Quantum Reinforcement Learning Applied to Board Games

Miguel Teixeira
Department of Informatics Engineering (DEI), Faculty of Engineering (FEUP), University of Porto
Porto, Portugal
up201605150@fe.up.pt

Ana Paula Rocha
Artificial Intelligence and Computer Science Lab (LIACC), Department of Informatics Engineering (DEI), Faculty of Engineering (FEUP), University of Porto
Porto, Portugal
arocha@fe.up.pt

António J. M. Castro
Artificial Intelligence and Computer Science Lab (LIACC), University of Porto
Porto, Portugal
ajmc@fe.up.pt

ABSTRACT

Reinforcement learning is a machine learning paradigm where an agent learns how to optimize its behavior solely through its interaction with the environment. It has been extensively studied and successfully applied to complex problems of many different domains in the past decades, i.e., robotics, games, scheduling. However, the performance of these algorithms becomes limited as the complexity and dimension of the state-action space increases. Recent advances in quantum computing and quantum information have sparked interest in possible applications to machine learning. By taking advantage of quantum mechanics, it is possible to efficiently process immense quantities of information and improve computational speed. In this work, we combined quantum computing with reinforcement learning and studied its application to a board game to assess the benefits that it can introduce, namely its impact on the learning efficiency of an agent. From the results, we concluded that the proposed quantum exploration policy improved the convergence rate of the agent and promoted a more efficient exploration of the state space.

CCS CONCEPTS

• **Computing methodologies → Reinforcement learning**.

KEYWORDS

reinforcement learning, quantum computing, board games

ACM Reference Format:
Miguel Teixeira, Ana Paula Rocha, and António J. M. Castro. 2021. Quantum Reinforcement Learning Applied to Board Games. In *IEEE/WIC/ACM International Conference on Web Intelligence (WI-IAT '21), December 14–17, 2021, ESSENDON, VIC, Australia.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3486622.3493944

1 INTRODUCTION

Reinforcement learning (RL) is a machine learning (ML) paradigm, where a learner learns how to behave through its interaction with a dynamic environment and the feedback received. Each action is associated with a reward signal, either positive or negative. Through trial and error, the learner must converge to a policy that maximizes said reward signal and achieves the desired goal.

Several challenges arise in RL problems. One of these challenges is the *exploration vs exploitation* dilemma [13]. To achieve its optimization goal, a learner not only has to exploit previously successful actions but also explore new actions to fully learn the environment and improve its future decisions. The dilemma is that the learner must choose actions that yield reward but can only do so after exploring new actions in the action space. Neither exploration nor exploitation can be ignored and, therefore, it is imperative to come up with a strategy that balances both points. Another challenge to overcome is the slow learning speed, due to the *curse of dimensionality* [2]. As the complexity of the problem increases, so does its state and action space. This negatively influences the rate of convergence, making it more difficult for the learner to learn an optimal strategy efficiently. These challenges are not trivial to solve.

Several approaches have emerged to tackle them, such as Dynamic Programming, Monte-Carlo methods and Temporal-Difference learning. Each approach has its strengths and weaknesses and differs in terms of efficiency and speed of convergence. Although these approaches introduce several improvements to key areas of RL, there is still room for more.

Quantum computing (QC) introduces a new way for computers to efficiently process large amounts of information, offering theoretical quantum improvements in Machine Learning [3]. It has motivated numerous studies in this area and led to an increase of interest in quantum-enhanced ML algorithms. Several studies have been made to evaluate the impact that these quantum approaches can have on reinforcement learning. These studies have shown promising results when applying quantum-enhanced algorithms to a RL problem, describing major improvements in the performance and learning efficiency compared with classical RL algorithms.

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

WI-IAT '21, December 14–17, 2021, ESSENDON, VIC, Australia
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-9115-3/21/12…$15.00
https://doi.org/10.1145/3486622.3493944

The rest of this paper paper is organized as follows. Section 2 presents a review of existing literature on quantum computing and quantum reinforcement learning. Section 3 defines the game of Checkers as an RL problem, detailing the representation used for the state space, the reward function and the opponent. Afterward, section 4 details the proposed solution, including the classical and quantum algorithms to be implemented and the challenges associated with each approach. Section 5 delineates the experiments conducted to assess the performance of both approaches. Finally, section 6 presents an overview of the main contributions and future work.

