chain built from a two-layered network. Finally, we explored the implementation of the required operators in a quantum circuit and provided an answer to the research question RQ3.

Chapter 6

Experiments and results

This chapter discusses the experiments implemented to test and compare the classical and the quantum agents. It starts by describing the experiments performed, as well as the model parameters used by the agents in each experiment. Then, it provides a comparison and analysis of the agents' performance. Finally, it draws some conclusions regarding the advantages of the quantum approach.

6.1 Experiments

We set up different experiments to compare the agents' performance and assess the impact that the exploration policy has on their learning process. The experiments are separated in two sets, each performed in two different board sizes: a 6x6 board and an 8x8 board. The 6x6 board provided a simpler environment with a small state space for the agents to explore. It was meant to serve as a baseline for comparison on how the agents' performance scales to more complex tasks. On the other hand, the 8x8 board presented a much more challenging problem to the agents, dramatically increasing the size of the state and action spaces to be explored. This environment tests the agents' ability to learn a winning strategy, as a suitable exploration policy is required to help guide the exploration. The length of the learning process, i.e., the number of games played in each experiment, was chosen based on the size of the board.

Opponent

As mentioned in chapter 4, the opponent was implemented using a Negamax search algorithm. Its intelligence is determined by the number of moves that it is allowed to look ahead in the game tree. When limited to a single move, the opponent acts as a purely greedy player, always choosing the next best action without any regard for its adversary's future moves. As the number of moves increases, the opponent plays more carefully and considers the possible counter-moves from its adversary, thus providing a more significant challenge to the learning agents.

In each set of experiments, the agents played against an opponent with a look-ahead of 1 and 3 moves, allowing us to gauge how the difficulty presented by the opponent influences the learning process. In the second set of experiments, the agents also played against an opponent that changes its strategy during the learning process, evaluating how the agents adapt to a changing environment. Although we wanted to pity the agents against even more difficult opponents, i.e., opponents with a look-ahead of more moves, it was not possible due to time constraints.

Metrics

During the experiments, several metrics are collected and later used to analyze the agents' performance, such as the number of wins/losses/draws, the total rewards per game, the number of explored states and the number of moves played per game. As the exploration policies employed by the agents are probabilistic, each experiment was run multiple times to obtain an accurate evaluation of the learning process. Thus, we performed 20 independent trials for each experiment and calculated the mean performance between the trials for each agent. These metrics were then used to generate the charts presented later, allowing us to assess and compare the agents' performance during the learning process. In order to reduce the fluctuation of the data and ease its analysis, we smoothed the data by calculating the moving average of each metric, using a moving window of 500 games.

6.2 Agent parameters

In order to compare the classical and the quantum versions of the exploration policy, we developed a total of 6 Q-learning agents. All agents have the same parameters α and γ, as the goal is to assess the impact that the parameter R and the action selection have on the results. The parameters used in the 6 developed agents are presented in table 6.1.

The classical agents, denoted with the prefix CQL, refer to the agents that employ the classical approach of the exploration policy. Analogously, the quantum agents, denoted with the prefix QQL, refer to the agents that employ the quantum approach of the exploration policy. For the same approach (classical and quantum), the agents differ from each other with respect to the value of the parameter R used, as presented in the mentioned table.

Agent	α	γ	R	Action selection
CQL-1	0.4	1.0	5	Classical
CQL-2	0.4	1.0	10	Classical
CQL-3	0.4	1.0	100	Classical
QQL-1	0.4	1.0	5	Quantum
QQL-2	0.4	1.0	10	Quantum
QQL-3	0.4	1.0	100	Quantum

Table 6.1: Agents and their parameters

The inverse temperature parameter β, used in equation 5.1, is also the same for all agents. Although the proposed exploration policy alleviates the importance of this parameter, it still impacts the agent's behaviour. Thus, setting up an appropriate initial value and growth rate can improve the learning efficiency of the agent. After some trial and error, we employed different growth rates depending on the size of the board and the type of opponent, as shown in figure 6.1.

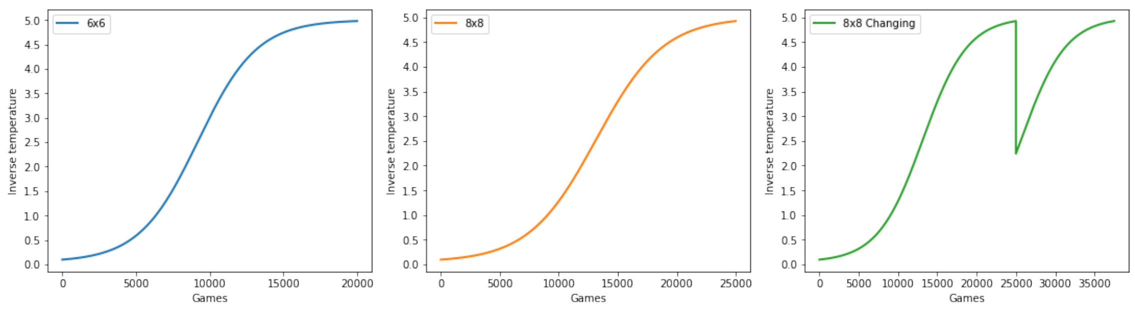


Figure 6.1: Growth rate of the inverse temperature parameter for each set of experiments.

As depicted in the figure, the growth rate of the parameter β follows a logistic curve. The agent starts with a low value of β, which steadily increases during the learning process, thus promoting the exploration of the state space. As the agent learns the environment, it starts converging to a greedier behavior.

For each set of experiments, we used different growth rates depending on the length of the learning process. For the experiments in the 6x6 board, the parameter β converges to its upper limit after 20000 games. Similarly, for the experiments in the 8x8 board, the parameter reaches the upper limit after 25000 games. In the experiment with a changing environment, the parameter follows the same growth rate as the other experiments in the 8x8 board but, after 25000 games, it undergoes a soft reset to allow the agent to adapt to the new opponent's strategy.

6.3 Analysis of the results

In this section, we will analyze the results obtained from each experiment and draw some conclusions. As mentioned previously, the first set of experiments was performed in a simpler version of the game, played on a 6x6 board. Meanwhile, the second set of experiments was performed in the regular version of the game, played on an 8x8 board. This section only includes the bar charts representing the results obtained at the end of the learning process. All the other metrics used to evaluate the agents can be found in appendix A.

