



# Performance Study of Minimax and Reinforcement Learning Agents Playing the Turn-based Game Iwoki

Santiago Videgaín & Pablo García Sánchez

To cite this article: Santiago Videgaín & Pablo García Sánchez (2021) Performance Study of Minimax and Reinforcement Learning Agents Playing the Turn-based Game Iwoki, Applied Artificial Intelligence, 35:10, 717-744, DOI: [10.1080/08839514.2021.1934265](https://doi.org/10.1080/08839514.2021.1934265)

To link to this article: <https://doi.org/10.1080/08839514.2021.1934265>



Published online: 15 Jun 2021.



Submit your article to this journal [↗](#)



Article views: 1295



View related articles [↗](#)



View Crossmark data [↗](#)





Citing articles: 1 View citing articles [↗](#)

RESEARCH ARTICLE



# Performance Study of Minimax and Reinforcement Learning Agents Playing the Turn-based Game Iwoki

Santiago Videgáin <sup>a</sup> and Pablo García Sánchez <sup>b</sup>

<sup>a</sup>Escuela Superior De Ingeniería Y Tecnología (ESIT), Universidad Internacional De La Rioja, Logroño, Spain; <sup>b</sup>Department of Lenguajes and Computer Systems, ETSIIT, University of Granada, Granada, Spain

## ABSTRACT

*Iwoki math* is an abstract board game that consists on placing tiles and that combines the calculation of simple mathematical operations with the spatial perception of two-dimensional objects. Due to its inherent features, it is also a very challenging environment to test different artificial intelligence technologies and methods. In this paper, a series of intelligent agents with different reasoning and decision capacities have been developed based on different artificial intelligence techniques applied to game theory, such as Minimax or Reinforcement Learning. Their capabilities have been tested by playing games with each other, but also against human players, obtaining remarkable results. The experimental results ratify conclusions already known at a theoretical level but also provide a new contribution that could be the basis for future research.

## ARTICLE HISTORY

Received 24 July 2020  
Revised 2 November 2020  
Accepted 6 November 2020

## Introduction

It is well known that board games not only can provide a lot of fun, but they also reinforce creativity and teach decision making, as well as respect for rules. But they have been particularly challenging in the field of artificial intelligence practically from its inception (Yannakakis and Togelius 2018). The reasons are that they are closed, but well-defined environments and most of them have a huge search space where the performance of complex algorithms can be analyzed.

The *iwoki math* (hereinafter *iwoki*) is a game that combines these benefits with its educational character. It reinforces the capacity for analysis and activates the cognitive functions to put into practice the best strategy throughout the games. It also trains the spatial perception of two-dimensional objects to make way for the calculation of simple mathematical operations. This game, even still being in the prototype phase, has been tested in playful environments, and it was a finalist in the Meeple Factory prototype competition 2019. Example of the elements of the game and rules are available in the Appendix Section.

**CONTACT** Pablo García Sánchez  [pablogarcia@ugr.es](mailto:pablogarcia@ugr.es)  Department of Computer Architecture and Computer Technology, ETSIIT, University of Granada, Granada, Spain

This article discusses the development of various agents who compete with each other by playing *iwoki*. In order to limit the scope of this work, the study is carried out on a simplified version of the *iwoki*, where only two players are involved in each game. Different agents have been tested in different game configurations: a Random Agent, a Greedy Agent, a Minimax Agent and a Reinforcement Learning Agent. Also, a human player will interact with the game using an interface as a Human Agent.

The objective of this paper is to determinate which implementations are most appropriate for agents who play *iwoki* in order to win and to evaluate which configurations optimize results. On the other hand, to provide new insights in the field of artificial intelligence applied to games, using *iwoki* as a research tool. It will allow to compare different agents, establish certain conclusions about them and explore new lines of research.

The rest of the paper is structured as follows: first, a background on artificial intelligence in board games and the particularities of *iwoki* are presented. Then, in [Sections 3 and 4](#), the methodology and experimental setup are described. The results are discussed in [Section 5](#). Finally the conclusions and future work are presented.

## Background

### *Iwoki* Characterization

Game theory is based on the analysis of the behavior of various agents, focused on pursuing a goal and interacting with each other according to their strategies. The agents obtain a positive or negative gain according to the success or failure of the interactions carried out. The scope of game theory reaches the economy, history, politics, international relations, military strategy, etc. (Osborne et al. 2004).

According to Yannakakis and Togelius (2018), we can classify games in different ways, with respect to some of their characteristics: number of players, observability, stochasticity, action space and time granularity.