2 RELATED WORK

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

Dong et al. [7] describe a novel QRL framework, taking advantage of the superposition principle and quantum parallelism. In the proposed method, the state and the action set are represented as quantum systems in superposition state, where each basis state corresponds to a possible state/action of the problem. The agent's policy in a given state is dictated by the measurement result of the quantum system, causing it to collapse to one of the actions according to their associated probability amplitude. After each action, the algorithm updates and amplifies the amplitude of said action through Grover iterations, depending on the value functions and the rewards received. An analysis of the simulated results demonstrates that the quantum algorithm achieves a good balance of exploration and exploitation through its action selection policy, scaling efficiently as the dimension of the problem increases.

Briegel and las Cuevas [5] propose a different approach to solve RL problems. The framework, denoted projective simulation (PS), considers a quantum agent interacting with a classical environment whose policy is determined by a simulation-based projection. The percepts and actions (or a sequence of both) are represented within the agent as clips, forming a weighted network of clips. The clip network represents the experience of the agent, either simulated or built through its past interactions with the environment. When the agent is given a percept, it triggers its past experience and causes a particular clip to be excited, resulting in a random walk through the network. Eventually, when an actuator clip is reached, the agent outputs an action. As the agent learns, the structure of the network and the weights between clips are updated according to the observed rewards, increasing the probability of performing better in the future.

The previous framework was later extended in the work of Paparo et al. [10], which introduced the concept of reflecting projective simulation. Unlike a standard PS agent, a reflecting PS agent fully mixes a Markov chain and, once mixed, proceeds to sample from its stationary distribution until a tagged action clip is obtained. The authors also introduce a quantum implementation of the agent, where the action selection process is enhanced by employing a quantum search algorithm based on quantum walks, resembling a randomized Grover-like search algorithm [8]. The results demonstrate that the algorithm enables the agent to efficiently explore its internal memory, with a proven quadratic speedup over its classical counterpart.

Albarrán-Arriagada et al. [1] study the application of a reinforcement learning algorithm in a measurement-based adaptation protocol. In the work presented, an agent interacts and learns the model of an unknown environment through successive measurements, using an auxiliary register. After each iteration, the agent adapts its quantum state depending on the feedback received from the measurement of the register, converging to the quantum state of the environment. The algorithm was tested under different parameters and was capable of cloning an arbitrary quantum state with a high degree of fidelity (over 90%) and a small number of iterations, achieving a better performance than quantum tomography.

Chen et al. [6] explore the application of variational quantum circuits (VQC) to approximate the Q value function in deep reinforcement learning. In their work, the authors follow the same principles as classical deep reinforcement learning, but replaced the network with a parameterized quantum circuit. The circuit is composed of several layers, each consisting of several single-qubit rotations that produce a highly entangled quantum state. The results demonstrate that the quantum network requires fewer parameters when compared to classical approaches.

Overall, these quantum approaches demonstrate an advantage over their classical counterparts, either in computational complexity or scalability.

3 CHECKERS AS A RL PROBLEM

3.1 State Representation

The game of Checkers presents a state space with around 5×10^{20} unique states [12]. Due to its considerable complexity, the state space requires special attention to how it is represented.

A naive approach is to consider each board setting as a unique state of the problem. As the agent learns, it will eventually identify the optimal moves for the states that it has previously visited. Although this approach is simple to implement, it often leads to poor results due to the high dimensionality of the game. It is necessary to reduce the state space complexity using a more compact representation.

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

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

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

i) the material of each player, i.e., the pieces of each player
ii) how that material is distributed across the board

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

In order to implement this second feature, we took inspiration from heat maps where an area can be hotter or colder depending on the value of a certain variable. As such, we divided the board into different areas, for a total of 7 areas, as shown in figure 1.