Experiment 1.1 - Look-ahead 1 move opponent (6x6 board)

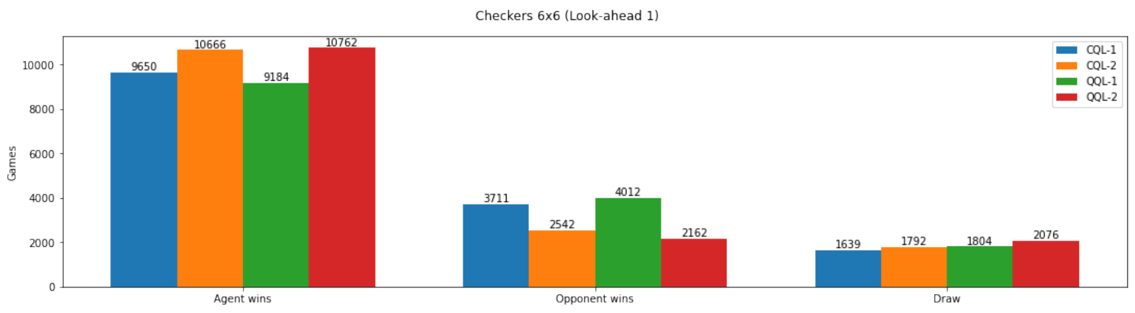


Figure 6.2: Number of wins/losses/draws of the agents playing in a 6x6 board against an opponent with a look-ahead of 1 move.

In the first experiment, all agents (classical and quantum) learned a strategy that wins against the opponent, with the classical agents winning roughly the same number of games as their quantum counterparts, as shown in figure 6.2. However, it is possible to note some differences that occurred during the learning process by further analyzing figure A.1.

It is clear that the agents CQL-2 and QQL-2 converged slightly faster than the remaining agents and learned a winning strategy while exploring fewer states. The difference from the other agents comes from their greater R value, which compels them to be more selective in their choice of actions. When comparing these two agents, both achieved a similar performance across all metrics. However, the quantum agent revealed a slight advantage at the beginning of the learning process, where it quickly learned how to not lose against its opponent.

Furthermore, it is also interesting to note that CQL-1 and QQL-1 learned a strategy that wins in fewer moves than the other agents. Their behavior is a consequence of their lower R value, which enabled the agents to explore more states of the environment and thus find a sequence of actions that further optimizes their strategy.

Overall, the results demonstrate that all agents were able to learn the environment and win against the opponent. However, it is not clear whether the quantum agents outperformed the classical agents. Nevertheless, we can conclude that the agents with a higher R had a more efficient exploration of the state space.

Experiment 1.2 - Look-ahead 3 moves opponent (6x6 board)

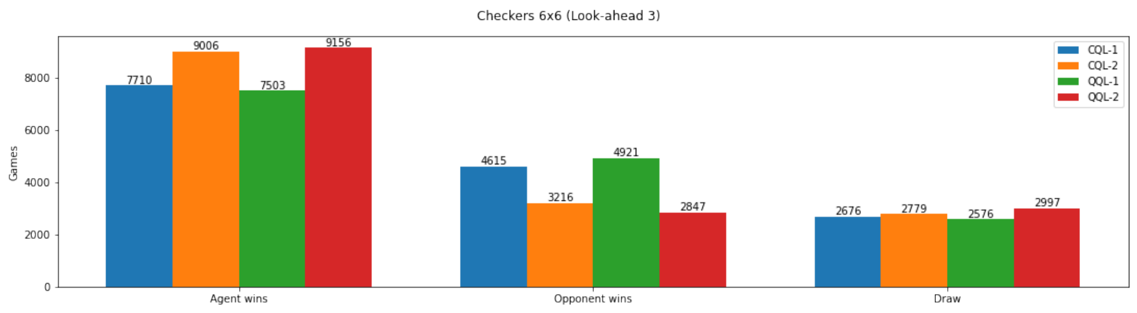


Figure 6.3: Number of wins/losses/draws of the agents playing in a 6x6 board against an opponent with a look-ahead of 3 moves.

In the next experiment, the results were very similar to the previous experiment, where the classical agents and their quantum counterparts presented similar performance across most metrics, as detailed in figures 6.3 and A.2. The most noticeable difference is that all agents required more games to converge to a winning strategy, causing the agents to lose and draw more games when compared to the prior experiment. These changes in the results are expected and a direct consequence of playing against a more knowledgeable opponent. As the opponent plays smarter and is more careful with its moves, a careless action from the agent can quickly lose the game.

Much like the previous experiment, it is not possible to affirm the advantages of the quantum approach. However, we can still conclude from the results that the parameter R has a positive impact on the learning process of an agent, increasing the convergence rate while requiring less exploration of the state space.

Experiment 2.1 - Look-ahead 1 move opponent (8x8 board)

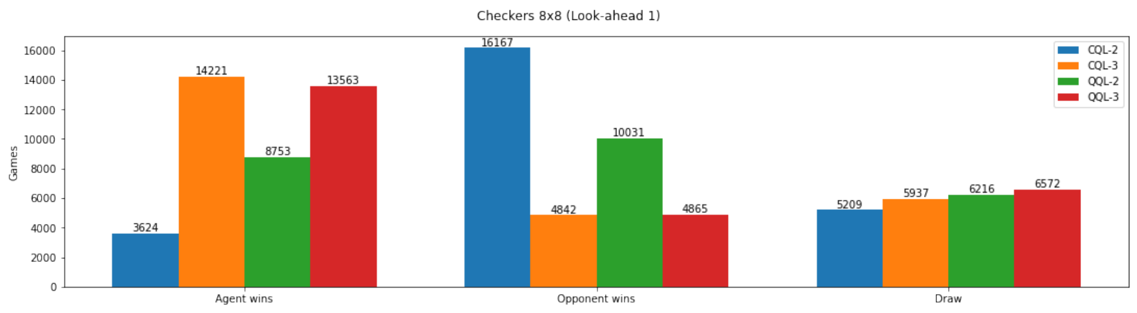


Figure 6.4: Number of wins/losses/draws of the agents playing in a 8x8 board against an opponent with a look-ahead of 1 move.