Depending on the number of players, the game can be with or without an opponent: In a game without an opponent, only one player is involved. The most used methods for its implementation are the brute force method without heuristics, or if we want to take into account the time and memory consumption as determining factors, the heuristic search by means of A\* or IDA\* (Russell and Norvig 2010). Examples of games without an opponent are 8-puzzle, solitaire, etc. In games with an opponent two or more agents compete for a common goal. Examples of games with an opponent are *3-in-a-row*, *chess*, *go*, *checkers*, etc. Other examples of where there is more than one opponent are *Poker* or *Monopoly*. On the other hand, cooperative games are those where the participants share common interests and act for the benefit of

such ends without competing with each other. In noncooperative games, each player is concerned only with his/her own interests (Zagal, Rick, and Hsi 2006).

A relevant feature is the order of the moves or actions (time granularity). It may be that the players' turns are alternate or random. They may also be sequential (such as *parcheesi*) or simultaneous, in which the players make their decisions at the same time (rock, paper or scissors, or even in more complex games, such as Real Time Simulators (Fernández-Ares et al. 2017)).

The knowledge of each player about the rest catalogs the game as one of perfect information, when it does not affect randomness and the information of each player is shared for the others (*checkers*), or of imperfect information, where randomness does intervene and each player hides information from the rest (*black jack*).

Determinism refers to the intervention of randomness or stochasticity. Thus, a game can be deterministic if there is no room for randomness, or non-deterministic (or stochastic) if randomness intervenes in any of the above characteristics. For example, the HearthStone card game includes random actions after playing some cards (García-Sánchez et al. 2020). The Bayesian Game concept considers as random variables the unknown information of the game. Bayes' theorem is applied by establishing probabilities about concrete states of the game based on previously known information, after a series of previous actions carried out by the opponents. Bayesian Game, therefore, makes sense for games with imperfect information (Harsanyi 1967).

Von Newman and Morgenstern (Von Neumann and Morgenstern 2007) define the concept of *zero-sum games* with an opponent, in which everything a player manages to win corresponds to what his/her opponent loses. In other words, the sum of what one wins and the other loses is zero. An example of a zero-sum game is an individual tennis match, where each player's point is detrimental to the opponent. However, in a doubles tennis match, the points scored by a partner is also used by the other. This case is an example of a non-zero-sum play. John Forbes Nash adds a new concept for non-zero-sum cases: the so-called Nash Equilibrium, a situation in which none of the players is interested in modifying their individual strategy so as not to be disadvantaged, knowing their opponents' one (Daskalakis, Goldberg, and Papadimitriou 2009).

Given the nature of *iwoki*, the characteristics and properties that must be taken into account when implementing the agents are as follows: First, it is a game with an opponent in which two agents are involved in a noncooperative way, since each agent looks after his/her own interests and they have no common objective. Thus, each player aims to get the highest score after the final point tally, which will make him/her the winner of the game. The rules of the game, the complete description of which is available at in the Appendix Section, are the same for the two players and known by both.

The opponents will play their turn in a sequential and alternative way from the beginning to the end of the game. There are different strategies for the initial turn of the game, for the advance turns and for when the game is about to be over. The player who starts the game can also be the one who plays the last turn before the final point tally, which does not mean that he/she has more chances to win. Since everything a player wins for his/her own benefit is lost by the other player in the same proportion, and vice versa, it is categorized as a zero-sum game. In other words, what a player gains with respect to his opponent is the difference in the score obtained after the final point tally, which is accomplished by performing the best possible move in each turn. Finally, the game can be considered as deterministic or not, according to certain modifications necessary for the implementation of a particular agent, as it is specified in the Experimental Setup Section.

Once the peculiarities of this game have been understood, the next subsections will discuss the methods that are going to be used in this paper.

### **Minimax Algorithm**

For two-player games in which both players try to win and where one player's actions depend decisively on the other's, adversarial search algorithms are used. Minimax is the basic opponent search algorithm, which has been traditionally implemented from perfect information board games (Turing 1953).

Basically, it means that each player chooses the best move for himself, keeping in mind that the opponent has chosen the most harmful option for his/her opponent. For each state of the game the algorithm generates a complete search tree, applies to each terminal node a utility function (also called *evaluation function*) and performs a propagation of those values to the root node. The Minimax algorithm results very appropriate for agents who play board games, putting into practice effective both offensive and defensive strategies (Kalles and Kanellopoulos 2008).

The main problem with this method is that the tree increases exponentially with depth and with the branching factor of the tree. That is, with the number of simulated movements, the order of complexity in time is  $O(a^b)$  and the order of complexity in space is  $O(ab)$  being  $a$  the Tree Branching Factor, and  $b$  the Tree Depth.

For example, in the case of chess, if we interpret that it has an average branching factor of 30 nodes (moves on each position) and 80 moves are made throughout the game,  $30^{80}$  nodes will be explored. This is more than the number of atoms in the universe. The complexity in time and space makes a tree that contemplates all the possibilities in each move unfeasible.