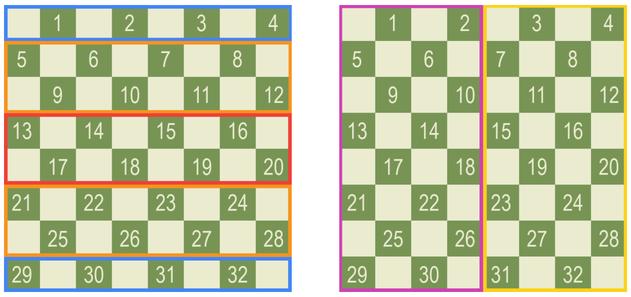

Figure 1: The board is divided in 7 areas. Horizontally: the blue areas represent the back row; orange areas represent the back; red area represents the center of the board. Vertically: the pink and the yellow areas represent the left and the right of the board, respectively.

The horizontal areas were chosen to help the agent recognize both offensive and defensive moves from its opponent, e.g., the opponent moving a piece from its back row or trying to make a push towards the center. The representation places greater emphasis on the center of the board and the back rows of both players, as control over those areas is the key to win the game. The representation also distinguishes between the left and right sides of the board, thereby providing more information to the agent.

For each area of the board, we calculate the material advantage that a player has over the other, following the formula depicted in equation 1.

$$\text{MaterialAdvantage} = \text{MaterialScore}_{Agent} - \text{MaterialScore}_{Opp} \quad (1)$$

where $\text{MaterialScore}_{Agent}$ and $\text{MaterialScore}_{Opp}$ represents the material score of the agent and the opponent in a certain area, respectively. The material advantage is calculated from the difference between the material score of the players in the area. A player's material score is given by the weighted sum of the number of *men* and *kings* that are present in the area, as shown in equation 2.

$$\text{MaterialScore} = 1.0 \times \#\,\text{Men} + 2.0 \times \#\,\text{Kings} \quad (2)$$

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

3.2 Reward Function

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

It is critically important to formulate a good reward structure that encourages the agent to complete its goal, as it directly impacts its performance. In a zero-sum game, the goal is quite simple: learn a strategy that allows the agent to win against its opponent. Thus, a straightforward reward function would be to reward or punish the agent depending on whether it wins or loses the game, respectively. Equation 3 is an example of it.

$$\mathcal{R}(s,a,s') = \begin{cases} 2.0, & \text{if the agent wins} \\ -2.0, & \text{if the agent loses} \\ 0.0, & \text{otherwise} \end{cases} \quad (3)$$

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

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

A possible solution is to use a denser reward function that returns a non-zero reward to the agent after each action taken, thus aiding it in recognizing and distinguishing good states from bad states. Designing such a function is often not trivial, as it requires domain knowledge about the problem at hand to know when to reward or punish the agent. Nevertheless, one could design a heuristic function that evaluates the state of the environment and use it to provide additional information, encouraging the agent to achieve certain sub-goals that might later help it reach the main goal. This approach is denoted reward shaping [13], as it modifies the original reward function by incorporating domain knowledge into it in order to help guide the agent in its exploration of the state space.

However, some problems might arise when applying reward shaping. If the new reward function is ill-formed, it might introduce bias to the agent's behavior and cause it to stray further away from its main goal. In order to prevent these issues, we employed potential-based reward shaping [9], which transforms a sparse reward function $\mathcal{R}$ into a denser reward function $\mathcal{R}'$, as shown in equation 4.

$$\mathcal{R}'(s,a,s') = \mathcal{R}(s,a,s') + \mathcal{F}(s,a,s') \tag{4}$$

where $\mathcal{F}$, denoted shaping reward function, is a real-valued function that provides additional information to the reward signal. The function $\mathcal{F}$ is a potential-based function, according to equation 5.

$$\mathcal{F}(s,a,s') = \gamma\ \Phi(s') - \Phi(s) \tag{5}$$

where γ is the discounting factor and Φ is a heuristic function that evaluates and determines how good a certain state is. By calculating the difference between the values of Φ for the current and the last state, $\mathcal{F}$ provides information regarding whether the transition was beneficial for the agent and if the action taken drove it closer to its goal.

In this work, we applied reward shaping to enhance the reward function in equation 3. In Checkers, the material advantage of a player has a strong correlation with winning or losing the game [11]. Thus, we devised a simple heuristic function that evaluates the board by calculating the relative material advantage, as shown in equation 6.

$$\Phi(s) = \frac{\text{MaterialScore}_{Agent} - \text{MaterialScore}_{Opp}}{\text{MaterialScore}_{Agent} + \text{MaterialScore}_{Opp}} \tag{6}$$

where $\text{MaterialScore}_{Agent}$ and $\text{MaterialScore}_{Opp}$ represents the material score of the agent and the opponent, respectively, calculated from the formula described in equation 2.