In this experiment, the agents were presented with a much more challenging environment. Therefore, the results obtained, presented in figures 6.4 and A.3, accentuated the differences between the agents, brought by the action selection method used and the parameter R.

Starting by assessing the impact of the parameter *R*, the results demonstrate that the agents with a higher value were more successful than those with a lower value. By analyzing the performance of the agents CQL-2 and QQL-2, we quickly notice that these agents played multiple games without displaying any improvement in their behavior, revealing a slower convergence rate. Meanwhile, the agents CQL-3 and QQL-3 were able to successfully learn and converge to a winning strategy, with a similar performance across most metrics. Furthermore, and akin to the previous experiments, these agents also achieved these results while exploring much fewer states than the other agents. Therefore, we can conclude that a higher *R* value enables the agents to take full advantage of the action selection process and its flags, allowing them to better re-think their choice of actions and achieve a more efficient exploration of the state space. Although its impact on the learning process was somewhat noticeable in the previous experiments, its advantages become evident in a more complex environment with large state and action spaces.

When comparing the classical agents with their quantum counterparts, the results confirm that the quantum agents demonstrate a better learning efficiency. The advantages of the quantum approach are especially evident in agents with smaller *R* values, as seen between CQL-2 and QQL-2. In this example, the quantum agent learned a winning sequence after 25000 games, while the classical version barely achieved a 60% winning probability. As both agents explored roughly the same number of states, we can infer that the action selection method influences how an agent explores the state space and chooses its actions, i.e., has an impact on how the agent learns the environment.

After analyzing the results of the agents CQL-3 and QQL-3, we can also conclude that the benefits of the quantum action selection diminish as the value of *R* increases. However, although both agents demonstrate similar performance metrics, the quantum version theoretically requires less tries to output a flagged action. Therefore, it can reduce the amount of time spent during the decision-making process, which can prove valuable in time-constrained tasks. Unfortunately, it is not possible to quantify that difference using the metrics collected.

Experiment 2.2 - Look-ahead 3 moves opponent (8x8 board)

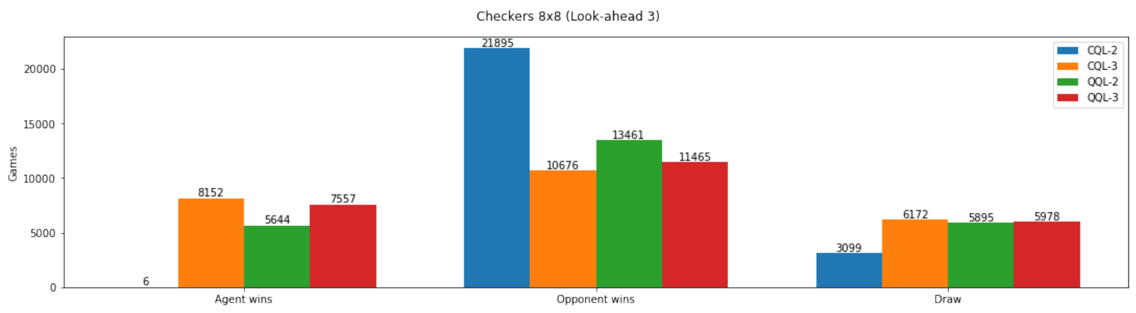


Figure 6.5: Number of wins/losses/draws of the agents playing in a 8x8 board against an opponent with a look-ahead of 3 moves.

For this experiment, we again increased the difficulty of the opponent. The results further stressed the differences between the agents, resulting in drastic changes in their performance, as detailed in figures 6.5 and A.4.

First, the agent CQL-2 was not able to learn the environment. The agent did not show any signs of progress throughout the entire learning process, losing most of its games. Meanwhile, its quantum counterpart managed to achieve a win probability of approximately 70%, demonstrating the advantages of the quantum action selection method.

Analogous to the previous experiments, the best-performing agents were CQL-3 and QQL-3, with both agents displaying similar win probabilities. However, the latter revealed a slight advantage over the former, as it was able to reach a win probability of around 90% near the end of the learning process. Consequently, although the classical version demonstrated a slightly better convergence rate throughout the whole process, the quantum approach achieved a better strategy in the end.

Experiment 2.3 - Changing opponent (8x8 board)

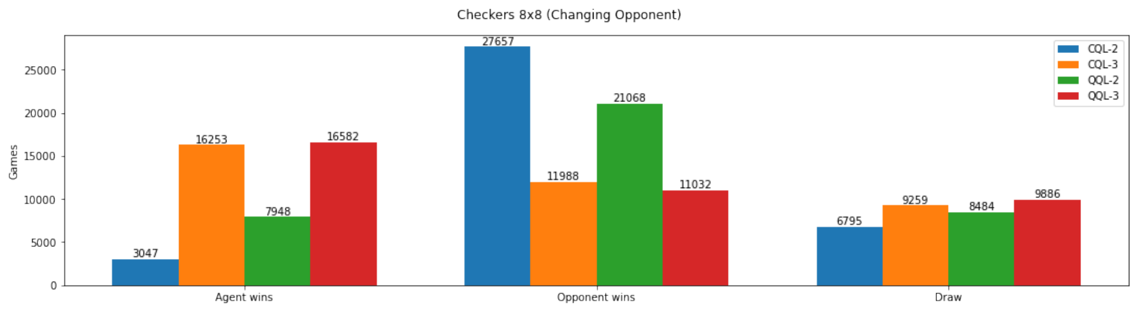


Figure 6.6: Number of wins/losses/draws of the agents playing in a 8x8 board against an opponent that changes its strategy.

For the final experiment, the agents faced an opponent that changes its strategy over time. Initially, the opponent starts with a look-ahead of 1 move, allowing the agents to converge to a winning strategy. Then, after a few thousand games, the opponent plays more intelligently and uses a look-ahead of 3 moves.

During the first part of the process, the agents' learning curve followed the same results obtained in experiment 2.2, as expected. When the opponent changed its strategy, the agents' performance plummeted as they now had to adapt their policy. The results, shown in figures 6.6 and A.5, demonstrate that only the agents CQL-3 and QQL-3 were able to do so, as the remaining agents were unable to re-learn a new strategy in such a short number of games.

