1 2 9 0

UNIVERSIDADE Ð
COIMBRA

Filipe Miguel Fonseca dos Santos

EMPOWERING CLASSICAL AI WITH QUANTUM COMPUTING

Dissertation in the context of the Master in Informatics Engineering, specialization in Software Engineering, advised by Professors Luís Macedo and João Fernandes and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

July 2022

Filipe Miguel Fonseca dos Santos

EMPOWERING CLASSICAL AI WITH QUANTUM COMPUTING

1 2 9 0 UNIVERSIDADE Ð COIMBRA

Faculty of Sciences and Technology
Department of Informatics Engineering

Empowering Classical AI with Quantum Computing

Filipe Miguel Fonseca dos Santos

Dissertation in the context of the Master in Informatics Engineering, specialization in Software Engineering, advised by Professors Luís Macedo and João Fernandes and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

July 2022

This page is intentionally left blank.

Abstract

Reinforcement Learning (RL) is a Machine Learning (ML) branch, in which an agent interacts with an environment by trial and error. Since RL can work without knowledge of the problem domain, it has the advantage of not needing previously labelled training data to function. As a result, it has found success in many areas, such as robotics and games. RL is frequently paired with Neural Networks (NNs), resulting in Deep Learning approaches, which can work well even when dealing with large state spaces.

At the same time, Quantum Computing is an area that has the potential to surpass classical supercomputers at specific tasks. While it is unknown when this potential will be realized, it is important to research possible applications. Furthermore, as current quantum hardware is noisy and quantum simulations are difficult to perform for more complex systems, it is especially relevant to figure out practical use cases for Quantum Computing in the near future. Quantum ML is one of the areas that shows potential to work under the current quantum context. Specifically for Quantum RL, there is the prospect of achieving a better balance between the exploration of the state space and the exploitation of the knowledge acquired, as seen in some recent research related with quantum tagged action selection, which has been applied to the board game of Checkers.

In this work, we applied quantum tagged action selection to the RL context of Connect Four, extending the scope of this technique to other board games. To do so, we paired it with an offline Deep Learning method, which was key in dealing with the state-space complexity of the problem. We tested both classical and quantum agents against a Randomized Negamax opponent. The results obtained showed a superior performance in comparison with a standard ϵ-greedy approach. Furthermore, the quantum version of the flagged action selection led to better training efficiency than its classical counterpart. Since going second is a major disadvantage in this board game, we also analysed the performance of the agents that trained as player 2, finding less conclusive but still ultimately positive results.

Keywords

Reinforcement Learning, Quantum Computing, Connect Four.

This page is intentionally left blank.

Resumo

Aprendizagem por reforço é um dos ramos da aprendizagem computacional, em que um agente interage com um ambiente por tentativa e erro. Como a aprendizagem por reforço pode funcionar sem conhecimento do domínio do problema, tem a vantagem de não precisar de dados de treino previamente etiquetados. Como consequência, tem tido sucesso em várias áreas, como a da robótica e a dos jogos. Aprendizagem por reforço é frequentemente acompanhada por redes neuronais, formando abordagens de aprendizagem profunda, que podem funcionar mesmo quando lidando com grandes espaços de estados.

Ao mesmo tempo, a computação quântica é uma área que tem o potencial de superar supercomputadores clássicos em tarefas específicas. Apesar de ser desconhecido quando este potencial será realizado, é importante investigar possíveis aplicações. Para além disso, como o hardware quantum atual tem ruído e simulações quânticas são difíceis de realizar para sistemas mais complexos, é especialmente relevante descobrir casos de uso práticos para a computação quântica num futuro próximo. Aprendizagem computacional quantum é uma das áreas que mostra potencial de funcionar dentro do contexto quantum atual. Especificamente para a aprendizagem por reforço quantum, há a perspetiva de conseguir um melhor equilíbrio entre a exploração do espaço de estados e a exploração do conhecimento obtido, como visto em trabalhos recentes relacionados com a quantum tagged action selection, que foi aplicada ao jogo de tabuleiro das Damas.

Neste trabalho, aplicámos quantum tagged action selection dentro do contexto da aprendizagem por reforço no 4 em Linha, estendendo o âmbito desta técnica a outros jogos de tabuleiro. Para o fazer, combinámo-la com um método de aprendizagem profunda offline, que foi a chave para lidar com a complexidade do espaço de estados do problema. Testámos agentes clássicos e quânticos contra um adversário que utilizou Randomized Negamax. Os resultados obtidos mostraram um desempenho superior em comparação com uma abordagem ϵ-greedy comum. Além disso, a versão quântica da flagged action selection levou a uma melhor eficiência ao treinar do que a sua versão clássica. Como o jogador 2 tem uma grande desvantagem neste jogo de tabuleiro, também analisámos o desempenho dos agentes que treinaram como jogador 2, encontrando resultados menos conclusivos, mas ainda assim positivos.

Palavras-Chave

Aprendizagem por Reforço, Computação Quântica, 4 em Linha.

This page is intentionally left blank.

Contents

This page is intentionally left blank.

Acronyms

CNN Convolutional Neural Network. 19

DCRL Deep Curriculum Reinforcement Learning. 21

DP Dynamic Programming. 6, 8–10

DQWL Discrete Quantum Walk on a Line. 16

DRL-PER Deep Reinforcement Learning - Prioritized Experience Replay. 21

MC Monte Carlo. 6, 9, 10

MCTS Monte Carlo Tree Search. 27, 32

MDP Markov Decision Process. 4, 6–8, 27, 31, 51

ML Machine Learning. iii, 1, 2, 24

NISQ Noisy Intermediate-Scale Quantum. 2, 21, 22

NLP Natural Language Processing. 1, 19

NN Neural Network. iii, 1, 3, 20–22, 32, 47, 48, 51

PQC Parameterized Quantum Circuit. 20, 21

RL Reinforcement Learning. iii, 1–4, 6–8, 19–23, 25, 27, 31–33, 41, 47, 51

SVQC Single-qubit-based Variational Quantum Circuit. 21

TD Temporal-Difference. 6, 10, 31, 32

VQC Variational Quantum Circuit. 21

This page is intentionally left blank.

List of Figures

This page is intentionally left blank.

List of Tables

This page is intentionally left blank.

Chapter 1

Introduction

In this chapter, the theme of this dissertation is introduced. We start by briefly describing what Reinforcement Learning (RL) is and looking at its current context, while also addressing Deep RL and explaining why Quantum Computing can be relevant for this Machine Learning (ML) branch. We then go over the motivation for this thesis, covering why Quantum RL is a relevant area to tackle, from a more practical point of view. We also cover how that leads to this dissertation's main topic. Afterwards, we look at the objectives of this work and the relevant research questions. This is followed by the contributions. Finally, we examine the structure of this document.

1.1 Context

RL is an area of ML in which an agent learns by interacting with an environment through trial and error, in a sequence of steps that take it from one state to another, by following a certain policy. Due to its ability to learn without relying on domain knowledge, RL can be useful in various fields, such as games, Natural Language Processing (NLP), robotics and even art [19].

The policy followed by the agent dictates which actions it can take in each state, and with which probability. According to the decisions it takes, the agent receives scalar rewards based on its performance. These rewards are then used to update the agent's policy, allowing it to keep learning better actions for each situation.

RL is especially interesting since it typically starts with no knowledge of the problem domain, which can give it key advantages in relation to other ML approaches, such as supervised learning. In the case of supervised learning, training data and its corresponding outputs are used for training, with the intent of learning the relation between them. However, getting enough high-quality hand labeled data can not only be inefficient, but also ultimately create an upper bound on the performance of the agent. For instance, learning from the decisions of top players in a game might bound the peak performance of the learner not too far from human capabilities.

However, when it comes to problems in the real world, it is very possible that the data being used is too high-dimensional, which can ruin the effectiveness of RL. A typical solution when dealing with that kind of data is to enhance RL with Deep Learning, which involves the use of Neural Networks (NNs), in order to represent states and approximate functions [19]. This Deep RL approach has been quite successful, with Alpha Go Zero [26], which avoids human data entirely, being able to completely outclass the classical Alpha Go algorithm that could already defeat the best players at the game of Go.

At the same time, we have the prospect of Quantum Computing having the potential to surpass classical computation in specific tasks in the future, being able to solve them in a reasonable time frame. While it is unknown how long it will take for this breakthrough to happen, there have been theoretical works and experiments that support this. For instance, Google's 53 quantum bit (qubit) quantum computer has been used to demonstrate that it can run certain calculations in a very small fraction of the time they would take on a classical supercomputer (200 seconds instead of the estimated thousands of years) [11]. All of this theoretical potential motivates the exploration of Quantum Computing.

On the other hand, we have to take into consideration the current limitations of Quantum Computing. We must first consider the limited number of qubits that current devices can take advantage of. Since adding more qubits exponentially increases the computational power of a quantum computer, it is only natural that having a reduced number of them heavily reduces their potential advantages. Furthermore, Quantum Computing is extremely prone to noise. This is something that can be alleviated with quantum error correction, but doing so requires even more qubits. As such, current quantum computations are noisy, which narrows the usefulness of Quantum Computing down to algorithms that are capable of functioning despite the noise. Simulations with a large number of qubits are also very difficult, since simulating such high-dimensional quantum environments is computationally intensive for classical computers.

However, Quantum Computing is gradually becoming more accessible due to simulators and remote access to quantum hardware through platforms such as IBM-Quantum, which has led to more research in the area. Due to properties inherent to quantum physics, it is expected that quantum approaches will be able to further improve RL through quadratic speedups and better balance between exploration and exploitation [32].

1.2 Motivation

Due to the limitations imposed by current quantum hardware, it is important to consider the practical applications of Quantum Computing in the near future. As quantum ML is one of the areas expected to be able to reach meaningful results in this Noisy Intermediate-Scale Quantum (NISQ) era, it is natural to think about what can be achieved in its three main branches: supervised learning, unsupervised learning and RL. The latter remains the least explored of the three [27], which incites interest in attempting to figure out what can be achieved in its domain.

Recent work has shown that quantum approaches and quantum-inspired algorithms can lead to different benefits for RL [27, 32], such as needing less parameters to execute the same tasks.

Furthermore, it has been shown that Quantum RL can be applied to the board game of Checkers, improving the exploration of the state space [30]. In other words, it helps dealing with the exploration-exploitation problem of RL, which involves having to correctly balance how much an agent should explore new states with how much it should exploit its knowledge by choosing the state it believes to be better. This work raises the question of whether similar or different benefits can be achieved in other board games. As Connect Four is a board game with slightly lower complexity than Checkers, we believe it is a prime candidate for this type of analysis.

1.3 Objectives and Research Questions

For this dissertation, the game of Connect Four was used as a way to test the potential of Quantum Computing for RL in the near future. Specifically, we wanted to verify what kinds of advantages could be found when using a quantum algorithm for the RL problem of learning how to win at Connect Four. For instance, if the training efficiency was better than classical approaches. We also took advantage of the quantum tagged action selection policy used in [30], to examine whether the improvements to the exploration of the state space achieved for Checkers could be found in a different board game, generalizing the applicability of this technique.

Considering that Connect Four is a game that has already been solved using classical brute-force methods, the aim of this study is not to create a quantum approach that always finds the optimal move. Rather, it is to develop a quantum enhanced algorithm that can be competitive with relatively powerful classical approaches.

Thus, we can think of the following research questions.

- Research Question 1: In what way can a quantum approach improve the exploration of the state space for the RL problem of Connect Four?
- Research Question 2: What changes need to be made so that a similar approach to the one used for Checkers [30] can be applied to Connect Four, in order to better explore the state space?
- Research Question 3: How does this exploration of the state space compare to a standard ϵ-greedy approach?
- Research Question 4: Considering that going second is a significant disadvantage in Connect Four, how does the quantum approach fare in this scenario?

1.4 Contributions

For this work, the following contributions were achieved:

- The application of quantum tagged action selection to the Connect Four RL problem, combining it with a classical NN and extending its scope to other board games.
- The comparison between quantum tagged action selection and its classical version, as well as a standard exploration policy, known as ϵ-greedy.
- The analysis of the effects of scaling the quantum algorithm to the more complex scenario of learning how to win as player 2.

1.5 Document Outline

Following this introduction, the subsequent chapters are as follows:

- Chapter 2 goes over some relevant background for this thesis, covering: RL and its key topics; a brief introduction to Quantum Computing and some of its algorithmic applications; the game of Connect Four.

- In Chapter 3, we look at literature related with classical Deep RL and Quantum RL, followed by an analysis of quantum frameworks.
- In Chapter 4, we present the methodology used in this work, covering the Randomized Negamax opponent and the representation of Connect Four as a Markov Decision Process (MDP), as well as the Q-Learning [28] approach employed. From there, the exploration-exploitation problem is introduced and we describe the classical and quantum approaches used to solve it.
- In Chapter 5, we detail the parameters used for the Randomized Negamax opponent, the training and testing strategies utilized, and the experimental setup for the agents.
- In Chapter 6, the results of the experiments for the player 1 and the player 2 agents are shown, as well as a discussion about them. We also present answers for the research questions, taking into account the work developed and the results obtained.
- In Chapter 7, we provide a conclusion to this thesis.

This page is intentionally left blank.

Chapter 2

Background Knowledge

In this chapter, background knowledge useful for understanding this work is given. We start with an introduction to Reinforcement Learning (RL) and then go into some of its key topics. These include Markov Decision Processes (MDPs), Dynamic Programming (DP), Monte Carlo (MC) methods and Temporal-Difference (TD) learning. From there, a brief overview of Quantum Computing and some of its key concepts is given. This can aid the reader in understanding some basic quantum properties, which are then useful for following along the literature review and the rest of this work. We also cover two specific algorithmic applications of Quantum Computing and explain how they work. These applications are Grover's algorithm and quantum walks, both of which are related with the quantum exploration policy used in this work. Lastly, we go over the game of Connect Four, covering its components, rules and conditions for reaching the end of a game and deciding a winner.

2.1 Reinforcement Learning

We will now go over a basic overview of RL. This will be mostly based on Richard Sutton and Andrew Barto's book [28], which provides a very complete introduction to the topic.

When it comes to RL, we consider the problem of learning by interacting with an environment. This involves a learning agent, who interacts with the environment so it can achieve a specific goal. This is done through the execution of actions, which can influence the environment by modifying its state. Instead of explicitly telling the agent what to do, its learning process is guided through the use of rewards. These rewards are signals, which can inhibit or promote the execution of certain actions, shaping how the agent behaves.