The heuristic function in equation 6 returns a real-valued number between $[-1.0, 1.0]$, representing the relative material advantage that a player has over its opponent. By imbuing said heuristic function into $\mathcal{R}'$, the new reward function will encourage the agent to consider how the total material is distributed between both players and how its actions affect that distribution. Thus, the agent will prioritize the states where it has more material than its opponent.

3.3 Opponent

For the opponent, we decided to pit the agent against an adversary with some domain knowledge, as it provides more interesting results than playing against a random opponent with no reasoning behind its choice of actions. The opponent was implemented using Negamax, a recursive depth-first search algorithm that searches the game tree for the next best move.

The evaluation of a game state is done by an heuristic function based on a linear combination of independent weighted features of the board, as shown in equation 7.

$$\text{Evaluation} = turn \times \sum_i w_i \times f_i, \tag{7}$$

where each feature f_i is associated with a weight w_i, representing its importance in relation to the other features. Negamax requires a symmetric evaluation function, which returns a score relative to the player that is being evaluated. To do so, the score of each feature f_i is given by calculating the difference between the feature score of each player, as shown in equation 8.

$$f_i = f_i(Agent) - f_i(Opp) \tag{8}$$

The choice of the features and the weights used in the evaluation determines how the opponent behaves. We considered four features of the board: the material of each player, the pieces with an unimpeded path to the back row, the pieces at risk of being captured and the position of the pieces. These features encourage the opponent to capture its adversary's pieces while defending its own, all while moving its pieces forward towards the center and the back row. The weights associated with each feature were chosen to prioritize certain features over others, e.g., the opponent prioritizes protecting a piece under attack rather than promoting a piece. Although the opponent does not play optimally, it offers a fair challenge to the learning agents.

4 IMPROVING EXPLORATION WITH QC

4.1 An Improved Exploration Policy

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

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

In order to address the previous issues, we propose an exploration policy based on the soft-max action selection and further enhanced through the use of flags. A flag is a simple structure that serves as the short-term memory of the agent and is used to identify actions that it believes worth exploring. Its use was first proposed in the work of [5], where it was demonstrated that it could significantly improve the performance of the agent at certain tasks. This exploration policy allows the agent to reflect on its choice of actions and helps it select more useful actions while exploring the environment.

4.1.1 Action Selection. When reaching a certain state, the agent starts by determining how likely it is to choose each of the available actions based on its knowledge of how good they are. To do so, the agent employs a Boltzmann distribution and converts the Q value of each action into the probability of selecting said action, as described in equation 9.

$$\mathbb{P}(a \mid s) = \frac{e^{\beta Q(s,a)}}{\sum_{a_i \in A(s)} e^{\beta Q(s,a_i)}} \tag{9}$$

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

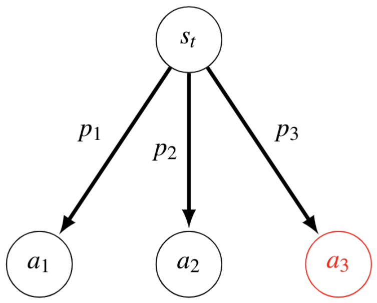

Figure 2: The action selection process can be visualized as a directed two-layered network, where the root node is the current state and the leaf nodes are all the available actions. The probability of transitioning from the root node to each leaf node is given by the Boltzmann distribution. In this example, the action a_3 is the only flagged action.

In the next step of the process, the agent repeatedly samples an action based on the probabilities calculated earlier until it obtains a flagged action. Such procedure can be thought of as a simple random walk on a directed two-layered network, as depicted in figure 2, where the agent repeatedly hops from the root node to a leaf node until a flagged action is reached.

The probability of choosing a flagged action may become low during the learning process, e.g., the total probability of choosing a flagged action is less than 10%. Thus, we introduced a fixed parameter R, which denotes the maximum number of available iterations to the agent. If no flagged action is obtained before reaching the maximum number of tries, the agent chooses the last sampled action.