The reason behind these differences in performance is easy to explain. After converging and following a policy for a prolonged number of games, the probability of choosing a flagged action becomes quite high. When the opponent changes its strategy, some flags get reset and the new flagged actions are now associated with a low probability, as it takes time for the agent to update their corresponding Q values. In this scenario, a high R value allows the agents to repeat the action selection process multiple times and, eventually, obtain a flagged action. Thus, the parameter R influences how fast an agent reacts to a change in the environment, at the cost of more time spent in decision-making. As the quantum algorithm provides a quadratic speedup in obtaining a flagged action, the quantum agent QQL-3 displayed slightly better adaptability than its counterpart, quickly converging to around 60% win probability.

6.4 Results overview

In the first set of experiments, all agents performed similarly and quickly learned the environment, even against stronger opponents. However, the second set of experiments introduced the agents to a more complex task with a larger state space. These experiments stressed the differences between the agents and allowed us to better distinguish the benefits introduced by the quantum algorithm.

Regarding the research question RQ2, we conclude that both the parameter R and the action selection method significantly impact the agent's learning efficiency. The parameter R promotes a more efficient exploration of the state space, enabling the agent to repeat the action selection process until it chooses a suitable action. The quantum approach complements and builds upon these advantages, providing a quadratic speedup for outputting a flagged action, thus improving the agent's decision-making, especially for lower R values.

6.5 Summary

In this chapter, we discussed the experiments performed to analyze the advantages and disadvantages of the classical and the quantum approach. First, we provided some information regarding the opponents used in the experiments, the parameters of each agent and the metrics collected throughout the learning process. Following, we presented an analysis of the results obtained from each experiment and answered the research question RQ2.

Chapter 7

Conclusion

With the development of quantum computing and quantum information, their application to reinforcement learning problems has been a topic of study in the last decade. Multiple approaches have been proposed, achieving a quantum speedup in different environment-agent interaction settings. These approaches have shown promising results, revealing quadratic improvements in the learning speed of the agent.

7.1 Main contributions

In this dissertation, we intended to assess the impact that quantum computing can have on the learning process of an agent when applied to the domain of board games. To that end, we started by defining Checkers as an RL problem and all its components, i.e., the state representation, the reward function used and the opponent. Then, we proposed an exploration policy and further enhanced it using quantum computing. Said policy aims to improve the agent's decision-making process, allowing it to reflect upon its choice of actions and rethink its decision as needed. Finally, both classical and quantum approaches were tested in different scenarios. The results obtained demonstrate that the quantum agents had a more successful exploration phase and were able to outperform their classical counterparts in learning efficiency.

During our work, we were also able to answer the following research questions:

RQ1. **How can quantum computing be applied to RL in the domain of board games?**
In chapter 5, we discussed the possible applications of quantum computing to a reinforcement learning task. One approach considered was to construct a oracularized environment to which the agent could query in superposition. However, this approach has drawbacks, especially when considering more complex games where the construction of the oracle operator is not trivial. Thus, this dissertation focused on employing quantum computing to enhance an agent's decision-making process.

RQ2. **How does the quantum approach impacts the learning efficiency of an agent?**
As demonstrated by the results obtained in chapter 6, the quantum algorithm promotes a more efficient exploration of the state space, achieving a theoretical quadratic speedup in obtaining a

suitable action. Thus, a quantum agent demonstrates a better convergence rate over its classical counterpart while exploring fewer states. Furthermore, a quantum agent also displays better adaptability in changing environments. The advantages of the quantum approach become more evident as the size of the state and action spaces increases.

RQ3. **Is the quantum approach scalable for more complex board games?**
The complexity associated with the implementation of the quantum algorithm scales with the size of the action space of the problem, as the number of qubits required increases with the number of possible actions in a particular state. Furthermore, the length of the resulting quantum circuit is determined by the number of applications of the oracle and the walk operator. Depending on the total probability of the flagged actions, the number of iterations required might over-extend the quantum circuit and, thus, negatively impact the construction of the said circuit.

7.2 Future work

At the end of this dissertation, it is possible to identify certain areas of improvement, especially regarding the experiments used to test and compare the agents' performance.

First off, some aspects of the proposed exploration policy were not explicitly evaluated due to the lack of collected metrics, such as the number of iterations required to obtain a flagged action. Analyzing the differences in those aspects would allow us to better compare the classical and quantum approaches.

Regarding the experiments, there are also other scenarios that would be interesting to study. A possible experiment would be to pit the agents against human players to assess their learning process and the quality of the strategy learned. Unlike the opponents tested in chapter 6, a human player is constantly changing and improving its strategies, allowing us to further test the agent's adaptability to changes in the environment and their ability of generalization. Another interesting experiment would be to test both approaches in a more complex board game, e.g., Chess, and study how their performance scales in problems with larger state and action spaces.

Appendix A

Results

This chapter presents the results obtained in the experiments, including the metrics used to evaluate the agents' performance and obtain some conclusions. As mentioned in chapter 6, we employed a moving average to smooth the data and facilitate its analysis, using a window of 500 games.

Each figure is composed of five different graphs. The graph on top presents the number of wins, draws and losses of each agent at the end of the learning process. The following graph shows the total reward received by the agents per game, providing a better analysis of the agents' learning efficiency and convergence rate throughout the learning process.

The bottom three graphs display some minor statistics that allow us to further analyze the agents' behavior. The first one displays the win probability of each agent, calculated from the percentage of games that each agent won in the last 100 games. The second graph presents the number of unique states explored by the agents, providing insight regarding how each agent explored the state space. Finally, the third graph describes the length of the games, i.e., the number of moves made by each agent per game.

A.1 Results for experiment 1.1

In this experiment, the agents played a total of 20000 games in a 6x6 board against an opponent with a look-ahead of 1 move. The obtained results are shown in figure A.1, displaying the first 15000 games.