When we think about action selection, we also have to consider the concepts of exploration and exploitation. When an agent is choosing new actions, to figure out which ones yield the best results, it is exploring the environment. Meanwhile, when it is choosing the actions it believes to be better, it is exploiting its current knowledge. These are two concepts that need to be very well balanced, so the agent can sufficiently explore the environment, while taking enough advantage of what it has learned.

While the environment and the agent are considered the main elements of RL, there are also four sub-elements involved: the policy, the reward signal, the value function and the model of the environment [28].

The policy is essentially a function that maps states to actions, dictating which action to take in each state.

The reward signal is a number given to the agent at each time step. This number is based on how well the selected action handles the current state. The agent's objective is to obtain as much total reward as possible. In essence, the policy keeps changing to get closer to this goal. In other words, policies are changed, so that they may lead to a selection of actions with better rewards. Accordingly, updates to the policy are dependent on the reward signal.

Meanwhile, the value function is used to predict how much reward will be received when starting from a specific state. That is to say, it focuses on the long-term value of a state and not just the immediate reward.

Finally, the model represents the behaviour of its corresponding environment, being useful for predicting future states and rewards. As such, it is generally used for planning the best actions to take, taking the predictions into account. Since there are both model-based and model-free RL methods, the model is considered an optional sub-element of RL.

2.1.1 Markov Decision Processes

When it comes to RL problems, it is possible to define them as MDPs, as long as they obey the Markov property. If the state of the system can represent all the relevant information about the sequence of states that led to the current one, this property holds [28].

We can then define RL problems using the tuple (S, A, P, R, γ), as explained by Yuxi Li [19]. Keep in mind that t represents the time step, $s' = S_{t+1}$, $s = S_t$ and $a = A_t$.

$\boldsymbol{S}$ is the set of states s that the environment can be in.

$\boldsymbol{A}$ is the set of actions a the agent can take in each state.

$\boldsymbol{P}(s'|s,a)$ is the transition model that defines the probability of transitioning from state s to state s', considering the action taken a.

$\boldsymbol{R}(s,a,s')$ is the reward function, dictating the reward r, which is received for taking action a in state s and reaching state s'.

$\boldsymbol{\gamma \in (0,1]}$ is the discount factor, which is used to make future rewards weight less.

As the goal of the agent is to try to maximize the total reward obtained, we need to consider the total sum of rewards received, G_t. The accumulation of future rewards leads us to a Total Reward

$$G_t = \sum_{k=0}^{\infty} R_{t+k+1}$$

If we discount future rewards using a discount factor γ, then we have a Discounted Reward

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

These two ways of representing the total sum of rewards are usually useful for different kinds of problems.

The Total Reward is ideal for episodic tasks, which are tasks that are divided into episodes. For situations in which the interaction between the agent and the environment can be divided into subsequences, which have a concrete terminal state, we can consider the sequence of events until the terminal state to be an episode. At the end of an episode, there is a reset to the starting state and a new episode begins. An example of an episodic task would be a game like Connect Four, where each game would be considered an episode, which would start from the starting position (no discs on the grid) until reaching a terminal state (when a player wins or when the grid is entirely filled up). In this case, each move is a subsequence of the corresponding episode.

Meanwhile, the Discounted Reward is typically used for continuing tasks, which are tasks that do not have a clear ending. This is done to avoid an infinite sum of rewards, as there is no concrete terminal state, unlike in episodic tasks.

To decide which action to do in each state, we can consider a policy $\pi(a|s)$, which defines the probability of taking action a in state s. The objective in an MDP is to get close to the optimal policy π_*, which is the one that maximizes the expected total reward.

Furthermore, we need to think about value functions. As mentioned previously, the notion of value is related with the expected rewards when starting from a certain state. By considering a policy π, we can then think about a state-value function $v_\pi(s)$, which defines the expected value for following policy π, starting from state s.

At the same time, it is also possible to think about the value of starting in state s, executing action a and then following policy π. This is defined as the action-value function $q_\pi(s, a)$.

The optimal policy π_* then corresponds to the optimal state-value and action-value functions, which are $v_*(s)$ and $q_*(s, a)$, respectively.

2.1.2 Dynamic Programming

DP is an RL approach used to find optimal policies. It is not considered practical in most situations, as it requires a perfect model of the environment to function. Furthermore, it is very computationally expensive, which means that problems with bigger state spaces are out of reach. However, a lot of the basis behind this methodology is used for more practical approaches, which makes it important from a theoretical standpoint.

For starters, we can think about policy evaluation. This involves figuring out which is the state-value function v_π that corresponds to the current policy π. We start with an arbitrary value function, which is then iteratively updated using an update rule. This is done by executing a full backup on each iteration, which corresponds to replacing the current value of each state s with the sum of the immediate reward and the current values of all possible following states. This process repeats until the difference between the new calculated values and the old ones is small enough, at which point we assume the policy evaluation has converged.

After obtaining the correct state-value function v_π, we need to consider policy improvement. Essentially, we want to find out if using a different policy would be better. This is where $q_\pi(s, a)$ comes into play. Recall that this action-value function represents the value of executing action a and then following policy π. If that value is greater than simply following policy π, then a policy that picks action a in state s and then acts the same way as the current policy would clearly be better. From there, we can think about a comparison across every state. If we compare the current value of each state with the values obtained with a greedy policy that always selects the action that maximizes $q_\pi(s, a)$, we will always get to a policy that is either equal or better than the previous one. If it is equal, then we have found the optimal policy.

The process of using policy evaluation to find the corresponding state-value function v_π and then using policy improvement to obtain a better policy repeats iteratively. This is referred to as policy iteration.

Since policy evaluation is a very long process itself, policy iteration will naturally be so as well. As such, it is important to consider ways to shorten policy evaluation. This can be done by stopping it after a single backup of each state, as opposed to repeating it multiple times. This simplified approach is called value iteration.

2.1.3 Monte Carlo Methods

MC methods learn from direct or simulated interaction with the environment, in order to estimate value functions and find optimal policies. They function similarly to DP methods, with a value function being changed to match the policy (policy evaluation) and a policy being updated to maximize returns according to the value function (policy improvement). However, they do not require complete knowledge of the environment, as everything is learned through the sample transitions experienced. While a model of the environment is still necessary for the simulated case, to be able to generate the samples, it is far easier to do so than to figure out the full probability distribution of the environment.

When considering episodic tasks, value function estimation and policy updates are done at the end of each episode. For each state-action pair occurring in an episode, the total return of rewards obtained from that state-action pair until the end of the episode is taken into account. From there, the value of each state-action pair is based on the average of returns received across all episodes up to that point. This way, we can construct values based on experience.

However, there is a problem that needs to be addressed. While following a policy, there are actions that might never be picked for a given state. If state-action pairs are being ignored, then it becomes impossible to estimate their value. In other words, something needs to be done to ensure that every state-action pair is estimated. This can be achieved with either on-policy or off-policy methods.

On-policy methods improve the same policy that is used to control the behaviour of the agent. To guarantee that every state-action is visited, a stochastic policy in which every state only has actions with nonzero probability of occurring is typically used. The policy is then gradually updated so that it becomes more deterministic and optimal. For example, it is possible to use an ϵ-greedy policy, which selects a random action with probability ϵ and an optimal action otherwise [28].

Off-policy methods make decisions using one policy, while trying to improve a different one. The policy controlling the behaviour of the agent is the behaviour policy, while the one whose value function we are trying to learn is the target policy. Since the behaviour policy is used to guarantee exploration, it should be a stochastic policy such as ϵ-greedy. On the other hand, the target policy can be fully greedy, always selecting the action most favoured by the action-value function $q_\pi(s, a)$.

2.1.4 Temporal-Difference Learning

TD methods share similarities with both DP and MC methods. On the one hand, they bootstrap estimates of following states to update the value of the current one, which is similar to what DP methods do. On the other hand, they learn from experience, without requiring full knowledge of the environment's dynamics, just like MC methods.

A key difference with MC methods lies in how long they wait to update their estimates. MC methods wait until the end of an episode and then compare the return obtained for each state with the corresponding estimate. Meanwhile, TD methods do so after reaching the next state and using the reward obtained R_{t+1} and the estimated value of S_{t+1} as the target. This leads us to the equation for TD(0), which is the most basic TD method:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)], \tag{2.1}$$

in which α is the learning rate.

There are several types of problems in which waiting for the termination of an episode slows down learning too much. Furthermore, empirical data has shown that TD methods tend to learn faster than MC methods.

An alternative way of taking advantage of TD learning is to use a batch of data. In this case, the value function is only updated after a certain number of time steps or episodes, corresponding to the batch. The updates consider the sum of all the increments calculated by the equation corresponding to the TD(0) update, which can be seen above.

Just like with MC learning, in TD learning we also have to consider the choice between on-policy and off-policy methods to deal with the exploration-exploitation dilemma. Examples of on-policy and off-policy methods include Sarsa and Q-Learning, respectively.

2.2 Quantum Computing

Quantum Computing is a computing paradigm that intentionally takes advantage of quantum properties [6]. By doing so, they have the potential to execute tasks that would otherwise be unfeasible to complete in a reasonable time frame.

While there are different kinds of Quantum Computing models, quantum circuits are the most common way to represent Quantum Computing. In the model of quantum circuits, we can think of the data, operations and results as quantum bits (qubits), quantum gates and measurements, respectively.

2.2.1 Qubits

Qubits are the basic unit used to represent quantum data. While a classical bit is represented by states 0 and 1, a qubit is similarly represented by states $|0\rangle$ and $|1\rangle$. However, while classical bits can be exclusively in state 0 or 1, qubits are considered to be in a combination of states $|0\rangle$ and $|1\rangle$, being characterized by $\alpha|0\rangle + \beta|1\rangle$, in which α and β are complex numbers and $|\alpha|^2 + |\beta|^2 = 1$. This means that each qubit is continuous, being able to take an infinite number of different values. This ability to be in a combination of states is called superposition. These qubits can be represented as column vectors $\psi = \binom{\alpha}{\beta}$.

Furthermore, we can use the Bloch Sphere, which has 3 coordinates (x, y and z), in order to map these 2D, complex vectors onto a real, 3D space [5]. In Figure 2.1, we can see the $|+\rangle$ state represented.

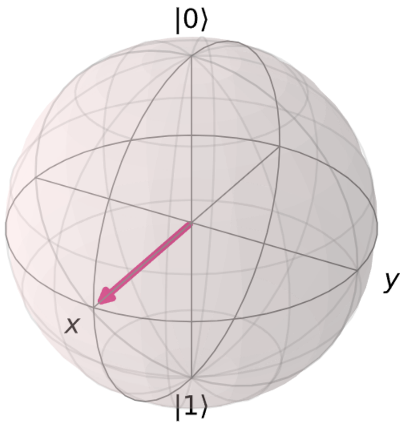


Figure 2.1: The Bloch Sphere. Adapted from [5].

We can then think of basis states for the poles in each axis of the Bloch Sphere. The most common ones are the basis states for the z-axis, $|0\rangle$ and $|1\rangle$. Besides these ones, the basis states for the x-axis, $|+\rangle$ and $|-\rangle$, are also very important, as they represent an even superposition of the previous two states (i.e. measuring them has a 50% probability of returning 0 and the same probability of returning 1).

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{2.2}$$

$$|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2.3}$$

$$|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \tag{2.4}$$

$$|-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \tag{2.5}$$

2.2.2 Single Qubit Gates

In order to apply operations on these qubits, quantum gates are used. These can be used to change the qubits or to create certain relationships between them, e.g. entanglement (which will be explained in Subsection 2.2.3). We can represent these quantum gates as unitary matrices, which can then multiply the vectors that represent the qubits.

There are multiple quantum gates [5], the most important of which will now be explained. We will be looking at the Pauli gates, which are the X, Y and Z gates, along with the identity I. The Hadamard and P gates will also be introduced, along with the S and T gates.

To start, we can look at the identity I, which simply has no effect when applied.

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{2.6}$$

From there, we have the X gate, which can take a qubit from state $|0\rangle$ to state $|1\rangle$ and vice-versa. More generally, it switches the amplitudes of states $|0\rangle$ and $|1\rangle$ [6]. This is similar to what a NOT gate does classically, so it is typically also recognized as the quantum NOT gate. As for the Bloch Sphere, the operation of this gate is depicted as a rotation by π radians around the x-axis.

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{2.7}$$

$$X\,|\psi\rangle \;=\; \beta\,|0\rangle + \;\alpha\,|1\rangle\,, with\,\psi = \alpha\,|0\rangle + \beta\,|1\rangle \tag{2.8}$$

From there, we also have the Z gate, which has no effect when applied to a $|0\rangle$ state, but changes the sign of a $|1\rangle$ state. Similarly to the X gate, the Z gate corresponds to a rotation by π radians around the z-axis of the Bloch sphere.

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2.9}$$

$$Z\,|0\rangle \;=\; |0\rangle \tag{2.10}$$

$$Z\,|1\rangle \;=\; -\,|1\rangle \tag{2.11}$$

Just like the other Pauli gates, the application of the Y gate corresponds to rotating by π radians around the y-axis of the Bloch sphere.

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \tag{2.12}$$

The H or Hadamard gate can take the basis states $|0\rangle$ and $|1\rangle$ into a superposition of these states, creating the $|+\rangle$ and $|-\rangle$ states, respectively.

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2.13}$$

$$H\,|0\rangle \;=\; |+\rangle \tag{2.14}$$

$$H\,|1\rangle \;=\; |-\rangle \tag{2.15}$$

There is also the P or phase gate, which is a parameterized gate, as it changes based on the parameter it receives. Essentially, it is a rotation of ϕ around the z-axis, with ϕ being a real number.

$$P(\phi) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix} \tag{2.16}$$

The S and T gates are merely special cases of the P gate. The S and T gates correspond to a ϕ of $\pi/2$ and $\pi/4$, respectively.

$$S = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{2}} \end{pmatrix} \tag{2.17}$$

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix} \tag{2.18}$$

2.2.3 Multiple Qubit Systems, the CNOT gate and Entanglement

When it comes to two-qubit systems, we can simply represent the state as:

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle = \begin{pmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{pmatrix} \tag{2.19}$$

With that in mind, we can now think of two-qubit gates as 4x4 matrices. The most prominent of these is the CNOT gate, which stands for Controlled NOT gate (also referred to as CX gate).