4.1.2 Flag Update. Initially, all actions are flagged, as the agent considers all transitions to be equally good. During the learning process, the agent continuously assesses the impact of an action on the environment and either adds or removes its flag, depending on whether the outcome was considered good or bad, respectively. When the last remaining flag is removed, the process restarts and all the other actions except the last one are flagged again. Therefore, for any given state, there will always be at least one flagged action.

When following this exploration policy, a problem that arises is how to evaluate the outcome of an action. Board games often require the agent to deal with both immediate and delayed rewards, where it might have to choose bad actions to gain the upper hand over its opponent, e.g., sacrifice a piece to later promote another piece. Thus, the agent determines whether a flag should be added or removed using the Q value of said action, calculated as shown in equation 10.

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \left(R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right) \tag{10}$$

where $Q(S_t, A_t)$ denotes the current Q value of the action A_t in the state S_t, R_t is the reward received, α is the learning rate and γ is the discounting factor. The Q value represents the agent's current knowledge regarding a certain action and takes into account its immediate and delayed rewards.

4.2 A Quantum Approach

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

In order to solve these issues, we enhanced the action selection process using a quantum algorithm, inspired by the work of [10] on reflecting projective simulation. The goal is to amplify the probability of choosing a flagged action to near unity (100%) while maintaining the relative probabilities between said actions, thus achieving the distribution presented in equation 11.

$$\widetilde{\mathbb{P}}(a \mid s) = \begin{cases} \dfrac{\mathbb{P}(a \mid s)}{\sum_{a_i \in \mathcal{F}(s)} \mathbb{P}(a_i \mid s)}, & \text{if flagged} \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

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

It is important to note that the algorithm we employ in this work is not optimal in searching for a flagged action. Instead, its task is to achieve a good approximation of the distribution mentioned above and then sample an action based on it, for which the algorithm is optimal [10].

4.2.1 Action Selection. Upon reaching a certain state, the quantum agent internally constructs an ergodic and reversible Markov chain P based on a two-layered network, as shown in figure 3. The states of the chain correspond to the available actions and the probability of transitioning to each state is the same as the original probability of choosing the corresponding action. Any state associated with a flagged action remains flagged in the Markov chain.

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

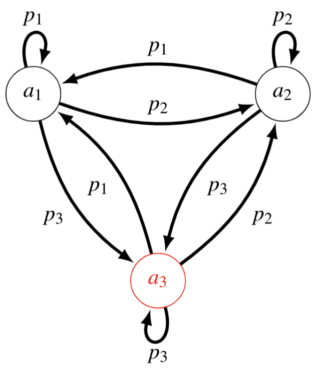

Figure 3: An ergodic and reversible Markov chain, built from the two-layered network depicted in figure 2.

state, according to equation 12.

$$|\pi\rangle = \sum_{a \in \mathcal{A}(s)} \sqrt{\pi_a}|a\rangle \tag{12}$$

where π_a denotes the probability of the action a under the stationary distribution π. Each action a is represented by its respective basis state $|a\rangle$ in a N-dimensional Hilbert space $\mathcal{H}$, where N is the number of states in the Markov chain. Then, the agent sequentially applies two operators to the quantum state. The first operator performs a reflection over the set of flagged actions and shifts the phase of the basis states that correspond to a flagged action, as described in equation 13.

$$\text{ref}(\mathcal{F}) : |a\rangle \longrightarrow \begin{cases} -|a\rangle, & \text{if } a \in \mathcal{F}(s) \\ |a\rangle, & \text{otherwise} \end{cases} \tag{13}$$

The second operator performs a reflection over the initial stationary distribution, thus amplifying the amplitude of the basis states that correspond to a flagged action. Said reflection is defined in equation 14.

$$\text{ref}(\pi) = 2|\pi\rangle\langle\pi| - I \tag{14}$$

where $|\pi\rangle$ denotes the encoded stationary distribution of P. The process closely resembles Grover's search algorithm, as each iteration rotates the quantum state closer to a state with support only over the flagged actions. After applying the required number of iterations, the resulting quantum state approximately encodes the distribution described in equation 11, following equation 15.

$$Q^T|\pi\rangle = \sin((2T+1)\theta) \sum_{x \in \mathcal{F}(s)} \sqrt{\frac{\pi_x}{\epsilon}}|x\rangle + \cos((2T+1)\theta) \sum_{y \notin \mathcal{F}(s)} \sqrt{\frac{\pi_y}{1-\epsilon}}|y\rangle \tag{15}$$

where T is the number of iterations, ϵ is the probability of the flagged actions under the stationary distribution π, such that $\epsilon = \sum_{a \in \mathcal{F}(s)} \pi_a$, and $\theta = \arcsin(\sqrt{\epsilon})$ [4]. Finally, the agent measures its quantum register and, if flagged, outputs the measured action. Otherwise, the process restarts.