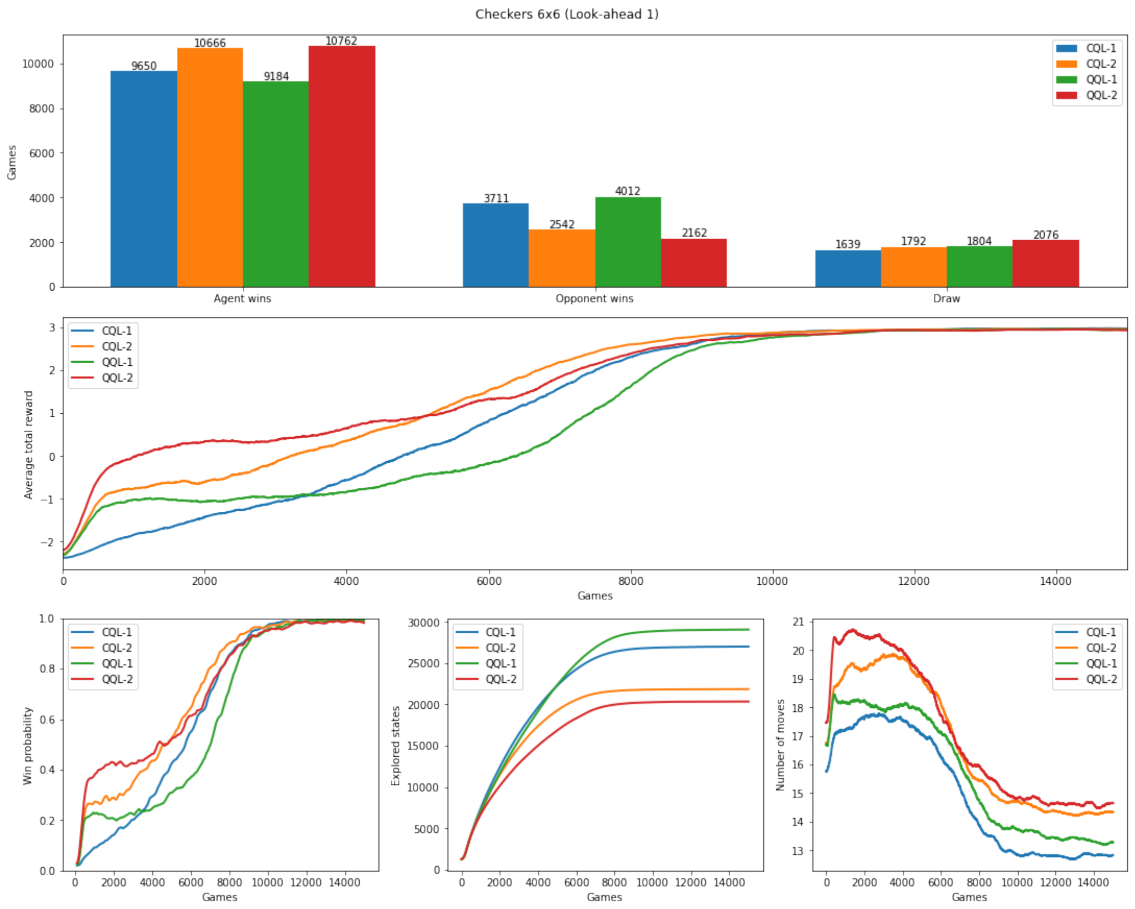


Figure A.1: Results of the agents playing in a 6x6 board against an opponent with look-ahead of 1 move.

A.2 Results for experiment 1.2

In this experiment, the agents played a total of 20000 games in a 6x6 board against an opponent with a look-ahead of 3 moves. The obtained results are shown in figure A.2, displaying the first 15000 games.

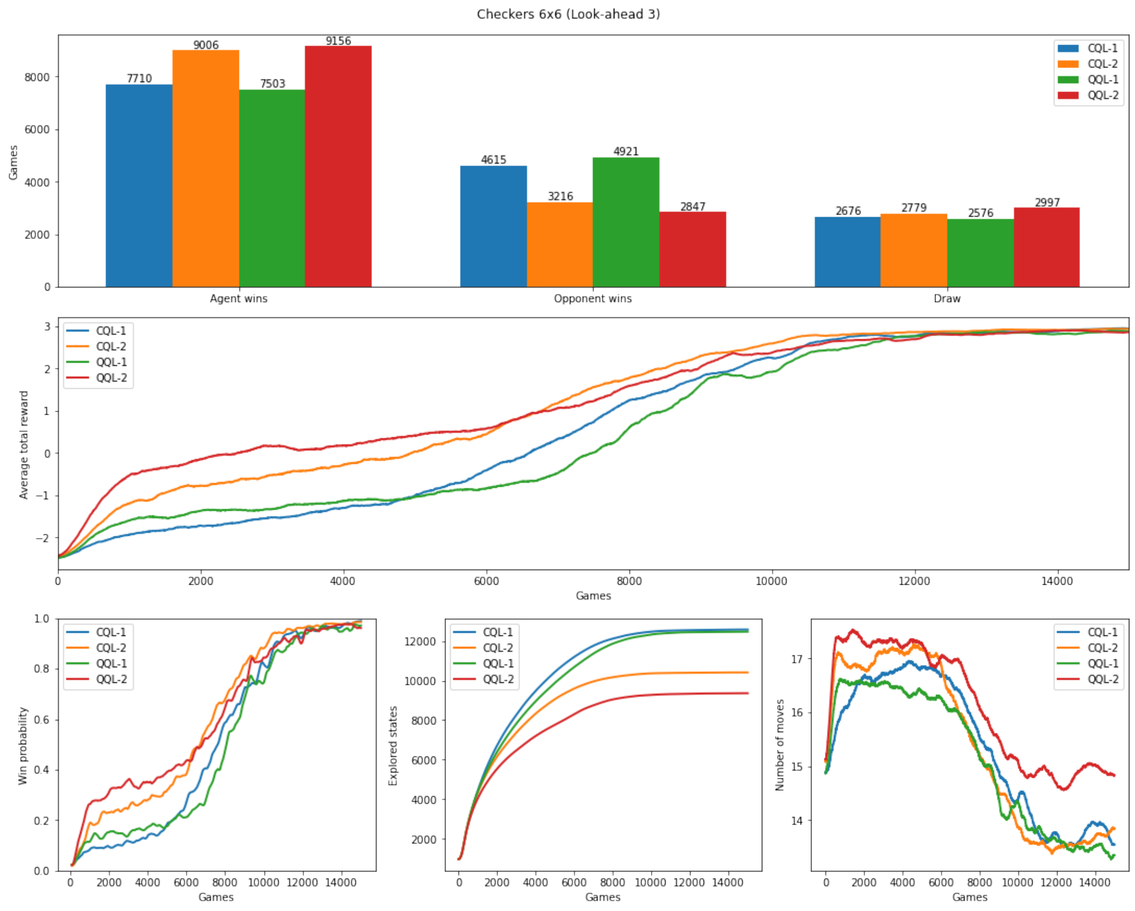


Figure A.2: Results of the agents playing in a 6x6 board against an opponent with look-ahead of 3 moves.