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.20}$$

The idea behind this gate is to look at the first qubit affected and decide what to do with the second one based on that. If the first qubit is $|0\rangle$, then nothing happens. If it is $|1\rangle$ however, the second qubit is flipped, just as would happen by applying an X gate directly to it.

$$\text{CNOT}|00\rangle = |00\rangle \quad \text{CNOT}|01\rangle = |01\rangle \quad \text{CNOT}|10\rangle = |11\rangle \quad \text{CNOT}|11\rangle = |10\rangle \tag{2.21}$$

The reason this gate is so important is because it can create entanglement. To understand why this is so important, we can first briefly explain what a product state is and how entangled qubits are different.

A state of multiple qubits is a product state if there is a tensor of states with less qubits equal to it [6]. In other words, a product state exists when $|\psi\rangle = |\psi_1\rangle|\psi_2\rangle$. When we have a product state, each qubit is still effectively independent of each other.

The CNOT gate allows us to reach entangled states, which cannot be represented as product states. This is due to the correlations between the first and second qubit, created by entangling the qubits. For example, if we have the following state, when we measure 0 on the first qubit, the result on the second qubit will always be 0 and vice-versa. Similarly, if we measure 1 on the first qubit, the second one will always be 1 and vice-versa. In other words, the only possible results are 00 and 11, with the first number representing the classical bit obtained by measuring the first qubit and the second one representing the classical bit obtained by measuring the second qubit.

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}} \tag{2.22}$$

This is one of the four Bell states, which all represent different entangled states. The other Bell States are the following:

$$\frac{|00\rangle - |11\rangle}{\sqrt{2}} \quad \frac{|01\rangle + |10\rangle}{\sqrt{2}} \quad \frac{|01\rangle - |10\rangle}{\sqrt{2}} \tag{2.23}$$

2.2.4 Measurements

In order to obtain the result of an operation, we must perform a measurement. Doing so implies collapsing the quantum state, which results in classical bits. Each resulting classical bit obtained can be in either the 0 or the 1 state, which are returned with probabilities equal to $|\alpha|^2$ and $|\beta|^2$, respectively. This means that measurements in quantum systems are probabilistic in nature, requiring multiple executions of the same circuit in order to obtain the expected value of each state.

2.2.5 Summary of Quantum Properties

We will now go over some key quantum properties, some of which were explained in more detail previously. This can serve as a summary of some basic quantum properties.

No-Cloning Theorem

The No-Cloning Theorem states that it is impossible to make an independent copy of a qubit. This is due to the fact that it is impossible to know the state of a qubit until it is measured, at which point it collapses the quantum state and becomes classical.

Superposition

A qubit is in a superposition characterized by $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, in which α and β are complex numbers. That is to say, a qubit is in a combination of states $|0\rangle$ and $|1\rangle$.

Entanglement

When two qubits are entangled, the result of measuring one of them is dependent on the result of measuring the other and vice-versa.

Interference

Interference is the manipulation of positive and negative amplitudes, in order to reinforce or weaken the probability of obtaining certain states. This property is used in order to reduce the probability of measuring the incorrect result, while amplifying the probability of reaching the desired outcome.

2.2.6 Grover's Algorithm

Grover's algorithm [17] is used for searching items in unsorted lists. The idea is to find an element that satisfies a certain condition, while making no assumptions about the structure of the elements. Classically, such a task would have a time complexity of $O(N)$ for N items, as each item would need to be evaluated individually until one of them met the condition. However, with Grover's algorithm, it is possible to reach a complexity of $O(\sqrt{N})$, which represents a quadratic speedup in comparison with the classical approach. This speedup is achieved by taking advantage of the superposition property, which makes it possible to analyse every element simultaneously.

Furthermore, this algorithm can serve as the basis for the amplitude amplification trick [4], in order to achieve quadratic run time speedups in other quantum algorithms. This further cements its importance as one of the key components for obtaining quantum advantage.

The process starts with the group of elements all having the same amplitude. In order to analyse if the elements respect the condition, it is necessary to use a Boolean function $F(x)$, which returns 1 when they do and 0 when they do not. Using the oracle $-1^{F(x)}$, the elements are evaluated and the ones respecting the condition rotate by a phase of π radians (phase kickback), which corresponds to negating their amplitudes. From there, a unitary transformation is applied, in order to calculate the average of the amplitudes. Then, the amplitudes of the elements suffer an inversion around the mean, which slightly lowers the amplitudes of all the elements that do not obey the condition, while sharply increasing the amplitudes of the ones that do. This inversion around the mean is referred to as Grover's Diffusion Operator and can be observed in Figure 2.2. The entire process repeats multiple times, in order to keep magnifying the amplitude of the elements that respect the condition, so they can be measured with higher probability. The number of repetitions is proportional to $\sqrt{N}$.

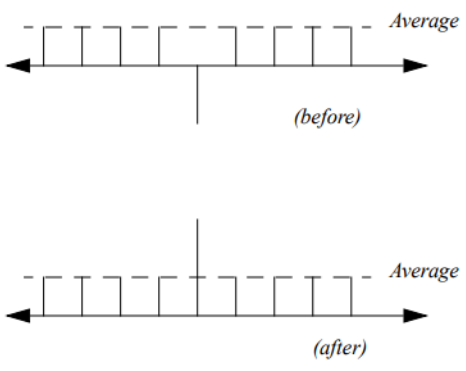


Figure 2.2: Inversion around the mean in Grover's algorithm. Adapted from [17].

2.2.7 Quantum Walks

Quantum walks are quantum processes that can be exploited by multiple algorithms. In order to properly introduce them, we must first define classical random walks. Firstly, we can think about a random walk on a line, which is a stochastic process in which a particle moves along a line with probability p of going to the right and probability $1-p$ of going to the left [31]. This particle is considered the walker, while the probability distribution is the coin, which controls the direction to move to. If we replace the line with a graph, it is possible to further generalize this process to higher dimensions, at which point it can be thought of as a Markov Chain.

As for quantum walks, they can be defined as quantum processes which move the walker in a direction, due to the influence of unitary transformations [7]. The most basic quantum walk is the Discrete Quantum Walk on a Line (DQWL). For the previous definition to hold for a DQWL, it is necessary to consider two quantum systems, one for the walker and one for the coin [31]. As for the unitary transformations, there is a coin operator that decides which direction to take and a shift operator that moves the walker in the corresponding direction. Just as in the classical case, this definition can be extended to graphs, in order to consider higher dimensional spaces. Furthermore, it is possible to consider the case in which time is continuous, which leads to continuous quantum walks.

While there are different unitary transformations that can be used for the coin operator, this is typically done using either the Hadamard transformation or Grover's Diffusion Operator.

Search via Quantum Walk

One of the key applications of quantum walks is searching for elements in a set. If we consider a set of N elements, there are M elements within the set that meet a certain condition. By taking advantage of quantum walks, it is possible to achieve a quadratic speedup for the number of iterations necessary to find a marked element, in comparison with a classical approach. Mario Szegedy [29] found this speedup to be possible only under certain conditions, which involve the corresponding Markov Chain being ergodic and having a symmetric transition matrix. This speedup for searching marked elements may sound very similar to Grover's algorithm. This is due to the fact that Grover's algorithm can be considered a quantum walk on the edges of a graph.

Frédéric Magniez et al. [21] improved upon Mario Szegedy's work by using quantum phase estimation, with the aim of approximating the reflection present in Grover's Diffusion Operator. They also generalized searching via quantum walks to a less restrictive group of Markov Chains.

By considering p_{xy} as the probability of reaching state y from state x, we can define the starting state of the searching algorithm as

$$|\pi\rangle = \sum_{x \in X} \sqrt{\pi_x} \, |x\rangle \, |p_x\rangle \tag{2.24}$$

in which π is the stationary distribution and $|p_x\rangle = \sum_{y \in X} \sqrt{p_{xy}} \, |y\rangle$. The key idea here is to transform this initial state into a normalized projection onto the group of marked states

M, which is defined as

$$|\mu\rangle = \frac{1}{\sqrt{p_M}} \sum_{x \in M} \sqrt{\pi_x} \, |x\rangle \, |p_x\rangle \tag{2.25}$$

in which $p_M = \sum_{x \in M} \pi_x$ is the sum of the probability of the marked states. In order to do so, a process similar to Grover's algorithm is applied T times, with T being selected uniformly at random in $[0, 1/\sqrt{\epsilon}]$ and ϵ being a lower bound on the probability of an element being marked.

2.3 Connect Four

Connect Four is a turn-based game in which players drop discs in a grid. The grid is composed of 7 columns and 6 rows. As for the discs, they are available in 2 different colors, which are used to distinguish the players.

Each turn, a player is forced to pick a column that has not been filled yet and inserts one of their discs in it. This is followed by their opponent, who must do the same. Whenever a player drops a disc, it falls on top of the highest one of that column (unless the column was empty). This process repeats until either a winner is found or the grid is completely filled up.

A player wins if 4 discs of the color they picked are positioned next to each other in sequence. This can happen horizontally, vertically or diagonally. If the grid becomes full and this condition is never verified, then the game is considered a draw.

This page is intentionally left blank.

Chapter 3

Literature Review

This chapter starts by covering related work in the field of classical Reinforcement Learning (RL). This is relevant for two reasons. Firstly, since a classical RL approach was developed to be used as a means of comparison with the quantum version. Secondly, as understanding the current capabilities of classical RL is useful for putting the current quantum context in perspective. We then look at the quantum literature and explore different topics, with a good portion of them being related with applications within the domain of games. Lastly, different quantum frameworks are presented and evaluated, so as to establish their capabilities and to be able to provide a proper explanation of the framework chosen for this work.

3.1 Classical Deep Reinforcement Learning

Yuxi Li [19] gives an introductory overview on Deep RL, covering many essential topics. For instance, the difference between deep and "shallow" learning is explained, with the former relying on hidden layers that receive the input of the previous layers. There is also a backpropagation of gradients, that leads to the optimization of the weights on the network. The book also goes over many different applications of Deep RL, such as games, Natural Language Processing (NLP) and robotics.

3.1.1 Atari Games

Volodymyr Mnih et al. [23] used a Convolutional Neural Network (CNN), in order to learn from raw video data. The objective here is to estimate future rewards, so the agent can better understand the Atari games it has to play. They made use of Experience Replay, in which experiences can be replayed from a buffer, so that local minimums that are too far from the desired can be avoided. They managed to obtain good results, outperforming previous approaches for 6/7 games and even top players in roughly half of the games. While they used uniform sampling for their work, they mention that there is potential to use more advanced strategies, which could yield better results.

3.1.2 The Game of Go

David Silver et al. [26] propose a more focused approach to Alpha Go, which they refer to as Alpha Go Zero. While the original Alpha Go was first trained by analysing moves made by professional players and only afterwards practiced against itself using RL, this new approach uses only RL. This modification was done to combat some of the known issues with supervised learning, such as (i) the fact that access to a good quantity of quality data can be difficult, and (ii) more importantly, the skill ceiling that can be reached when using human data, since the performance might not reach the levels it otherwise could. While Alpha Go was already capable of defeating the best human players, it was found that Alpha Go Zero could go further beyond, winning every single time in a head-to-head matchup.

3.2 Quantum Neural Networks

S. Mangini et al. [22] present a summary of recent findings and the landscape of the field. They go over the multiple models, such as:

1. Quantum Neural Networks, which are usually defined as Parameterized Quantum Circuits (PQCs) that attempt to emulate classical Neural Networks (NNs). Due to their relatively low requirements in terms of resources, they have the potential to be useful in near-term noisy quantum computers.
2. Quantum Generative Neural Networks, which are a quantum version of its classical counterpart. One way to implement them is through Quantum Generative Adversarial Networks, which are composed of 2 quantum NNs: a generator that learns how to create accurate samples and a discriminator that attempts to distinguish real examples from the generated samples.
3. Quantum Convolutional Neural Networks, which consist of repeated convolutional and pooling layers. Convolutional layers are parameterized unitary operations that apply local transformations to their input, in order to extract information. Pooling layers consist in measuring selected portions of qubits, in order to achieve dimensionality reduction (through the use of mathematical operations, e.g. sum).
4. Quantum Dissipative Neural Networks, in which multiple layers are connected. Once a layer is no longer needed, it is discarded and the process moves on to the following one.

However, they also mention possible difficulties in translating classical approaches to the quantum context, specifically the non-linear functions that are necessary in Deep Learning (as opposed to the linear nature of Quantum Computing) and the inability to copy.

Christa Zoufal et al. [34] tackle the problem of loading classical data into the quantum context. They explore the option of loading an approximation of the state, as opposed to attempting to represent it in its entirety. It is demonstrated that this new approach requires O(poly(n)) gates, instead of the usual O(2^n). As such, this approximate loading can be useful for algorithms that are resistant to small errors in the input.

3.3 Quantum Reinforcement Learning

3.3.1 Quantum-inspired Experience Replay

Qing Wei et al. [32] proposed a new method for training NNs, using a quantum-inspired version of a replay buffer, in which experiences can be stored and later replayed for training. Their strategy involved using quantum representations of the experiences and then applying what they referred to as preparation and depreciation operations. The preparation operation prioritizes the experiences in terms of importance (complexity), while the depreciation operation takes into account the number of replays of each experience, to guarantee diversity. Their experiments involved different types of Atari games from the OpenAI Gym platform [10]. The authors found that their approach achieved either better or roughly equal training efficiency when compared to classical state-of-the-art algorithms, namely Deep Reinforcement Learning - Prioritized Experience Replay (DRL-PER) and Deep Curriculum Reinforcement Learning (DCRL). It is important to notice that this new method is a quantum inspired algorithm and not a purely quantum one, as it can be simulated on a classical computer.