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

5 EXPERIMENTS AND RESULTS

5.1 Experiments

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

5.2 Agents Parameters

In order to compare the classical and the quantum versions of the exploration policy, we developed a total of 6 Q-learning agents. All agents have the same learning rate α and discount factor γ, as the goal is to assess the impact that the parameter R and the action selection have on the results. The parameters used in the 6 developed agents are presented in table 1.

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

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

Table 1: Agents and their parameters

5.3 Analysis of the Results

Experiment 1 - Opponent with look-ahead of 3 moves (6x6 board). In the first experiment, all agents (classical and quantum) learned a strategy that wins against the opponent, with the classical agents winning roughly the same number of games as their quantum counterparts, as shown in figure 4.

It is clear that the agents CQL-2 and QQL-2 converged slightly faster than the remaining agents and learned a winning strategy while exploring fewer states. The difference from the other agents comes from their greater R value, which compels them to be more selective in their choice of actions. When comparing these two agents, both achieved a similar performance across all metrics. However, the quantum agent revealed a slight advantage at the

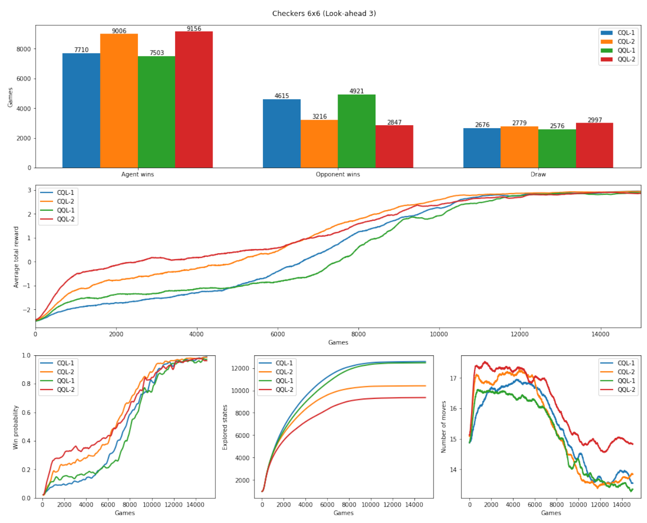

Figure 4: Results of the agents playing in a 6x6 board against an opponent with a look-ahead of 3 moves.

beginning of the learning process, where it quickly learned how to not lose against its opponent.

Overall, the results demonstrate that all agents were able to learn the environment and win against the opponent, although it is not clear whether the quantum agents outperformed the classical agents. However, we can still conclude that the parameter R has a positive impact on the learning process of an agent, increasing the convergence rate while requiring less exploration of the state space.

Experiment 2 - Opponent with look-ahead of 3 moves (8x8 board). In this experiment, the agents were presented with a much more challenging environment. The results obtained, presented in figure 5, accentuated the differences between the agents, brought by the action selection method used and the parameter R.

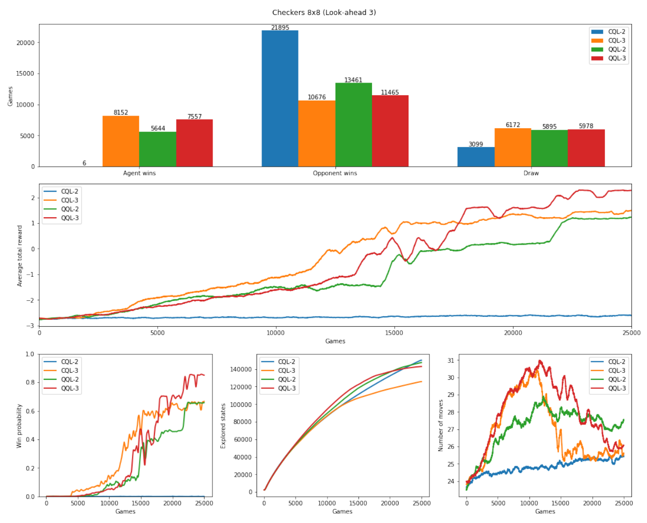

Figure 5: Results of the agents playing in a 8x8 board against an opponent with a look-ahead of 3 moves.