A.3 Results for experiment 2.1

In this experiment, the agents played a total of 25000 games in a 8x8 board against an opponent with a look-ahead of 1 move. The obtained results are shown in figure A.3.

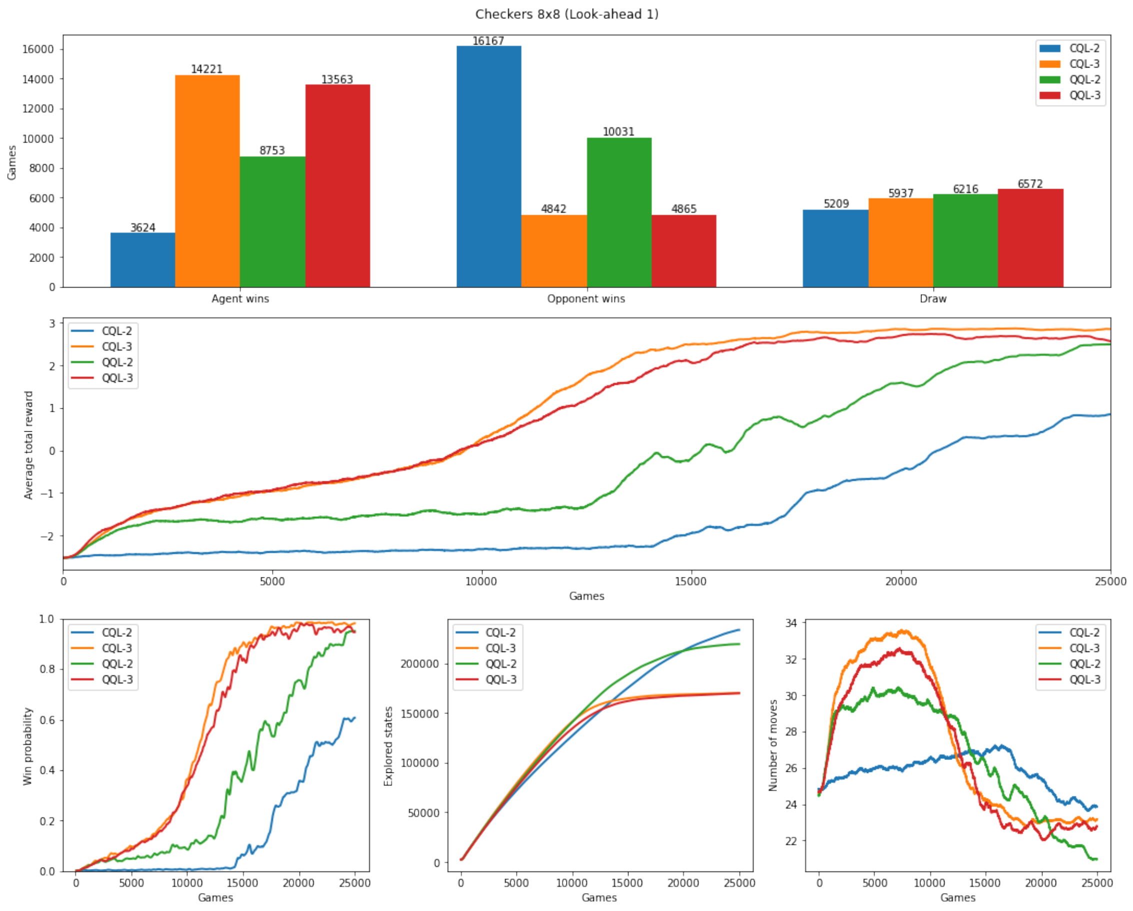


Figure A.3: Results of the agents playing in a 8x8 board against an opponent with look-ahead of 1 move.

A.4 Results for experiment 2.2

In this experiment, the agents played a total of 25000 games in a 8x8 board against an opponent with a look-ahead of 3 moves. The obtained results are shown in figure A.4.