3.3.2 Variational Quantum Circuits

Chen et al. [11] used Variational Quantum Circuits (VQCs) for a Q-Learning approach, in order to approximate the optimal Q-value function. Part of their approach involved translating classical techniques that solve issues in Deep Q-Learning into a quantum context. Those techniques were experience replay, which is useful for lowering correlations, and the use of a target network, to avoid abrupt changes in the Q-value function. They utilized this RL approach to solve the Frozen Lake and the Cognitive Radio problems. These two were chosen since they are standard OpenAI Gym environments of low complexity, being suited for the current era of Noisy Intermediate-Scale Quantum (NISQ). The second one involves choosing an unoccupied radio channel, showing potential practical applications of Quantum RL for the future. As for the quantum state representation of these environments, computational basis encoding was used. Their tests involved 3 different types of experiments. They first tested standard simulations on classical hardware, taking advantage of the python library PennyLane. Secondly, they repeated the same tests while simulating noise, by using a backend that allows the use of noise models, Qiskit-Aer. Finally, they used the IBM Quantum platform to test the agents trained in the first experiment on real quantum hardware. The results from this final experiment showed that even agents trained on noiseless simulations could reach the desired goals when run on quantum computers.

Andrea Skolik et al. [27] analysed training methods for a Deep Q-learning algorithm that uses a PQC as the Q-function approximator. Unlike NNs, which are the typical approximator, PQCs have a fixed range of output values, which depend on its measurements. That fixed range is a limitation, since the Q-values (expected rewards) that need to be learned can have varying values, depending on both the environment and the agent's performance. In order to get around this issue, the authors suggest adding trainable weights to the outputs. They also evaluated the importance of data encoding when it comes to increasing the expressivity of the model, exploring two methods: data re-uploading and the addition of trainable weights for the inputs. In data re-uploading, the encoding is done at the beginning of each layer, instead of solely on the first one. They used Cirq and TensorFlow Quantum in order to study the impact of these techniques on two Atari benchmark environments, Frozen Lake (discrete) and Cart Pole (continuous). For Cart Pole, their results validated their assumptions regarding the importance of the encoding and readout strategies used. They compared it with classical NNs in terms of episodes needed to win the game and found that the classical model could not complete that task with the same number of parameters, needing triple of that to achieve the same performance.

Jen-Yueh Hsiao et al. [18] tackled the problem of sample inefficiency that is typically present in classical RL. For that purpose, they propose a Single-qubit-based Variational Quantum Circuit (SVQC), which exclusively employs single qubit gates, instead of relying on operations that take advantage of entanglement. The outputs of the quantum circuit are then used as input for a classical NN, which is followed by a softmax function that calculates the probabilities of selecting each action. They evaluated their method in multiple OpenAI Gym benchmark environments, including Cart Pole, Acrobot and LunarLander. The results obtained showed that it is not only possible to improve the speed of convergence relative to classical NN approaches, but that the number of trainable parameters necessary to achieve the same performance is lower. This work is also the first time the LunarLander environment has been successfully completed by a Quantum RL algorithm. Furthermore, additional tests were done to ascertain the usefulness of this strategy in current NISQ devices. These tests used agents that had been trained on quantum simulators and compared their performance on the real quantum devices with the one they had on the simulators, which revealed fairly similar results.

3.3.3 Atari Games

Owen Lockwood et al. [20] propose encoding data into a quantum circuit using a classical NN, as opposed to doing it in a static manner. This novel approach tries to overcome previous issues in dealing with large inputs, specifically when it comes to Quantum RL with Atari games, whose inputs consist of the pixels on the screen. Encoding data of this sort would typically require either a massive number of qubits or gates, depending on the specific method. For their experiments, they used a simulator on the TensorFlow Quantum environment, in order to compare their hybrid quantum-classical methods with classical ones. Their results showed that the hybrid agents did not learn the environments, as opposed to the classical approaches, with them concluding that new advancements in Quantum RL might be necessary to get the desired results in more complex environments. However, they also mention that there are some key strategies not utilized in their work, such as data re-uploading, which could be a key factor for the results obtained.

3.3.4 Projective Simulation

Hans J. Briegel et al [9] established an RL framework that projects an agent into simulated situations, before it chooses the action to take. Projective Simulation is centered around clips, which represent episodes stored in memory. By considering a stochastic clip network, it is then possible to think about a clip being excited by a perception of the environment, which leads to the corresponding experience being replayed. That clip can then jump to a neighboring clip that contains similar experiences and so forth, through the use of classical or quantum random walks, until an actuator clip produces an action. This allows the agent to simulate future situations before making a choice, by projecting itself into those clips. There is also the possibility of creating fictitious clips, which can let the agent plan using experiences that never actually happened, before committing to a decision. Furthermore, tags were employed so that transitions between clips could be classified based on how they were rewarded. This way, actions that had previously been rewarded for a certain state have a higher chance of being picked again, as not only is the related transition probability higher, but the corresponding tag can confirm that the simulated action is a good choice. They applied their framework to the invasion game, so as to demonstrate its capabilities in a simple environment. The results obtained showed comparable learning speed to similar strategies, such as experience replay and Dyna-style planning.

Giuseppe Paparo et al. [24] expanded upon this initial concept by further exploring a specific variant of this model, known as Reflecting Projective Simulation. This name derives from the fact that the agent reflects upon its choices when it has to select an action. When a percept clip is excited, by using a re-normalized probability distribution with support only over its flagged actions, we then apply a quantum walk and check if a tagged action was obtained. The quantum walk over the clip network mimics the search for marked elements seen in [21], with the marked elements being the flagged actions. If the action was tagged, then it is selected, otherwise the process repeats up to a predefined number of tries, which corresponds to the agent reflecting on that choice. Their work proved that it is possible to achieve a quadratic speedup with the quantum version of the framework, while still respecting that the actions need to be chosen according to that specific re-normalized probability distribution.

3.3.5 Checkers

Miguel Teixeira [30] applied Quantum RL to the game of Checkers, by using a quantum-enhanced agent. In order to avoid dealing with a large state space composed of every possible board position, they instead represented the problem using the relative position of the pieces on the board and their number. This way, similar board states were interpreted similarly by the agent. As for the agent itself, a quantum flag update mechanism inspired by [9] was used to facilitate the identification of positive actions, by amplifying the probability of selecting flagged actions. This mechanism is based on searching via quantum walks [21]. Their experiments involved a comparison with a classical agent, which used Negamax to pick its moves. The agent's learning efficiency was found to be better than the aforementioned classical approach.

3.4 Quantum Frameworks Analysed

In this section, we will be covering multiple quantum frameworks, explaining how they can be used and why they are useful. This is not meant as a purely comparative perspective, but more so as a showcase of their potential usefulness, as some of these tools are compatible with each other in multiple ways.

IBM-Quantum Experience [1] is an established platform on the field of Quantum Computing, that provides entry to anyone that creates an account. It has an extensive and complete documentation, facilitating the creation of complex programs.

There are three main ways to take advantage of this environment:

1. Firstly, it is possible to use a simple drag and drop feature, which allows the user to position gates on a circuit and view the corresponding representations of the system (such as the Statevector and the Q-Sphere).
2. Secondly, there is a very basic OpenQASM language, which can be used to create very simple programs.
3. Finally, there is the possibility of working with Qiskit, which is a very thoroughly documented and more complete Python based framework. This third option is what is used for developing in real scenarios. It is especially convenient since it is integrated with Jupyter Notebook, a widely used Python editor.

After creating a program, a user can select whether they want it to be ran on a simulator or on a real quantum computer. In case of the latter, it is possible to check the characteristics of the quantum hardware selected beforehand (such as the connections between qubits and the errors associated), in order to make a more informed decision. As these quantum systems are shared between all users, it is necessary to queue the task to be executed and wait for it finish. It should also be pointed out that IBM's quantum simulators allow the specification of noise models, which can be useful for simulating quantum programs with a more clear view of how they would perform in a real situation.

However, it is worth noting that most of the quantum hardware, especially the most powerful, is only available for specific users.

Cirq [2] is Google's open source python library that can be used to develop quantum programs. It is possible to use Cirq by installing and importing it, which can be done either locally or on Google Colab. Google provides thorough documentation and multiple tutorials for this framework.

Currently, the average user can only run programs created with this tool on quantum simulators. Access to Google's quantum hardware is restricted to users with approved projects. It is also possible to integrate Cirq with other kinds of quantum hardware, specifically AQT hardware, Azure Quantum, IonQ hardware, Pasqal hardware and Rigetti hardware. However, these are either restricted to partners, are only in public preview or have some other type of limitation restricting public access.

There are multiple libraries and extensions that can be used in tandem with Cirq, such as OpenFermion and TensorFlow.

Microsoft's Quantum Development Kit [3] has recently started its public preview. This development kit is based around the programming language Q#, which is a high-level language that focuses on allowing users to create more complex quantum programs, without having to worry about circuit details as much [15]. The idea behind developing this framework was to create something not only suited for running quantum programs on current quantum hardware, but also for future large-scale quantum computers. This scalability is achieved through abstractions of standard quantum algorithms and access to a multitude of general and domain-specific libraries.

While any user can simulate their quantum programs in their own classical setup, running Q# programs on quantum hardware requires an Azure subscription, which has several limitations for users with free accounts.

It is also worth noting that the development kit is compatible with Cirq and Qiskit, allowing users to submit circuits developed with those frameworks.

PennyLane [8] is a python library that aids users in creating quantum Machine Learning (ML) algorithms. It can be particularly useful for hybrid classical-quantum optimization architectures, as it provides an interface with standard ML libraries.

It is cross-platform in multiple ways. For instance, it can use quantum hardware from different sources as its backend, such as IBM-Quantum or IonQ hardware. On top of that, it can be integrated with other quantum frameworks, including Microsoft's Quantum Development Kit and Google's Cirq. Furthermore, it can be combined with other Python libraries, e.g. TensorFlow and PyTorch, which are both ML frameworks. It is also possible to run programs using PennyLane's own simulators.

Quirk [16] is a web-based quantum circuit simulator, which allows users to drag and drop gates on a circuit in an interactive way. This platform can be accessed by anyone and does not require an account. It is designed to be used to test very simple circuits, as it only allows users to use a maximum of 16 qubits.

With this program, users can experiment with different circuit setups, to confirm that they make sense. It is also possible to view more information about each state related with a circuit, such as their amplitudes, which can aid users in understanding them at a deeper level.

Essentially, this is a simple visualization tool, which together with other frameworks can help users design the circuits to implement or to better understand the existing circuits.

3.4.1 Framework Used in this Work

IBM-Quantum Experience was chosen as the framework for this thesis. This decision came about for multiple reasons. Firstly, as the approach presented is based on the Quantum RL method developed for Checkers [30], it makes sense to employ the same framework, so as to be able to reuse some of the code related with the creation of quantum circuits. Secondly, this is a framework that the author of this work is quite familiar with and will be able to take full advantage of. Lastly, the completeness of its documentation makes it easier to solve any potential issues. It is worth noting that the experiments were run on a quantum simulator, as opposed to real quantum hardware, as training RL agents would take far too long otherwise.

This page is intentionally left blank.

Chapter 4

Methods

In this chapter, the methods are presented. We start by covering the opponent, which the agents have to play against. We discuss how we ultimately got to the Randomized Negamax approach, explain its usefulness and then proceed to describe it in detail. We then define Connect Four as a Markov Decision Process (MDP) and explain how Q-Learning was applied to this board game. From there, we tackle the exploration-exploitation dilemma within the context of this problem, first going over the approaches applied for the classical agents. For that purpose, we define a standard ϵ-greedy approach and then describe the more complex action selection with tags utilized. Additionally, the quantum exploration policy is covered, which is the quantum version of the action selection with tags. We show how it differs from the classical version, the advantages it has, go into detail about its quantum reflection process and how the action selection probabilities were encoded.

4.1 The Opponent - Randomized Negamax

We will start by establishing an opponent for the Reinforcement Learning (RL) agents to play against. This way, it will be possible to have a fair comparison between the classical agents and the quantum agents. While comparing them directly might seem preferable at first, doing so would complicate matters, as going first in Connect Four is a very significant advantage. This way, we can indirectly compare classical agents that act as player 1 with quantum agents that act as the same player. The same logic applies for player 2.

We first considered using Monte Carlo Tree Search (MCTS) agents for this role, as it was the opponent employed in the work used as a basis for the Q-Learning approach developed in this dissertation [33]. However, this choice revealed problematic. As the MCTS agents need to be trained, a significant amount of time is spent training agents that merely exist to serve as a point of reference. It is further necessary to decide on the number of episodes these agents need to be trained for, in order to serve as a competent enough adversary. This is a problem, since it means that in order to create opponents of different strengths, it is imperative to train multiple agents for a different number of episodes and assess their performance. Furthermore, it is essential to train agents with the same number of episodes, in order to ensure that no outliers are selected as actual opponents. The same logic applies to the parameters of the MCTS, which by themselves lead to the training of more agents, so as to validate those parameters. On top of that, even if the MCTS agents are trained for extended periods of time, they can still be vulnerable to very simple strategies, such as stacking a specific column of the Connect Four board. As a result, this type of opponent was considered inadequate.

We then thought about using an opponent that plays randomly for most of the game, but has a built-in mechanism that looks ahead two moves and forces it to choose winning moves and never pick ones that lose on the spot, unless all of the moves available are losing. The problem with this idea is that, although we can ensure that the agents are being evaluated on their ability to close out a game, the lead-up to that consists of merely fighting a random opponent, which does not make for a very interesting challenge of the agents' abilities.

From there, we contemplated adopting a Minimax [25] strategy for the opponent. This algorithm can be applied when we have a zero-sum game [25], which implies that the gain for one player is the equivalent loss for the other player, hence the sum of the scores of both players is always zero. The idea behind the algorithm is to recursively look ahead a certain number of moves and evaluate all the possible actions in each move selection through an evaluation function, in order to pick the best one. In each step corresponding to its opponent, the action that maximizes that player's score is selected, which is in turn the one that minimizes the score of the Minimax player [28]. Essentially, this leads to choosing the best move according to an evaluation function, while assuming that both players are selecting the actions that are considered optimal.

The specific variant of Minimax we chose was Negamax [1], which alters the formulation of the problem, so that in each step one player considers the negated score from the other player. This is equivalent to the standard Minimax formulation, but avoids having to use two slightly different functions for each player. This is defined by the following equation, in which depth is the number of moves we are looking ahead.

$$score_A = -Negamax(depth - 1)_B \tag{4.1}$$

Finally, we put some of the previous ideas together, in order to form a Randomized Negamax algorithm. The idea here is to have a Negamax algorithm, but after returning from all the recursive calls back to the final move selection, there is a random chance ω of selecting a move at random, as opposed to the optimal one. This is done in order to present a more complicated challenge for the agents, as they will have to deal with some randomness in the move choices from the Negamax opponent. This random selection prioritizes the moves with positive scores, so as to guarantee a strong performance under that randomness. Furthermore, we incorporated the previously mentioned strategy of always selecting moves that directly win the game and never selecting ones that directly lose the game, if possible. This way, we ensure that the agents cannot win in overly trivial ways, such as stacking a fourth disc when the Negamax player has no reason not to fill that spot first. This algorithm can be visualized in Algorithm 1.