Starting by assessing the impact of the parameter R, the results demonstrate that the agents with a higher value were more successful than those with a lower value. By analyzing the performance of the agents CQL-2 and QQL-2, we quickly notice that these agents revealed a slower convergence rate when compared to the agents CQL-3 and QQL-3, respectively. Therefore, we can conclude that a higher R value enables the agents to take full advantage of the action selection process and its flags, allowing them to better re-think their choice of actions and achieve a more efficient exploration of the state space. Although its impact on the learning process was somewhat noticeable in the previous experiment, its advantages become evident in a more complex environment with large state and action spaces.

When comparing the classical agents with their quantum counterparts, the results confirm that the quantum agents demonstrate a better learning efficiency. The advantages of the quantum approach are especially evident in agents with smaller R values, as seen between CQL-2 and QQL-2. In this example, the quantum agent achieved a 70% win probability, while the classical version was not able to learn the environment and did not show any signs of progress throughout the entire learning process. As both agents explored roughly the same number of states, we can infer that the action selection method influences how an agent explores the state space and chooses its actions, i.e., has an impact on how the agent learns the environment.

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

Experiment 3 - Changing opponent (8x8 board). For the final experiment, the agents faced an opponent that changes its strategy over time. Initially, the opponent starts with a look-ahead of 1 move, allowing the agents to converge to a winning strategy. Then, after a few thousand games, the opponent plays more intelligently and uses a look-ahead of 3 moves.

During the first part of the process, the agents' learning curve followed the same results obtained in experiment 2.2, as expected. When the opponent changed its strategy, the agents' performance plummeted as they now had to adapt their policy. The results, shown in figure 6, demonstrate that only the agents CQL-3 and QQL-3 were able to do so, as the remaining agents were unable to re-learn a new strategy in such a short number of games.

The reason behind these differences in performance is easy to explain. After converging and following a policy for a prolonged number of games, the probability of choosing a flagged action becomes quite high. When the opponent changes its strategy, some flags get reset and the new flagged actions are now associated with a low probability, as it takes time for the agent to update their corresponding Q values. In this scenario, a high R value allows the agents to repeat the action selection process multiple times and, eventually, obtain a flagged action. Thus, the parameter R influences

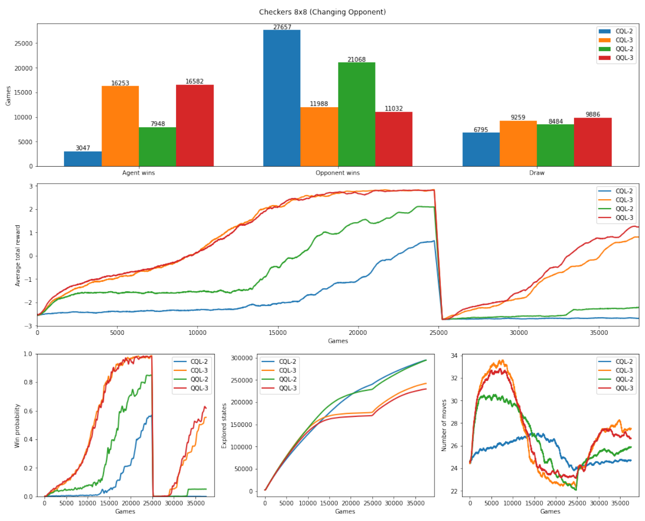

Figure 6: Results of the agents playing in a 8x8 board against an opponent that changes its strategy.

how fast an agent reacts to a change in the environment, at the cost of more time spent in decision-making. As the quantum algorithm provides a quadratic speedup in obtaining a flagged action, the quantum agent QQL-3 displayed slightly better adaptability than its counterpart, quickly converging to around 60% win probability.