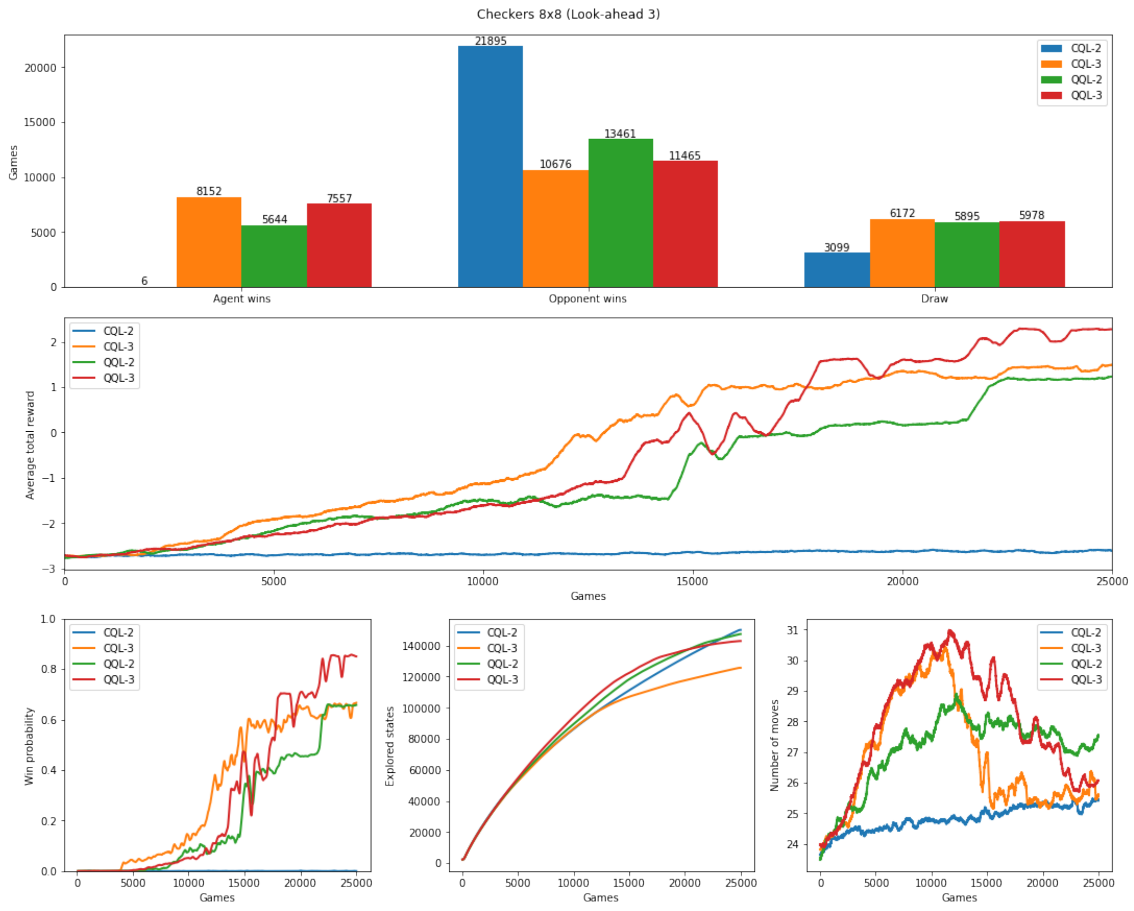


Figure A.4: Results of the agents playing in a 8x8 board against an opponent with look-ahead of 3 moves.

A.5 Results for experiment 2.3

In this experiment, the agents played a total of 37500 games in a 8x8 board against an opponent that changes its strategy over time. The opponent starts with a look-ahead of 1 move and, after 25000 games, uses a look-ahead of 3 moves. The obtained results are shown in figure A.5.

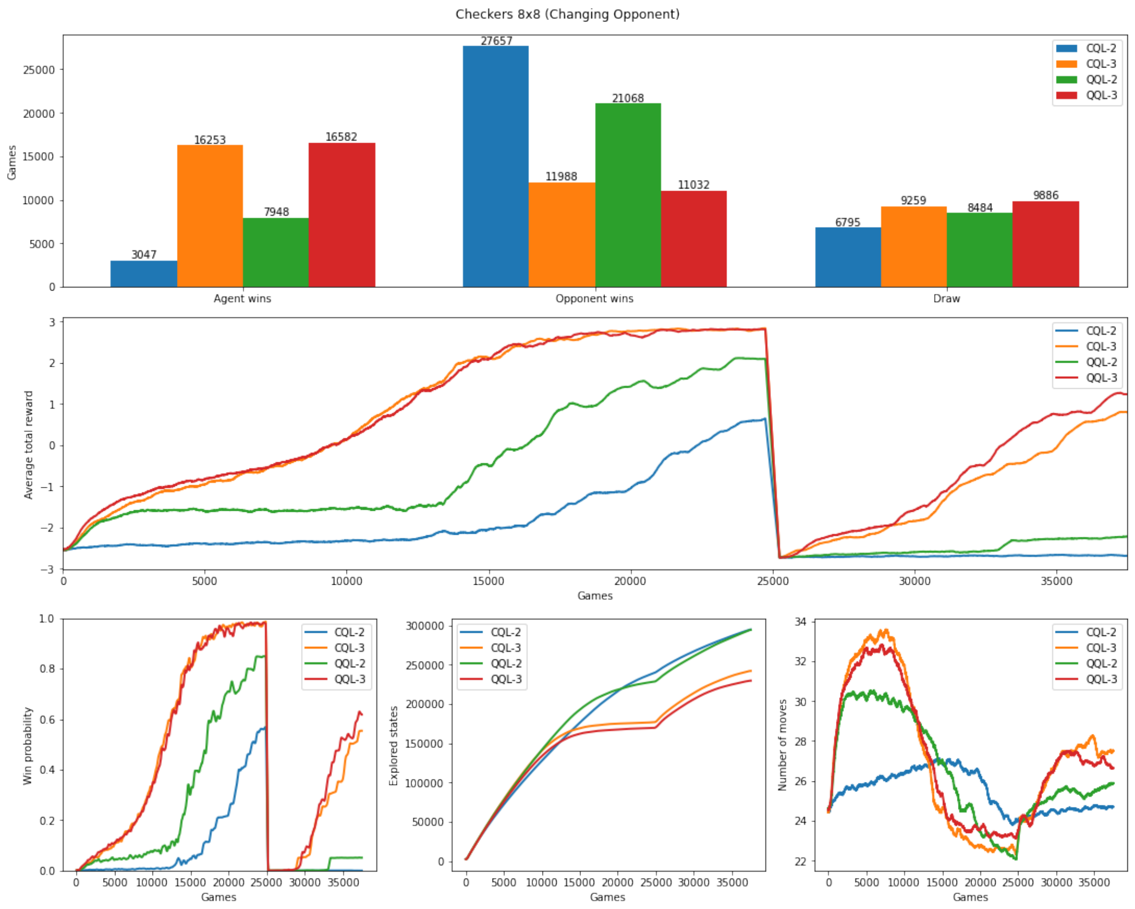


Figure A.5: Results of the agents playing in a 8x8 board against an opponent with look-ahead of 1 and 3 moves.

References

[1] Héctor Abraham, AduOffei, Rochisha Agarwal, Ismail Yunus Akhalwaya, Gadi Aleksandrowicz, Thomas Alexander, Matthew Amy, Eli Arbel, Arijit02, Abraham Asfaw, et al. Qiskit: An open-source framework for quantum computing, 2019.

[2] Francisco Albarrán-Arriagada, Juan C. Retamal, Enrique Solano, and Lucas Lamata. Measurement-based adaptation protocol with quantum reinforcement learning. *Physical Review A*, 98, 10 2018.

[3] Andris Ambainis. Quantum walks and their algorithmic applications. *International Journal of Quantum Information*, 1, 12 2003.

[4] Andris Ambainis, Julia Kempe, and Alexander Rivosh. Coins make quantum walks faster. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1099—1108. Society for Industrial and Applied Mathematics, 2005.

[5] Richard E. Bellman and Rand Corporation. *Dynamic Programming*. Rand Corporation Research Study. Princeton University Press, 1957.