The Randomized Negamax algorithm makes use of Alpha-Beta Pruning [25] in order to more efficiently explore the game tree. This technique relies on parameters α and β, which are lower and upper bounds on the evaluations, respectively. When a score bigger than the upper bound is found for player A, the analysis of the corresponding tree branch can stop, as player B would pick a move that denies the opportunity of surpassing that upper bound. Meanwhile, when player B finds a score smaller than the lower bound, the analysis can also stop, as player A would choose a move that guarantees that lower bound. This way, it is possible to reduce the amount of time spent evaluating actions from the game tree.

The *heuristic* function called in the Randomized Negamax algorithm corresponds to an

[1]https://www.chessprogramming.org/Negamax (Accessed: 3 July 2022)

Algorithm 1 Randomized Negamax

```
1: procedure RandomizedNegamax($state, depth, \alpha, \beta, tag$)
2:     if $depth = 0$ or gameDone($state$) then
3:         return $move \leftarrow -1$, $eval \leftarrow$ heuristic($state, tag$)
4:     $moves \leftarrow$ availableMoves($state$)
5:     $winning\_moves \leftarrow$ list(), $losing\_moves \leftarrow$ list(), $evaluations \leftarrow$ dictionary()
6:     $max\_eval \leftarrow -\infty$, $min\_eval \leftarrow \infty$
7:     $tag\_b \leftarrow$ changeTag($tag$)
8:     for $move$ in $moves$ do
9:         $state\_b \leftarrow$ makeStateFromMove($state, move, tag$)
10:        $evaluation \leftarrow -$randomizedNegamax($state\_b, depth - 1, -\beta, -\alpha, tag\_b$)[eval]
11:        $evaluations$.update($move \leftarrow evaluation$)
12:        if $evaluation > max\_eval$ then                        ▷ Update Max Evaluation
13:            $max\_eval \leftarrow evaluation$
14:            $best\_move \leftarrow move$
15:        if $evaluation < min\_eval$ then                        ▷ Update Min Evaluation
16:            $min\_eval \leftarrow evaluation$
17:        if $evaluation > 0$ then                                ▷ Append Winning Move
18:            $winning\_moves$.append($move$)
19:        else                                                    ▷ Append Losing Move
20:            $losing\_moves$.append($move$)
21:        $\alpha \leftarrow$ max($\alpha, max\_eval$)                  ▷ Alpha-Beta Pruning
22:        if $\alpha >= \beta$ then
23:            break
24:    if length($winning\_moves$) $> 0$ then              ▷ Choose from the winning moves
25:        $move \leftarrow$ randomChoice($winning\_moves$)
26:    else                                                ▷ Choose from the losing moves
27:        $move \leftarrow$ randomChoice($losing\_moves$)
28:    if $depth = starting\_depth$ and $max\_eval < 10000$ and $min\_eval > -10000$ then
29:        $p \leftarrow$ random number in [0, 1]
30:        if $p < \omega$ then
31:            return $move = move, eval = evaluations[move]$
32:    return $move \leftarrow best\_move, eval \leftarrow max\_eval$
```

evaluation of the current state and can be visualized in Algorithm 2. This evaluation starts by checking if the game is over, so that it can employ a protection against choosing randomly when that specific move choice could end the game. If the game would end in a draw or the player currently being evaluated would be the winner of the game, then a score of 10000 is received. On the other hand, if the other player would win, then -10000 is returned. When the Randomized Negamax function gets to the final move choice, it will compare the highest and lowest evaluations with 10000 and -10000 respectively, so that the final move decision is never random if the sequence of moves that follow it can end the game. This way, the Negamax player does not lose in overly simple ways. Ten thousand was chosen as it is a bigger number than the highest possible heuristic obtained normally, while still being smaller than ∞. The same logic applies for the negative case.

Algorithm 2 Heuristic

```
1: procedure HEURISTIC($state, tag$)
2:     $winner \leftarrow$ gameWinner($state$)
3:     if $winner = tag$ or $winner = 0$ then                              ▷ Win or Draw
4:         return 10000
5:     else if $winner \neq$ None then
6:         return $-10000$                                                ▷ Opponent Wins
7:     $max\_heuristic \leftarrow 0$
8:     $min\_heuristic \leftarrow 0$
9:     for $i \leftarrow [0, length(state[:, 0]) - 3[$ do                   ▷ Rows
10:        for $j \leftarrow [0, length(state[0, :]) - 3[$ do                ▷ Columns
11:            $new\_max, new\_min \leftarrow$ connect4($state[i : i + 4, j : j + 4]$)
12:            if $new\_max > 1$ then
13:                $max\_heuristic \leftarrow max\_heuristic + new\_max \times 2$
14:            if $new\_min < -1$ then
15:                $min\_heuristic \leftarrow min\_heuristic + new\_min \times 2$
16:    if $tag = 1$ then
17:        return $max\_heuristic + min\_heuristic$
18:    else
19:        return $-(max\_heuristic + min\_heuristic)$
```

If the game has not ended however, then the evaluation is calculated using an heuristic, with the *state* in the algorithm being the Connect Four grid itself, with 7 columns and 6 rows. The idea is to consider every 4 by 4 square inside the Connect Four grid, with the tags for the players 1 and 2 being identified with 1 and -1, respectively. By summing along the column, row and both diagonals of each of those squares, we obtain four numbers, which represent the advantage a player has in each of those four lines. Positive values indicate that player 1 has the advantage, while negative values signify the opposite. The *connect*4 function simply returns the highest and lowest of those four values to the *heuristic* algorithm. This iteration through every 4 by 4 square is done in the *for* loops of the *heuristic* function. From there, if the highest value returned is higher than 1, then we sum the double of its value to the *max_heuristic* variable, with a similar strategy being applied for the lowest value. The reason for only considering values higher than 1 and lower than -1 is to only benefit players that are actually connecting multiple discs, while penalizing lines that have discs from the other player in the middle. The values are multiplied by 2 to create a bigger gap between, for instance, connecting 2 and 3 discs. The final value returned is the sum of the highest heuristic, which is always positive, with the lowest heuristic, which is always negative. This value is negated if the player being evaluated is player 2, as that player's discs have the value -1 instead of 1.

4.2 Connect Four as a Markov Decision Process

We can define the RL problem of Connect Four as an MDP, as it obeys the property of the future only depending on the current state and action selected (and not on the past) [19].

State Space *S*: The state space is composed of every possible combination of empty spaces and discs from either player. Since the board has 7 columns and 6 rows, that means we have $7 \times 6 = 42$ spaces on the grid. As such, we have at most 42^3 possible states (since each space can be empty, have a piece from player A or have a piece from player B). By ignoring positions that are impossible to reach, such as having discs in the middle with nothing below them, we can get a lower state-space complexity of 4.53×10^{12} [14]. As mentioned in [33], we can consider that half of these states correspond to one player and the other half to another, as the player going first will always see a board filled with an even number of pieces before selecting a column to play, while the opposite is true for the other player. In the context of this thesis, the state space is represented through a matrix with 7 columns and 6 rows, which corresponds to the Connect Four grid. In this matrix, 0 represents an empty space, 1 represents a disc from player 1 and -1 represents a disc from player 2.

Action Space *A*: The action space is defined by the columns a player can insert a disc in. Since there are 7 columns, this space corresponds to 7 minus the number of full columns in which discs cannot be inserted.

Reward Function *R*: The reward function used was adapted from [33] and is the following:

$$R(s,a,s') = \begin{cases} 1 & \text{if the agent wins the game} \\ 0.5 & \text{if the game is a draw} \\ -1 & \text{if the agent loses the game} \\ 0 & \text{if the winner is yet to be decided} \end{cases} \tag{4.2}$$

Discount Factor γ: A discount factor of 1 was chosen for this problem, which means the Total Reward is being considered.

4.3 A Connect Four Q-Learning Approach

The RL method used in this work was Q-Learning, which is a model-free, Temporal-Difference (TD) learning approach, as mentioned in Chapter 2. The idea is to try to learn the optimal action-value function Q directly, with $Q(S_t, A_t)$ representing the expected future reward for the state-action pair [27, 28]. Q-Learning can be defined by the following equation:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \underset{a}{max} Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{4.3}$$

in which α is the learning rate, R_{t+1} is the reward for selecting action A_t in state S_t, γ is the previously mentioned discount factor and $\underset{a}{max} Q(S_{t+1}, a)$ is the highest action value Q out of the possible transitions for the following state. It is also worth mentioning that

$R_{t+1} + \gamma \underset{a}{max} Q(S_{t+1}, a)$ is considered the target value, while $R_{t+1} + \gamma \underset{a}{max} Q(S_{t+1}, a) - Q(S_t, A_t)$ is considered the TD error.

We can think of an optimal action as selecting the action with the highest Q value for a state.

In [33], a classical Deep Q-Learning approach was used to train RL agents that can learn how to play Connect Four. Those agents were trained by playing against random agents and then evaluated by playing against MCTS. The results obtained showed that the trained Q-Learning agents had a similar win rate to the MCTS agents.

The code present in their work was used as a starting point for the Q-Learning approach used in this thesis. It initially allowed for the creation of classical agents, which themselves served as the basis for the quantum agents, as well as a means of comparison.

Since Connect Four is not as complex as Chess or Atari games, it would be possible to solve the game in a classical computer using a brute-force approach, taking advantage of search algorithms [12]. As such, the idea behind the Q-Learning approaches proposed in this dissertation is not to find an algorithm that surpasses those brute-force methods, but rather to create RL agents that can play well against relatively powerful classical approaches.

4.3.1 Relevant Choices for this Approach

The Q-Learning method used as reference was changed in order to fulfill multiple purposes. For instance, in the original work, the ϵ used for the ϵ-greedy exploration policy was changed manually. For this dissertation however, the corresponding ϵ-greedy approach uses a varying ϵ, which decreases gradually with the number of episodes. Furthermore, different exploration policies were used. These ideas will be thoroughly explored throughout this chapter.

Furthermore, offline learning was used to train the Q-Learning agents. This means that the agents played a certain number of games, equal to the size of the batch (which was fixed at 300 episodes) and were then trained on that batch before moving on to the next batch of games. Each batch ran for 5 epochs, which means that the Neural Network (NN) spent 5 cycles with the same batch. This is the same way that the agents were trained in [33].

Two networks were tested for the Deep Learning of the Q-function, with them being used to learn the Q values at the end of every batch, allowing Equation 4.3 to be solved for each transition stored in the batch. For each state-action pair, the network predicts its value by considering the Connect Four matrix, which represents the state, appended to a vector with seven numbers, a 1 in place of the column selected and zeroes everywhere else, which represents the action taken. Network A was the one used in the work that contains the code adapted for this project [33]. Meanwhile, Network B is the same Network B used in [12]. Preliminary results were inconclusive regarding the performance of the two networks. As such, Network A was ultimately chosen, as it was the one present in the original work this Q-Learning approach is based on.

4.4 Exploration Policies for the Classical Approach

A common problem in RL is balancing exploration with exploitation. An agent explores by trying new actions (usually randomly) and exploits by picking an action considered optimal by its current policy. If an agent does not explore enough, it will be missing better moves that it simply never tried to use. However, if an agent does not exploit enough, than its behaviour will be almost random and it will not take enough advantage of what it is learning.

4.4.1 A Greedy Solution

In order to deal with this issue, an ϵ-greedy approach was tested. The idea is to pick an exploratory action with probability ϵ and to choose an optimal action with 1-ϵ probability. This is a very generic approach that can be applied to most RL problems in order to achieve some decent results.

$$\epsilon - greedy = \begin{cases} \epsilon & \text{Pick a random action} \\ 1-\epsilon & \text{Pick an optimal action} \end{cases} \tag{4.4}$$

We first tested this approach using a fixed ϵ, which seemed to work relatively well. However, since varying the ϵ yielded better results, that approach was ultimately chosen. With that in mind, ϵ is updated following this equation:

$$\epsilon_{episode} = \frac{1}{log(episode+1)} \tag{4.5}$$

in which the +1 in the denominator is used to avoid $log(1) = 0$ when $episode = 1$, as that would lead to 1/0.

4.4.2 Action Selection with Tags

In order to achieve a better exploration of the state space, we can instead employ action selection based on tags, as seen in [30]. This is the strategy that will also be adopted for the quantum version of the algorithm, with the necessary adjustments. We will cover both the key concepts related to this approach, as well as how they are applied in this work.

The idea to use flags for action selection was initially presented in [9]. The main concept is to associate each state's actions with tags, in order to identify which ones have been rewarded positively in previous action selections. During action selection, the agent can take advantage of these flags to reflect on its choices, so that it is more likely to select an action capable of yielding a good result. This process is known as reflection.

In [24], a specific way to utilize the flags in the reflection process is mentioned. Though there are some differences, as we will see later, this strategy is the baseline for the approach utilized in this thesis, so we will start by explaining how it works. Every action of each state starts out by being tagged. When an action is selected for a certain state, if the corresponding reward is not positive, then the tag is removed. If there is a situation in which all the actions relative to a certain state have lost their flags, then all of them become flagged once again, ensuring that there is always at least one action with a flag for each state. During the reflection process, the agent repeatedly attempts to sample a flagged action for a predefined number of iterations R, which corresponds to reflecting about its choice. Each of these steps of deliberation can be seen as a classical random walk over a directed weighted graph, in which the transition probabilities lead the agent from the state to each of the possible actions. More accurately, each step can be described as a random walk over a Markov Chain, with the transition probabilities to each action having the same value as the probability of selecting that action while in that state. Figure 4.1 is an example of this type of Markov Chain. The objective of this iterative process is to ultimately approximate $\widetilde{\pi}_s$ for each state s, which is a re-normalized version of the original stationary distribution π_s, but with support only over the flagged actions $f(s)$. $\widetilde{\pi}_s$ can be visualized in Equation 4.6.