As a rule, the applications of the Minimax algorithm finish the search at a certain depth. The nodes of that depth become terminals and use a state

evaluation function that determines a heuristic. This technique is called *Minimax with suspension*. It is also possible to limit the search by run time. The root node will stay with the best option found until the maximum time is reached. Applying these modifications on the Minimax algorithm will not be able to guarantee that the agent will find the optimal play, but it will be reliable if the heuristics are applied properly. Following these modifications to solve the problem of complexity, there are different techniques to improve Minimax, such as alpha-beta pruning, heuristic continuation (Russell and Norvig 2010) or the Scout Algorithm (Pearl 1980), among others. In the case of the basic Alpha-Beta pruning the number of nodes to be explored is reduced by avoiding passing through nodes in the tree that do not influence the decision of the agent. When a node does not provide a better value than the explored one until that moment, the branch is pruned.

### **Reinforcement Learning**

Reinforcement Learning is a model in which an agent is interpreted as an entity that interacts with the *environment* and learns from the consequences of its *actions*. The environment provides the agent the different alternatives in each specific *state*. For each of these alternatives the agent can perform an action. As a consequence of executing certain action, the environment will report to the agent a reward or a punishment. The execution of this action will take it to a new state with new alternatives available (Sutton and Barto 2018).

Markov Chain is a probabilistic model in which the future state is conditionally independent of past state (Kemeny and Snel 1960). Markov Decision Process (MDP) derives from Markov Chain and mathematically formalizes Reinforcement Learning problems (Kaelbling, Littman, and Moore 1996). Describe a fully observable environment, defined by the tuple  $(S, A(s), P, \gamma, R(s, a))$ , where  $S$  is the finite set of states,  $A(s)$  is the finite set of possible actions in each state  $s$  and  $P$  specifies the transitions between states. It is a probability distribution of the state  $s'$  which is accessed by executing the action  $a$  from the current state  $s$ :  $P(s', s|a)$ . The discount factor ( $\gamma$ ) indicates how important the reward is in the long term, and takes values between 0 and 1. The higher the value of  $\gamma$  is, the more important future rewards will be. Conversely, a small value will tend to only take into account immediate rewards. Finally, the reward  $(R(s, a))$ , obtained by executing action  $a$  from a state  $s$ . It is a real number and does not depend on any previous state or action. MDP must fulfill the Markov Property. This means that it is a necessary and sufficient condition that  $P(s_{t+1}|s_t) = P(s_{t+1}|s_1, \dots, s_t)$ , which implies that the future state depends only on the present state and is independent of the previous states. In other words, “given the present, the past and future are independent of each other” (Kemeny and Snel 1960).

The agent chooses actions based on a policy  $\pi$ , which defines the agent's conduct. It determines what action should be taken in each state and depends only on the current state, not the historical one. The policy does not change over time. That is, in the same state, the same action will always be performed. The optimal policy  $\pi^*$  is the one that maximizes the cumulative sum of rewards. For any MDP, there exists always a deterministic optimal policy  $\pi^*$ , which is better than or equal to all other policies. The objective of the algorithm is to find this optimal policy.

The Q-value function  $Q^\pi(s, a)$  is defined as the cumulative reward expected by choosing the action  $a$  in the state  $s$  and then following the policy  $\pi$ . The optimal Q-value function  $Q^*(s, a)$  obtains the maximum achievable reward value with a policy  $\pi$  from a state  $s$  and an action  $a$ . Consequently, knowing the optimal Q-value function  $Q^*(s, a)$  implies having the optimal policy  $\pi^*$ .

As the agent obtains rewards, the Q-values are updated by the *Bellman equation* (Sutton and Barto 2018), until the optimal Q-value is reached. Once the action  $a$  has been executed in the state  $s$ , the system must take into account the maximum Q-value for each possible action  $a'$  in the state  $s'$ . The equation is defined as

$$Q(s, a) \leftarrow (1 - \alpha) \cdot Q(s, a) + \alpha \cdot \left( R + \gamma \cdot \max_a Q(s', a') \right) \quad (1)$$

where  $\alpha$  indicates the learning rate at which the Q-values are updated in each iteration,  $R$  is the reward, and  $\max_a Q(s', a')$  refers to the estimation of the optimal future value (next state, all actions). Each suitable  $Q(s, a)$  value is stored in a table called the *Q-table*, for all state-action pairs. When the function  $Q(s, a)$  learned up to that moment is used to determine the action to be performed, the algorithm is said to work in *exploitation* mode. However, it is very common to select actions at random in order to take into account new possibilities. In this case the algorithm is said to work as *exploration* mode (Silver et al. 2016).

A basic example of implementation with Reinforcement Learning is the one developed on the simple Tic-Tac-Toe game (Song 2012). The algorithm gets the agent to end up playing competently without previous experience and allow it to reach the specified goal, such as victory, defeat or draw.