5.4 Results Overview

When playing in a 6x6 board, all agents performed similarly and quickly learned the environment, even against stronger opponents. However, the experiments performed in a 8x8 board introduced the agents to a more complex problem with a larger state space. These experiments stressed the differences between the agents and allowed us to better distinguish the benefits introduced by the quantum algorithm.

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

6 CONCLUSION

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

The results obtained revealed that the quantum approach promotes a more efficient exploration of the state space, achieving a theoretical quadratic speedup in obtaining a suitable action. Thus, a quantum agent demonstrates a better convergence rate over its classical counterpart while exploring fewer states. Furthermore, a quantum agent also displays better adaptability in changing environments. The advantages of the quantum approach become more evident as the size of the state and action spaces increases.

It is also possible to identify certain areas of improvement. First off, some aspects of the proposed exploration policy were not explicitly evaluated due to the lack of collected metrics, such as the number of iterations required to obtain a flagged action. Analyzing the differences in those aspects would allow us to better compare the classical and quantum approaches. Secondly, it would also be interesting to study the learning process of the agents when playing against human players. Unlike the opponents tested in section 5, a human player is constantly changing and improving its strategies, allowing us to further test the agent's adaptability to changes in the environment and their ability of generalization. Finally, another interesting experiment would be to test both approaches in a more complex board game, e.g., Chess, and study how their performance scales in problems with larger state and action spaces.

REFERENCES

[1] Francisco Albarrán-Arriagada, Juan C. Retamal, Enrique Solano, and Lucas Lamata. 2018. Measurement-based adaptation protocol with quantum reinforcement learning. *Physical Review A* 98 (10 2018), 042315. Issue 4. https://doi.org/10.1103/PhysRevA.98.042315
[2] Richard E. Bellman and Rand Corporation. 1957. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA.
[3] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. 2017. Quantum machine learning. *Nature* 549 (9 2017), 195–202. Issue 7671. https://doi.org/10.1038/nature23474
[4] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. 2002. Quantum amplitude amplification and estimation. *Quantum Computation and Information* 305 (2002), 53–74. https://doi.org/10.1090/conm/305/05215
[5] Hans J. Briegel and Gemma De las Cuevas. 2012. Projective simulation for artificial intelligence. *Scientific Reports* 2 (12 2012), 400. Issue 1. https://doi.org/10.1038/srep00400
[6] Samuel Yen-Chi Chen, Chao-Han Huck Yang, Jun Qi, Pin-Yu Chen, Xiaoli Ma, and Hsi-Sheng Goan. 2020. Variational Quantum Circuits for Deep Reinforcement Learning. *IEEE Access* 8 (2020), 141007–141024. https://doi.org/10.1109/ACCESS.2020.3010470
[7] Daoyi Dong, Chunlin Chen, Hanxiong Li, and Tzyh-Jong Tarn. 2008. Quantum Reinforcement Learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 38 (10 2008), 1207–1220. Issue 5. https://doi.org/10.1109/TSMCB.2008.925743
[8] Frédéric Magniez, Ashwin Nayak, Jérémie Roland, and Miklos Santha. 2011. Search via Quantum Walk. *SIAM J. Comput.* 40 (1 2011), 142–164. Issue 1. https://doi.org/10.1137/090745854
[9] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. 1999. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML '99)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 278–287.
[10] Giuseppe Davide Paparo, Vedran Dunjko, Adi Makmal, Miguel Angel Martin-Delgado, and Hans J. Briegel. 2014. Quantum speedup for active learning agents. *Physical Review X* 4 (2014), 031002. Issue 3. https://doi.org/10.1103/PhysRevX.4.031002
[11] Arthur L. Samuel. 1959. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development* 3, 3 (1959), 210–229. https://doi.org/10.1147/rd.33.0210
[12] Jonathan Schaeffer, Neil Burch, Yngvi Bjornsson, Akihiro Kishimoto, Martin Muller, Robert Lake, Paul Lu, and Steve Sutphen. 2007. Checkers Is Solved. *Science* 317 (9 2007), 1518–1522. Issue 5844. https://doi.org/10.1126/science.1144079
[13] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction* (Second ed.). The MIT Press, Cambridge, MA, USA.