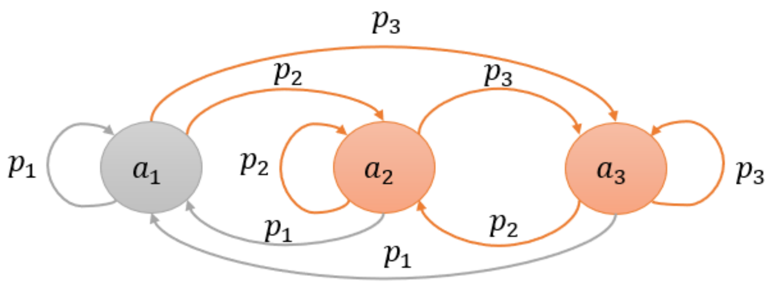


Figure 4.1: A Markov chain for a specific state, with the transition probability to each action matching the probability of selecting that action when in that state. The flagged actions are coloured orange. The reflection process can be thought of as a random walk over this Markov Chain, repeated for up to R iterations.

$$\widetilde{\pi}_s(a) = \begin{cases} \frac{\pi_s(a)}{\sum_{a' \in f(s)} \pi_s(a')} & \text{If a} \in f(s) \\ 0 & \text{Else} \end{cases} \tag{4.6}$$

The concept of applying a random walk so as to transition to one of the possible actions is explored in this thesis. For Connect Four specifically, we consider the transition from each state into its available actions, each of which corresponds to inserting a disc on a specific column. These actions all start out by being flagged and may be removed by a process which will be explained later in this section. We can see this illustrated in Figure 4.2.

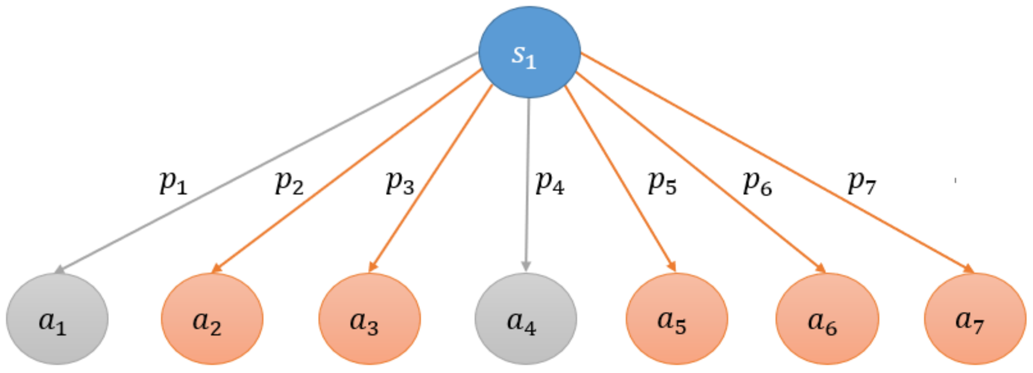


Figure 4.2: A graph representing the transition probabilities for each action of a specific state, with each action corresponding to inserting a disc on that column. The flagged actions are coloured orange. Each iteration of the reflection process can be seen as a random walk on a Markov Chain similar to the one on Figure 4.1, with the transition probabilities matching the ones seen here.

This strategy of gradually removing flags according to a criteria and deliberating on the action choice for a specific number of iterations was used as a starting point in [30], in conjunction with a soft-max policy that will be described below. The approach for action selection in this thesis uses the same key ideas as the one developed for that work, which focused on improving the exploration of the state space for Checkers.

The main difference between this method and the previously mentioned one [24] is in the update mechanism of the flags. Instead of deleting flags based on whether or not their transitions were rewarded, instead they are removed if their corresponding Q value was smaller than 0. That way, the update takes into account a more nuanced view of how each transition impacts the odds of winning a game. This is due to the fact that simply using the immediate reward leads to ignoring that a move evaluated as worse at the moment can potentially lead to a game winning scenario and vice versa. In general, using the Q value makes the most sense when considering more complex environments, in which the immediate reward does not necessarily present the full picture. This change is especially crucial for this thesis in particular, considering that the reward function only rewards the final transition. Meanwhile, if the action selected had a positive Q value despite not having a tag, then it receives a new one. Also, when all actions of a certain state lose their flags, all of them except the last one picked gain it back, as opposed to simply allowing every action of that state to have a flag again. This mechanism for updating flags happens after the reflection process and can be seen in Algorithm 3, in which the *flagged_actions* and the *q_values* are relative to the current state. Keep in mind that this is the same process proposed in [30].

Algorithm 3 Flag Update Mechanism

```
1: procedure UpdateFlags($flagged\_actions, q\_values, action\_selected$)
2:     if $q\_values[action\_selected] < 0$ then
3:         Remove $action\_selected$ from $flagged\_actions$
4:     else
5:         if $action\_selected$ not in $flagged\_actions$ then
6:             Append $action\_selected$ to $flagged\_actions$
7:     if $flagged\_actions$ is empty then
8:         $flagged\_actions \leftarrow$ all available actions, except $action\_selected$
```

As for the probabilities of selecting each action for each state, a Boltzmann distribution, also known as soft-max policy, was used. The key concept for this distribution is to make the probability of selecting an action related to its Q value. That way, action selection can benefit from randomness in order to improve the exploration, while still exploiting its current knowledge by choosing actions with higher Q values more often. Furthermore, it makes use of a temperature T, which controls the balance between exploration and exploitation. T decreases with the number of episodes, which corresponds to gradually increasing the relative probability of selecting actions with high Q values, leading to more exploitation as the agent becomes more familiar with the environment. This probability distribution can be seen in Equation 4.7, in which $A(s)$ represents the set of actions for state s, a represents an action and $Q(s, a)$ represents the Q value of a state-action pair.

$$P(a|s) = \frac{e^{Q(s,a)/T}}{\sum_{a' \in A(s)} e^{Q(s,a')/T}} \tag{4.7}$$

We can see the reflection process, in which the agent attempts to sample a flagged action for a maximum of R iterations, in Algorithm 4. If no flagged action is obtained in R steps, then the last action sampled is chosen. Notice that this is the previously mentioned process explored in [24], but using a Boltzmann distribution for the probabilities of each action, as was done in [30].

Algorithm 4 Classical Reflection

```
1: procedure ClassicalReflection($flagged\_actions, R$)
2:     for $[0, R[$ do
3:         $action \leftarrow$ sample according to Boltzmann Distribution
4:         if $action$ in $flagged\_actions$ then
5:             break
6:     return $action$
```

4.5 A Quantum Exploration Policy

As previously mentioned, this exploration policy is the quantum version of the action selection with tags introduced in the preceding section, first proposed in [30], which itself was based on [24]. As the objective of this type of exploration policy is to output flagged actions with high probability, it is crucial to guarantee that this happens consistently. This is where Quantum Computing can come into play, making it possible to achieve a quadratic speedup in the number of iterations necessary to obtain a tagged action during the reflection process, as will be explained during this section. Achieving this kind of speedup can be very beneficial, as it implies that we are not reaching the maximum number of iterations R as often, which would otherwise result in sampling random actions far too frequently, which goes against the purpose of the algorithm.

We will now cover the quantum reflection process and how it differs from the classical approach. Recall that the aim of the reflection process is to approximate the $\widetilde{\pi}_s$ seen in Equation 4.6, which is the re-normalized stationary distribution for state s, with support only over the flagged actions. In the classical version, this is accomplished with a classical random walk over a Markov Chain, at each step of the iterative process, as seen in Figure 4.1. So naturally, the quantum equivalent would be to use a quantum walk in its place, as shown in [24]. This process is similar to the search via quantum walk presented in [21], in which a randomized Grover search is applied in order to find marked elements. In this case, the marked elements correspond to the flagged actions.

It is important to mention that in a standard quantum walk, it would be necessary to employ two quantum registers [21]. This implies the use of $2 \times log(N)$ qubits for a set of N elements, with N corresponding to the number of actions available when considering the context of action selection. However, as explained in [30], for this action selection process, we are essentially considering the rank-one Markov Chains mentioned in [13, 24], since each state is being assigned a Markov Chain containing solely the actions for that state, with transitions to each action having the probability of selecting that action while in that state. As such, every action in the chain has the same transition probabilities to the other actions, which means that it is possible to use the same unitary transformation to encode them. In other words, it is only necessary to utilize a single quantum register with $log(N)$ qubits. This is important, as it means that not only are less qubits required, but also that the number of controlled operations on the qubits is reduced, both of which are key to achieving successful quantum advantage in the current quantum context. Once again, Figure 4.1 illustrates this type of Markov Chain.

The first step of this quantum reflection process is to encode the stationary distribution π_s, which leads us to the quantum state $|\pi_s\rangle$. This state can be seen in Equation 4.8, with $\pi_s(a)$ representing the probability of selecting action a and $|a\rangle$ being its corresponding basis state. This part of the process will be explained in more detail in the following section.

$$|\pi_s\rangle = \sum_{a\in A(s)} \sqrt{\pi_s(a)}\, |a\rangle \tag{4.8}$$

From there, it is necessary to repeatedly apply two reflection operators in succession, in an identical manner to a randomized Grover algorithm. Note that the reflections performed by the reflection operators are related with the quantum states, not to be confused with the reflection process in which the agent reflects over its actions for a maximum of R steps. The number of repetitions for the two reflections, T, is chosen uniformly at random in $[0, 1/\sqrt{\epsilon}]$, with ϵ representing a lower bound on the probability of an action being flagged. The reason for this choice is due to the fact that a randomized Grover algorithm should be applied a number of times chosen uniformly at random from that interval [21].

The first operator is a reflection over the flagged actions, which is referred to as $ref(f(s))$. Its aim is to check which actions have a tag and then flip the phase of every action belonging to $f(s)$. This is achieved with the use of a Boolean function $F(a)$, which returns 1 if the action has a tag and 0 otherwise. By applying the oracle $-1^{F(a)}$, the actions are evaluated, with the flagged ones being subjected to a phase kickback, which is a rotation by a phase of π radians. Notice that this is very similar to the first step of Grover's algorithm, which was explained in Subsection 2.2.6. This operator is represented in Equation 4.9.

$$|a\rangle \to \begin{cases} -|a\rangle & \text{If a} \in f(s) \\ |a\rangle & \text{Else} \end{cases} \tag{4.9}$$

As for the second operator, it is a reflection over the encoded stationary distribution $|\pi_s\rangle$. By reflecting the actions over $|\pi_s\rangle$ after having flipped the phase of the ones with flags, the amplitudes of the flagged actions will greatly increase, while the amplitudes of the other actions will slightly decrease. This is identical to the inversion around the mean in Grover's algorithm, known as Grover's Diffusion Operator, which was explained in Subsection 2.2.6 and is shown in Figure 2.2. This operator can be seen in Equation 4.10, with I representing the identity.

$$ref(\pi_s) = 2\,|\pi_s\rangle\,\langle\pi_s| - I \tag{4.10}$$

Typically, quantum phase estimation is used in order to approximate this second reflection, which can otherwise be difficult to implement. However, since we are considering transformations being applied on a single quantum register, as mentioned earlier, it is possible to obtain this operator using the transformations shown in Equation 4.11, which was proposed in [30]. D_0 represents a reflection over the state $|0\rangle$. U_π is the operator used to encode the stationary distribution and will be explained in more detail in the following section. $U_\pi^\dagger$ is the conjugate of U_π. First, $U_\pi^\dagger$ reverses the encoding, then D_0 rotates over state $|0\rangle$ and lastly U_π encodes the probability distribution once again. This process is equivalent to a reflection over π_s.

$$ref(\pi_s) = U_\pi D_0 U_\pi^\dagger \tag{4.11}$$

Finally, the quantum circuit is measured. If the action obtained is tagged or the maximum number of reflections R has been reached, then it is selected by the agent. Otherwise, the entire process repeats for that state.

The full quantum reflection procedure is shown in Algorithm 5. The code used for this algorithm was adapted from the one used in [30].

Algorithm 5 Quantum Reflection

```
 1: procedure QuantumReflection($flagged\_actions, R, \epsilon$)
 2:     $circuit \leftarrow$ encode the quantum state $|\pi_s\rangle$
 3:     for $[0, R[$ do
 4:         $T \leftarrow$ chosen uniformly at random in $[0, 1/\sqrt{\epsilon}]$
 5:         for $[0, T[$ do
 6:             $circuit$.reflection_operator($flagged\_actions$)
 7:             $circuit$.diffusion_operator()
 8:         $action \leftarrow circuit$.measure()
 9:         if $action$ in $flagged\_actions$ then
10:             break
11:     return $action$
```

As the average number of repetitions required for this process is $1/\sqrt{\epsilon}$, as opposed to $1/\epsilon$ for the classical case, we achieve a quadratic speedup for obtaining a flagged action [21], while still obeying the fact that the actions need to be sampled according to the re-normalized distribution [24]. While this strategy is not regarded as being optimal for simple searching problems [21], it has been shown to be optimal when it is merely necessary to output an action from a good approximation of the re-normalized distribution $\widetilde{\pi}_s$ [24].

4.6 Encoding The Action Selection Probabilities

The encoding of the action selection probabilities was done using the coherent controlization scheme introduced in [13]. This method was also utilized in [30] and the code utilized for the probability encoding in this thesis was adapted from that work.

Coherent controlization is built around a unitary operator U_π, which can take the quantum state from $|0\rangle$ into the encoded stationary distribution $|\pi_s\rangle$.

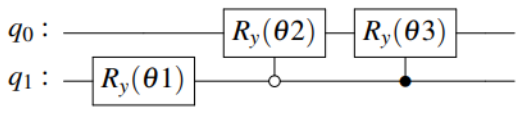
$$U_\pi |0\rangle = |\pi_s\rangle \tag{4.12}$$

This type of encoding utilizes angles as parameters, with these corresponding to the probabilities from the stationary distribution. As such, it is first necessary to convert the classical probabilities into angles. These probabilities are the ones obtained from the Boltzmann Distribution, just as in the classical process. The procedure used to convert them involves recursively calculating angles and can be visualized in Algorithm 6. Keep in mind that the *calculateAngle* process called is a simple function that returns $2 \times \arccos \sqrt{x}$, with x being the value received by *calculateAngle*.