[6] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. *Nature*, 549, 9 2017.

[7] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46(4-5):493–505, Jun 1998.

[8] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Quantum Computation and Information*, page 53–74, 2002.

[9] Hans J. Briegel and Gemma De las Cuevas. Projective simulation for artificial intelligence. *Scientific Reports*, 2, 12 2012.

[10] Chunlin Chen, Daoyi Dong, Yu Dong, and Qiong Shi. A quantum reinforcement learning method for repeated game theory. In *2006 International Conference on Computational Intelligence and Security*. IEEE, 11 2006.

[11] Samuel Yen-Chi Chen, Chao-Han Huck Yang, Jun Qi, Pin-Yu Chen, Xiaoli Ma, and Hsi-Sheng Goan. Variational quantum circuits for deep reinforcement learning. *IEEE Access*, 8, 2020.

[12] Jens Clausen and Hans J. Briegel. Quantum machine learning with glow for episodic tasks and decision games. *Physical Review A*, 97, 2 2018.

[13] Andrew W. Cross, Lev S. Bishop, John A. Smolin, and Jay M. Gambetta. Open quantum assembly language, 2017.

[14] Paul A. M. Dirac. A new notation for quantum mechanics. *Mathematical Proceedings of the Cambridge Philosophical Society*, 35(3):416–418, 1939.

[15] Daoyi Dong, Chunlin Chen, Hanxiong Li, and Tzyh-Jong Tarn. Quantum reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38, 10 2008.

[16] B. L. Douglas and J. B. Wang. Efficient quantum circuit implementation of quantum walks. *Physical Review A*, 79, 5 2009.

[17] Vedran Dunjko, Nicolai Friis, and Hans J. Briegel. Quantum-enhanced deliberation of learning agents using trapped ions. *New Journal of Physics*, 17(2):023006, Jan 2015.

[18] Vedran Dunjko, Jacob M. Taylor, and Hans J. Briegel. Quantum-enhanced machine learning. *Phys. Rev. Lett.*, 117:130501, September 2016.

[19] Albert Einstein, Boris Podolsky, and Nathan Rosen. Can quantum-mechanical description of physical reality be considered complete? *Physical Review*, 47, 5 1935.

[20] Caroline Figgatt, Dimitri Maslov, Kevin A. Landsman, Norbert M. Linke, Shantanu Debnath, and Christopher Monroe. Complete 3-qubit grover search on a programmable quantum computer. *Nature Communications*, 8, 12 2017.

[21] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing - STOC '96*. ACM Press, 1996.

[22] Peter Høyer. Arbitrary phases in quantum amplitude amplification. *Physical Review A*, 62(5), Oct 2000.

[23] Alexei Y. Kitaev. Quantum measurements and the abelian stabilizer problem. *arXiv e-prints*, nov 1995.

[24] Ji-An Li, Daoyi Dong, Zhengde Wei, Ying Liu, Yu Pan, Franco Nori, and Xiaochu Zhang. Quantum reinforcement learning during human decision-making. *Nature Human Behaviour*, 4, 3 2020.

[25] Owen Lockwood and Mei Si. Playing atari with hybrid quantum-classical reinforcement learning, 2020.

[26] Owen Lockwood and Mei Si. Reinforcement learning with quantum variational circuits, 2020.

[27] Frédéric Magniez, Ashwin Nayak, Jérémie Roland, and Miklos Santha. Search via quantum walk. *SIAM Journal on Computing*, 40, 1 2011.

[28] Alexey A. Melnikov, Adi Makmal, Vedran Dunjko, and Hans J. Briegel. Projective simulation with generalization. *Scientific Reports*, 7, 12 2017.

[29] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[30] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, 1999.

[31] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[32] Julio Olivares-Sánchez, Jorge Casanova, Enrique Solano, and Lucas Lamata. Measurement-based adaptation protocol with quantum reinforcement learning in a rigetti quantum computer. *Quantum Reports*, 2, 5 2020.

[33] Giuseppe Davide Paparo, Vedran Dunjko, Adi Makmal, Miguel Angel Martin-Delgado, and Hans J. Briegel. Quantum speedup for active learning agents. *Physical Review X*, 4, 2014.

[34] Eleanor Rieffel and Wolfgang Polak. *Quantum Computing: A Gentle Introduction*. The MIT Press, 1st edition, 2011.

[35] G. Rummery and Mahesan Niranjan. On-line q-learning using connectionist systems. *Technical Report CUED/F-INFENG/TR 166*, 11 1994.

[36] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

[37] Jonathan Schaeffer, Neil Burch, Yngvi Bjornsson, Akihiro Kishimoto, Martin Muller, Robert Lake, Paul Lu, and Steve Sutphen. Checkers is solved. *Science*, 317, 9 2007.

[38] Kishore S. Shenoy, Dev Y. Sheth, Bikash K. Behera, and Prasanta K. Panigrahi. Demonstration of a measurement-based adaptation protocol with quantum reinforcement learning on the ibm q experience platform. *Quantum Information Processing*, 19, 5 2020.

[39] Neil Shenvi, Julia Kempe, and Katherine B. Whaley. Quantum random-walk search algorithm. *Physical Review A*, 67, 5 2003.

[40] Peter W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*. IEEE Comput. Soc. Press, 1994.

[41] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–359, 10 2017.

[42] Shivani Singh, Cinthia H. Alderete, Radhakrishnan Balu, Christopher Monroe, Norbert M. Linke, and C. M. Chandrashekar. Universal one-dimensional discrete-time quantum walks and their implementation on near term quantum hardware, 1 2020.

[43] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Second edition, 2018.

[44] Mario Szegedy. Quantum speed-up of markov chain based algorithms. In *45th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, 2004.

[45] Quantum AI team and collaborators. Cirq, October 2020.

[46] Salvador Elías Venegas-Andraca. Quantum walks: a comprehensive review. *Quantum Information Processing*, 11, 10 2012.

[47] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8, 5 1992.

[48] Marco Wiering and Martijn Otterlo. *Reinforcement Learning: State-Of-The-Art*, volume 12. Springer-Verlag Berlin Heidelberg, 01 2012.

[49] Shaojun Wu, Shan Jin, Dingding Wen, and Xiaoting Wang. Quantum reinforcement learning in continuous action space, 2021.