Algorithm 6 Probabilities to Angles

```
1: procedure ProbabilitiesToAngles($probabilities, previous = 1.0$)
2:     if len(probabilities) $= 2$ then
3:         return calculateAngle($probabilities[0]/previous$)
4:     $lhs, rhs \leftarrow$ split($probabilities, 2$)          ▷ Split the probabilities in 2 arrays
5:     $angles \leftarrow []$
6:     $angles$.append(calculateAngle(sum($lhs$)/$previous$))
7:     $angles$.append(probabilitiesToAngles($lhs, sum(lhs)$))
8:     $angles$.append(probabilitiesToAngles($rhs, sum(rhs)$))
9:     return $angles$
```

After obtaining the angles, U_π can be constructed by taking advantage of controlled rotations over the y-axis, using the angles as parameters. This type of quantum circuit can be seen in Figure 4.3, which was adapted from [30].



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

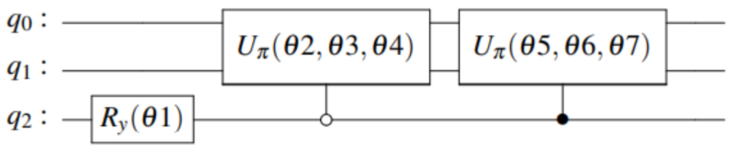

q_0 :
q_1 :
q_2 :
$R_y(\theta 1)$
$U_\pi(\theta 2, \theta 3, \theta 4)$
$U_\pi(\theta 5, \theta 6, \theta 7)$
(b) Circuit implementing the operator U_π for a Markov chain with up to 8 states.

Figure 4.3: Examples of coherent controlization. Adapted from [30]. The empty circle $\circ$ represents that the corresponding transformation is applied if the control state is $|0\rangle$, while the filled circle $\bullet$ acts the same way in the presence of $|1\rangle$. The circuit at the top shows the use of 2 qubits to encode up to $2^2 = 4$ actions, while the one at the bottom represents the use of 3 qubits to encode up to $2^3 = 8$ actions. Notice that the second circuit is constructed by recursively applying the first one.

This page is intentionally left blank.

Chapter 5

Experimental Setup

In this chapter, the experimental setup is covered. First, the choices of parameters related with the Randomized Negamax opponent are explained. From there, an overview of how the Reinforcement Learning (RL) agents were trained and tested is given. Lastly, aspects related with the experimental setup of the RL agents are presented. To avoid confusion, any time the term "agent" is used, it is meant to represent either a classical or quantum agent, whereas the term "opponent" corresponds to the opponent making use of the Randomized Negamax strategy.

5.1 Setup for the Opponent (Randomized Negamax)

The probability ω of selecting a random move was chosen to be 0.3. This value was picked after some preliminary tests with different values for ω. Values lower than this led to the agents playing too many similar games, which defeated the purpose of using a randomized approach. Meanwhile, values higher than 0.3 were making the challenge too trivial for the agents, as the use of too many random moves made it simple to setup winning scenarios.

As for the *depth* parameter, which represents the number of moves the opponent should look ahead, a value of 2 was selected. Using a *depth* of 1 would mean that the opponent only takes into account the positions it can reach after making its move, while completely ignoring what the agent is capable of doing during the action selection that comes after that. As for higher depths, while they would make for interesting experiments, preliminary tests revealed that even a depth of 4 would lead to a very significant increase in the amount of time necessary to run a set of games. This longer play time is a consequence of two factors. Firstly, the opponent takes longer to make a decision, as it needs to look further ahead for every choice it makes. Secondly, since the opponent is effectively a stronger player, the agents are forced to train for a much higher number of episodes, so that they can achieve reasonable win rates. Due to time constraints, doing multiple runs of these longer experiments was simply not feasible.

5.2 Training and Testing the RL Agents

The agents were trained by letting them play multiple games against the Randomized Negamax opponent and measuring the average number of moves played and the number of wins, losses and draws they obtained, among other statistics which will be detailed below.

This process was repeated for 20 independent runs, each with a unique seed, with the results for each metric being the average across all of those runs.

The number of games played depended on whether the agent was acting as player 1 or player 2. Since going second in Connect Four is a considerable disadvantage, the agents need to play more games in order to reach reasonable win rates. As such, 1800 episodes were used to train the going first agents, while 3600 were used for the going second agents.

The training time was also recorded. Considering the limitations of the current quantum context, the quantum agents are expected to perform worse on this metric.

Furthermore, the number of states explored during training was also used as a metric. This number can allow for a better understanding of how the win rates obtained by each type of agent relate with the number of states they were exposed to during training.

For the agents with tagged action selection, the average number of iterations to obtain a flagged action was registered. This metric is used to compare the classical and quantum versions of this approach on how consistent they are at sampling flagged actions. Specifically, a lower average number of iterations indicates that flagged actions are obtained more frequently, since the reflection limit R is not being hit as often, which would results in the last action sampled being chosen instead. Note that when a flagged action is not hit during action selection, surpassing the limit R, the corresponding number used for the average will keep summing the iterations of the following action selection process, and so on, until a flagged action is eventually hit.

After finishing the training, each agent was tested by playing 1000 games against the Randomized Negamax opponent. This was done to obtain a good representation of the performance of the agents after their training had been completed. For these tests, only the average number of moves and the number of wins, losses and draws were considered. Just as in the training phase, the results shown will be the average across 20 seeds. Note that while testing, all of the agents made use of an optimal action selection function, which always returns the action with the highest Q value. This was done with the intent of enabling a fair comparison between the different agents, only taking into account the actions considered optimal by their current policy, which corresponds to exploiting the knowledge they obtained while training.

5.3 Setup for the RL Agents

The learning rate α of the Q-Learning equation was set to 0.5. This applies to every type of agent.

For the agents using action selection with tags, the maximum number of iterations R was set to 5. As for the temperature parameter T seen in Equation 5.1, since the idea of using a Boltzmann distribution was inspired by [30], the temperature function used in that work was used as a basis, but altered to fit the context of our approach. In particular, a parameter that we refer to as δ was changed so that a high inverse temperature $\frac{1}{T}$ is reached around 1800 or 3600 episodes, depending on whether it is a going first or a going second agent. In Figure 5.1, the inverse temperature functions can be observed.

$$T = 0.2 + \frac{20 - 0.2}{1 + e^{0.35 \times (episode/\delta)}} \tag{5.1}$$

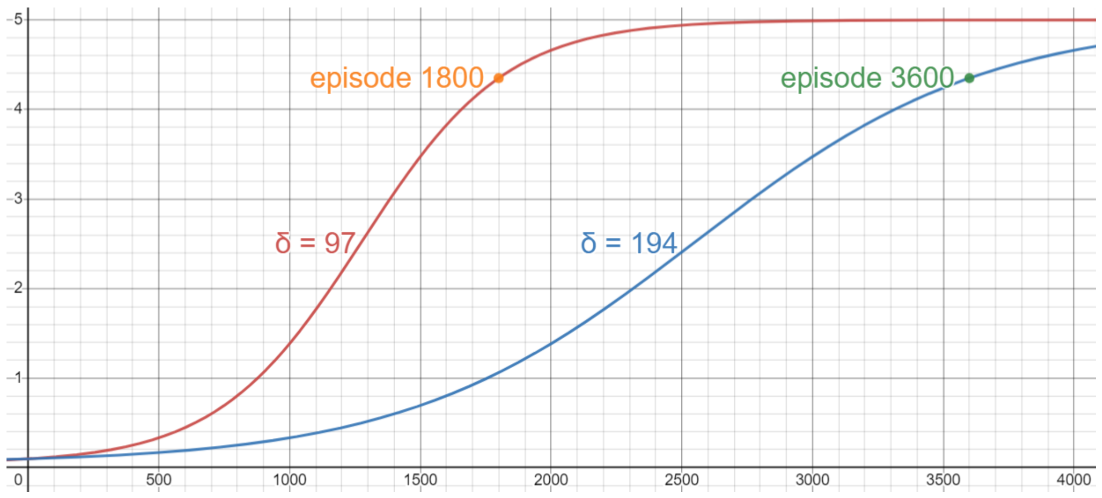


Figure 5.1: The Inverse Temperature Functions.

This page is intentionally left blank.

Chapter 6

Results and Discussion

In this chapter, we present and discuss the results obtained from the experiments, which were designed as described in Chapter 5. We first go over the results from the player 1 agents. Then, we do the same for the player 2 agents. Lastly, we provide answers for the Research Questions.

6.1 Player 1 Agents

In Table 6.1, we can observe the results that the player 1 agents obtained during training. The average number of moves was higher for the classical ϵ-greedy agents, which corresponds to playing longer games. Since their training win rate is still quite similar to the other agents, this is likely due to them taking longer to convert strong board positions into a win, which would imply that they might have overly emphasized exploration in favour of exploitation. Regarding the number of states explored, the classical ϵ-greedy agents were once again the exception, being exposed to a significantly higher number of them, which gives credibility to the previous statement about them overly favouring exploration. This higher number of states explored could be a consequence of them having played longer games, which would naturally lead to exploring more states.

Table 6.1: Player 1 Agents (Training)

Agent	Iterations	States	Win%	Loss%	Draw%	Moves	Time(h)
Classical, ϵ-greedy		**9743**	8.1	91.3	0.6	**20.4**	2.6
Classical, tags	1.495	8138	7.6	92.4	0.02	15.5	2.1
Quantum, tags	**1.414**	7986	8.6	91.4	0.04	15.4	2.8

As for the average number of iterations necessary to obtain a flagged action, the quantum agents required less iterations than their classical counterparts, which is in accordance with the theory surrounding them. While the difference might seem small, there are two factors to consider. First, that the maximum number of iterations R was set to 5. And second, that a considerable number of flagged actions are obtained in the first iteration. As such, it becomes clear that a difference of almost 0.1 is quite substantial, as is verified by their results while testing in Table 6.2, in which the quantum agents had the highest win rate by a significant margin.

Meanwhile, the quantum agents spent the most time training, which is not a surprise considering the difficulties in simulating quantum computation. However, the difference between these agents and the classical ϵ-greedy agents is not very significant, which could be due to them having played longer games, on average.

Table 6.2: Player 1 Agents (Testing)

Agent	Win%	Loss%	Draw%	Moves
Classical, ϵ-greedy	**19.3**	78.4	**2.3**	**22.1**
Classical, tags	63.2	35.9	0.9	15.0
Quantum, tags	**71.4**	28.3	0.3	12.8

In Table 6.2, the results obtained while testing the player 1 agents are presented. We once again find that the classical ϵ-greedy agents played longer games, which is likely due to them having favoured exploratory moves too frequently while training. This had a clear impact on how often they drew a game, with them having a higher number of draws than the other agents. Looking at the win rates, we can conclude that the tagged action selection policies are much more effective at training the agents, as they both managed to achieve reasonable results, while the classical ϵ-greedy agents had a very poor performance. Between the flagged action selection agents, the quantum version had a higher win rate, which validates the hypothesis of them having better training efficiency than the classical variant.

6.2 Player 2 Agents

In Table 6.3, we can view the results from the training of the player 2 agents. Once again, the quantum agents required less iterations to obtain flagged actions, which should be due to the quantum speedup achieved during the reflection process. The time spent training was lower for the classical agents, just as before. The difference might seem more pronounced this time, but this is merely due to the values themselves being higher, with the classical agent taking around 70% of the time in both cases. Of course, as you scale to more complex problems, while the relative difference might be the same, adding more hours to each run of an experiment becomes quite a detriment. This further shows that quantum simulation has very clear limitations.

Table 6.3: Player 2 Agents (Training)

Agent	Iterations	States	Win%	Loss%	Draw%	Moves	Time(h)
Classical, tags	1.604	13109	6.2	93.8	0.02	14.4	**3.2**
Quantum, tags	**1.477**	12995	5.5	94.4	0.03	14.3	4.5

The results from testing the player 2 agents are shown in Table 6.4. Comparing the win rates obtained with those of the player 1 agents, we can see that even with double the number of training episodes, the player 2 agents still obtained a worse performance. This is in line with the theory regarding the advantage of playing first in Connect Four. As for how the results of the player 2 agents stack up to each other, it is surprising to see that the classical agents had a better win rate. Initially, this seemed to be due to the quantum agents having more runs with outliers (3 as opposed to 1). However, if we remove those outliers, then the win rate becomes very similar for the two agents, at around 57-58%, when we would expect the quantum agents to do better. Since the quantum agents obtained flagged actions more reliably, that means that the algorithm is still working as intended. The most likely explanation for these strange results is the number of episodes chosen. If we were to pick a higher number of episodes, to the point where the quantum agents are achieving at least a 70% win rate, as is the case for the player 1 agents, it is possible that the quantum agents would perform better. Since training the player 2 agents is a much more complex problem, it is possible that for the number of episodes selected, exploration still needs to be favoured significantly in comparison with exploitation, with the relatively low win rates of around 50% supporting this theory. As such, the advantage of selecting flagged actions more consistently would not really come into play. But for a higher number of episodes, in which exploitation needs to be balanced very carefully, the quantum agents would likely achieve better results. Regardless, these results show that we should not expect the quantum agents to achieve a better win rate in every situation, even if they are sampling flagged actions more consistently.

Table 6.4: Player 2 Agents (Testing)

Agent	Win%	Loss%	Draw%	Moves
Classical, tags	**55.9**	43.8	0.3	15.3
Quantum, tags	50.1	49.7	0.2	16.2

6.3 Answering the Research Questions

We will now provide an answer to each Research Question, taking into account the work developed and the results obtained.

Research Question 1: In what way can a quantum approach improve the exploration of the state space for the Reinforcement Learning (RL) problem of Connect Four?

To improve the exploration of the state space, we employed the flagged action selection policy used in [30], which combines flagged action selection [24] with a Boltzmann distribution. The quantum version of the flagged action selection policy takes advantage of quantum walks to achieve a quadratic speedup during the reflection process, so that it can avoid reaching the maximum number of iterations R reliably, which would otherwise lead to a random action being selected. This in turn ensures that it will output flagged actions more consistently, which allows the quantum agents to explore the state space more efficiently, since they can focus on exploring the more promising paths. The Boltzmann distribution further stresses this idea, by enhancing the probabilities of the actions with higher Q values.

Looking at the results for the player 1 agents, we can verify that the quantum version of the approach needed less iterations to obtain flagged actions, on average. This, combined with its higher win rate, indicates that the quantum agents explored the state space more efficiently, by taking advantage of their ability to select flagged actions with more consistency.

Research Question 2: What changes need to be made so that a similar approach to the one used for Checkers [30] can be applied to Connect Four, in order to better explore the state space?

We utilized offline Deep Q-Learning, as opposed to online non-Deep Q-Learning. The preliminary tests revealed that using a classical Neural Network (NN) to learn the correct Q values led to more efficient training. This had an impact in different aspects of the work, which will be highlighted below.

For the state representation, the author of that work divided the Checkers board in different sections and identified each state by how many pieces were in each one. This way, similar states were grouped together, which allowed them to deal with the high dimensionality of the state-space of that board game. Since the state-space complexity of Connect Four is not as high and the NN used already allows it to generalize past knowledge, we decided to represent the state with a simple matrix that corresponds to the Connect Four board. When the NN has to take in the representation of the environment, it receives a matrix with the state representation appended to a vector identifying the action selected, as explained in Subsection 4.3.1. The NN then predicts the Q value of the corresponding state-action pair using that representation, which will naturally allow it to gradually assign similar values to similar state-action pairs.

In [30], reward shaping was utilized to deal with the fact that the winner of a game is only decided during the last transition of the game, which means that this would typically be the only transition with a non-zero reward. In other words, they introduced smaller rewards to every transition, based on whether the agent was achieving certain goals that in theory should give it a better winning probability. In our work, some preliminary tests were done with this type of approach. However, the agents trained with this methodology were not learning how to win efficiently, so we decided to drop it. It is possible that better reward shaping functions could have been used, but this shows that this type of approach is not strictly necessary and might not be ideal for every board game. It is worth considering that the offline Deep Learning approach employed helped the agents to get a better grasp of the impact of each transition, due to it training after each batch of 300 games, using the NN to evaluate every position from the batch. Once again, the NN takes into account both the state and the action selected, which leads to it ultimately attributing similar Q values to similar state-action pairs. The weights of the network are adjusted at the end of the training of each batch, by comparing the current Q values with the target Q values from Equation 4.3, in order to better predict the correct Q values.

In summary, it is possible to extend the quantum flagged action selection to other board games, as long as some adjustments are made.

Research Question 3: How does this exploration of the state space compare to a standard ϵ-greedy approach?

In the experiments regarding the player 1 agents, both the classical and the quantum flagged action selection policies achieved far better win rates than the ϵ-greedy approach, while exploring fewer states. The two flagged agents explored a similar number of states, but the quantum version had the higher win rate of the two. These results suggest that the quantum flagged action selection policy led to a better balance between exploration and exploitation than the standard ϵ-greedy method, which seems to have overvalued exploration too much. Thanks to the flags implemented in the flagged approach, it was possible to identify the more appealing actions in an efficient manner, allowing the agents to exploit their knowledge more reliably, so as to avoid neglecting exploitation during training.

Research Question 4: Considering that going second is a significant disadvantage in Connect Four, how does the quantum approach fare in this scenario?

As seen in Tables 6.3 and 6.4, the quantum agents still sampled flagged actions more consistently, which shows that the approach can scale to more complex problems. However, the win rates obtained seem to indicate otherwise, with the classical agents having better results. If the outliers are ignored, then their win rates are similar, but this is still different than the expectation of the quantum agents performing better. As mentioned in Section 6.2, this is likely due to the number of episodes chosen. Since they are insufficient to obtain a relatively high win rate for either the classical or the quantum agents, it seems that exploration still needs to be largely favoured for that number of episodes. As such, the advantage of the quantum agents would not be as helpful and could perhaps even be detrimental on some level. Assuming a higher number of episodes, high enough to allow for high win rates, the quantum agents would likely perform better, as balancing the exploitation correctly would become key.

This page is intentionally left blank.

Chapter 7

Conclusion

In this work, quantum tagged action selection was applied to the Reinforcement Learning (RL) context of Connect Four. The objective was to to better balance how much an agent should explore and how much it should exploit, by taking advantage of a quadratic speedup for outputting flagged actions. Doing so extended the scope of this technique to board games other than Checkers, showcasing its general usefulness in this domain.

To that end, we created a Randomized Negamax opponent for the agents to play against, whose strategy is similar to a standard Negamax, but with some added randomness in its move choices, so that the agents had a more complicated challenge to overcome. We also defined Connect Four as a Markov Decision Process (MDP) and explained the Q-Learning approach taken.

In order to extend this exploration policy to Connect Four, an offline Deep Learning method was incorporated, with a classical Neural Network (NN) training on each batch of games. This NN evaluated each state-action pair by using the representation of the Connect Four board and the action selected as input, which naturally allowed for similar state-action pairs to be rated similarly. This way, it was possible to deal with the relatively high dimensionality of the problem.

The experiments involved training both classical and quantum agents and evaluating their performance. By comparing both the classical and the quantum tagged action selection policies with a standard ϵ-greedy approach, we found that the former were trained far more efficiently. Between the flagged action selection agents, the quantum version found flagged actions in less iterations, on average, which ultimately led to it achieving a higher win rate.

There were also experiments for agents that trained as player 2. While the quantum agents still sampled flagged actions in less iterations, their win rates were not superior. Since neither the classical nor the quantum agents won a relatively high percentage of games going second and the quantum version still obtained flagged actions in less steps, these results are likely due to the number of episodes not being high enough to learn the environment in depth. In this scenario, exploration still needs to be disproportionately favoured, so the advantage of the quantum agents would not be as relevant. Assuming a number of episodes high enough for both agents to achieve relatively high win rates, it is likely that the quantum version would perform better.

This work presents a reasonably successful application of Quantum RL in the current quantum context, further demonstrating the potential for practical applications in this area.

Future work could involve testing trained agents on real quantum hardware. This was not possible due to time constraints, but it would be intriguing to see if agents trained on a quantum simulator could perform well on real quantum computers. It would also be interesting to scale the problem to more difficult opponents. For example, Randomized Negamax agents that look ahead more than two moves or a completely distinct type of opponent. As for expanding the work to other contexts, the most natural choice would probably be Chess, which has a much higher complexity than both Connect Four and Checkers and is a well known and studied game.

This page is intentionally left blank.

References

[1] IBM Quantum. https://quantum-computing.ibm.com/, 2021. (Accessed: 10 December 2021).

[2] Cirq. https://quantumai.google/cirq, 2021. (Accessed: 10 December 2021).

[3] Q# and the quantum development kit. https://azure.microsoft.com/en-gb/resources/development-kit/quantum-computing/, 2022. (Accessed: 18 January 2022).

[4] Grover's algorithm. https://quantum-computing.ibm.com/composer/docs/iqx/guide/grovers-algorithm, 2022. (Accessed: 22 April 2022).

[5] Amira Abbas, Stina Andersson, Abraham Asfaw, Antonio Corcoles, Luciano Bello, Yael Ben-Haim, Mehdi Bozzo-Rey, Sergey Bravyi, Nicholas Bronn, Lauren Capelluto, Almudena Carrera Vazquez, Jack Ceroni, Richard Chen, Albert Frisch, Jay Gambetta, Shelly Garion, Leron Gil, Salvador De La Puente Gonzalez, Francis Harkins, Takashi Imamichi, Pavan Jayasinha, Hwajung Kang, Amir h. Karamlou, Robert Loredo, David McKay, Alberto Maldonado, Antonio Macaluso, Antonio Mezzacapo, Zlatko Minev, Ramis Movassagh, Giacomo Nannicini, Paul Nation, Anna Phan, Marco Pistoia, Arthur Rattew, Joachim Schaefer, Javad Shabani, John Smolin, John Stenger, Kristan Temme, Madeleine Tod, Ellinor Wanzambi, Stephen Wood, and James Wootton. Learn quantum computation using qiskit. http://community.qiskit.org/textbook, 2020. (Accessed: 3 July 2022).

[6] Elias Alvarez. A practical introduction to quantum computing: from qubits to quantum machine learning and beyond. https://indico.cern.ch/event/970903/, 2020. (Accessed: 18 January 2022).

[7] Andris Ambainis. Quantum walks and their algorithmic applications. *International Journal of Quantum Information*, 1(04):507–518, 2003.

[8] Ville Bergholm, Josh Izaac, Maria Schuld, Christian Gogolin, M Sohaib Alam, Shahnawaz Ahmed, Juan Miguel Arrazola, Carsten Blank, Alain Delgado, Soran Jahangiri, et al. Pennylane: Automatic differentiation of hybrid quantum-classical computations. *arXiv preprint arXiv:1811.04968*, 2018.

[9] Hans J Briegel and Gemma De las Cuevas. Projective simulation for artificial intelligence. *Scientific reports*, 2(1):1–16, 2012.

[10] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[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*, 2020.

[12] Rob Dawson. Learning to play connect 4 with deep reinforcement learning. https://codebox.net/pages/connect4, 2020. (Accessed: 10 December 2021).

[13] V Dunjko, N Friis, and H J Briegel. Quantum-enhanced deliberation of learning agents using trapped ions. *New Journal of Physics*, 17(2):023006, jan 2015.

[14] Stefan Edelkamp and Peter Kissmann. Symbolic classification of general two-player games. In Andreas R. Dengel, Karsten Berns, Thomas M. Breuel, Frank Bomarius, and Thomas R. Roth-Berghofer, editors, *KI 2008: Advances in Artificial Intelligence*, pages 185–192, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[15] Alan Geller. Why do we need q#? https://devblogs.microsoft.com/qsharp/why-do-we-need-q/, 2022. (Accessed: 18 January 2022).

[16] Craig Gidney. Quirk: Quantum circuit simulator. https://algassert.com/quirk, 2022. (Accessed: 17 January 2022).

[17] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[18] Jen-Yueh Hsiao, Yuxuan Du, Wei-Yin Chiang, Min-Hsiu Hsieh, and Hsi-Sheng Goan. Unentangled quantum reinforcement learning agents in the openai gym. *arXiv preprint arXiv:2203.14348*, 2022.

[19] Yuxi Li. Deep reinforcement learning. *CoRR*, abs/1810.06339, 2018.

[20] Owen Lockwood and Mei Si. Playing atari with hybrid quantum-classical reinforcement learning. In *NeurIPS 2020 Workshop on Pre-registration in Machine Learning*, pages 285–301. PMLR, 2021.

[21] Frédéric Magniez, Ashwin Nayak, Jérémie Roland, and Miklos Santha. Search via quantum walk. *SIAM Journal on Computing*, 40(1):142–164, jan 2011.

[22] S. Mangini, F. Tacchino, D. Gerace, D. Bajoni, and C. Macchiavello. Quantum computing models for artificial neural networks. *EPL (Europhysics Letters)*, 134(1):10002, apr 2021.

[23] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[24] 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(3), jul 2014.

[25] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: a Modern Approach, 3rd. Edition.* Pearson Education, Inc., 2010.

[26] 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 van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, Oct 2017.

[27] Andrea Skolik, Sofiene Jerbi, and Vedran Dunjko. Quantum agents in the gym: a variational quantum algorithm for deep q-learning. *Quantum*, 6:720, may 2022.

[28] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction, 2nd Edition*. MIT press, 2018.

[29] Mario Szegedy. Quantum speed-up of markov chain based algorithms. In *45th Annual IEEE symposium on foundations of computer science*, pages 32– 41, Nov 2004.

[30] Miguel Teixeira. Quantum reinforcement learning applied to games. *Universidade do Porto. MSc Thesis*, 2021. https://hdl.handle.net/10216/135628.

[31] Salvador Elías Venegas-Andraca. Quantum walks: a comprehensive review. *Quantum Information Processing*, 11(5):1015–1106, Jul 2012.

[32] Qing Wei, Hailan Ma, Chunlin Chen, and Daoyi Dong. Deep reinforcement learning with quantum-inspired experience replay. *IEEE Transactions on Cybernetics*, pages 1–13, 2021.

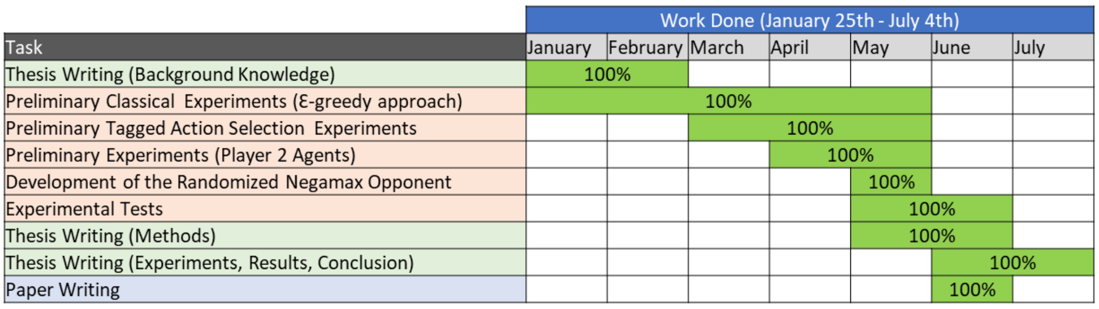
[33] Gilad Wisney. Deep reinforcement learning and monte carlo tree search with connect 4. https://towardsdatascience.com/deep-reinforcement-learning-and-monte-carlo-tree-search-with-connect-4-ba22a4713e7a, 2019. (Accessed: 10 December 2021).

[34] Christa Zoufal, Aurélien Lucchi, and Stefan Woerner. Quantum generative adversarial networks for learning and loading random distributions. *npj Quantum Information*, 5(1):103, Nov 2019.

Appendices

This page is intentionally left blank.

Appendix A



	Work Done (January 25th - July 4th)						
Task	January	February	March	April	May	June	July
Thesis Writing (Background Knowledge)	🟩 100%	🟩					
Preliminary Classical Experiments (Ɛ-greedy approach)	🟩	🟩	🟩 100%	🟩	🟩		
Preliminary Tagged Action Selection Experiments			🟩	🟩 100%	🟩		
Preliminary Experiments (Player 2 Agents)				🟩 100%	🟩		
Development of the Randomized Negamax Opponent					🟩 100%		
Experimental Tests					🟩 100%	🟩	
Thesis Writing (Methods)					🟩 100%	🟩	
Thesis Writing (Experiments, Results, Conclusion)						🟩 100%	🟩
Paper Writing						🟩 100%	

Figure 1: Gannt chart showing the work developed in the context of this thesis.