The problem with Reinforcement Learning algorithms is that they suffer from the curse of dimensionality, as the number of states grows exponentially. In order to deal with the great complexity involved in a large number of states and actions, different solutions are proposed. One of them is the MDP with hierarchical decomposition (Bai, Wu, and Chen 2015), that breaks down the problem into a set of sub-problems that are easier to solve. It establishes a different way of calculating  $Q^*$  that allows to face a great depth of the state search tree. A similar approach is done to create a multilayer RL framework to

a Real Time Strategy (RTS) game (Andersen et al. 2009). The authors decomposed the large state space of the game and built small RL systems in different layers, taking into account the granularity of the strategies to control the behavior of the units in the game. Another possible solution is to combine RL with other techniques. For example, in (Kebriai, Ahmadabadi, and Rahimi-Kian 2014) a hybrid model is applied for the Cournot game. The agent uses a KNN (k-nearest neighbor) predictor to predict the next actions of its rivals and learns the payoff of the state-action pair using RL.

Other technique is Deep Reinforcement Learning (van Hasselt, Guez, and Silver 2016). This method uses a deep neural network to approximate the values of  $Q(s, a, \Theta)$ . The state  $s$  is the input of the neural network, the output is a Q-value for each possible action  $a$  and  $\Theta$  are the parameters of the network. Deep Reinforcement Learning has led to great achievements in the recent history of artificial intelligence, as is the case of AlphaGo, created by DeepMind (Google) (Silver et al. 2016). In 2016, AlphaGo beat 18-time world Go champion Lee Sedol in 4 of the 5 games they played. Until then it was thought that this was a goal which artificial intelligence would still take many years to achieve. Another successful example of Deep Reinforcement Learning implementation is the one applied to classic Atari 2600 computer games, such as Space Invaders, Seaquest, Breakout, Beam Rider, etc., in the Arcade Learning Environment (Mnih et al. 2013). In this case, a Convolutional Neural Network is trained with a Deep Reinforcement Learning algorithm that uses as input the raw pixels and returns as output a value function that estimates future rewards.

In this section some types of algorithms used to model games of various characteristics have been mentioned. This paper focus on the most appropriate ones for both agents that include intelligent behavior. The *Minimax agent* with Alpha-Beta suspension and pruning will present reasoning similar to the human with high experience in the game. The maximum time will be applied to the in-depth search of each of the branches of the root node. This, as well as the depth of the tree, will be parameters to be considered experimentally. The *QLearner agent*, implemented with Reinforcement Learning, will be able to take advantage of the knowledge acquired with the experience to choose the best strategy. Both agents will be described in next section.

## Methodology

This section describes the methodology that will be followed in this paper. It has been divided into two research lines:

- Perfect information experiments: The agents *Random* and *Greedy* will be the rivals of the *Minimax*, in a modification of the game that contemplates perfect information, since, as previously discussed, for the Minimax agent all the tiles must be visible by both



players. Then, a comparison will be made between different instances of the best of the agents (predictably the best will be the Minimax, which takes into account the strategies of an expert human player). Certain parameters will be determined, such as the depth level of the search tree or the maximum time of exploration of the branches. Finally, the author of the game will test the Minimax agent with its best configuration in a series of Minimax vs. Human games.

- Imperfect information experiments: The Reinforcement Learning Agent (called *QLearner*) will also face the Random and Greedy agents, but in this case with the normal version of the *iwoki*, that is, keeping the small tiles hidden from the opponent's view. An evaluation will be made of the efficiency in terms of run time and memory consumption.

In general, the best configuration for the *iwoki* will be experimentally determined. The following methodology will be used: The *iwoki* will be modeled, so that it can represent all the content of the game abstractly. Likewise, an implementation of the methods that the agents will have available in the game is carried out, which will constitute the game space.

Next, the five agents will be developed: one will play randomly (*Random Agent*), another one by the Greedy method (*Greedy Agent*), a third one will do it using a Minimax search algorithm with suspension and Alpha-Beta pruning (*Minimax Agent*), a Reinforcement Learning Agent (*QLearner Agent*) that will take advantage of the experience obtained in previous training games and, finally, an agent that provides an API so that a human can measure the effectiveness of the previous ones (*Human Agent*).

Once the implementation is completed, the agents will play against each other in two-player games. Each game may have a history of each one of the moves that the agents make in their turns. This will allow to make an exhaustive pursuit of the evolution of the game and even to be able to make a simulation in the physical version of the *iwoki*. This way it will be possible to contrast the actions of the agents with what a human would have decided to do in each turn. The number of games played by the agents will be enough to establish a series of firm conclusions regarding the nature of the agents. The new insights provided in all the previous steps will allow identifying new ways to go deeper into the study of the implemented solutions, as well as to establish new research lines.

### **Agent Description**

Once the components of the game space have been implemented, the detail of the agents is described as follows.

#### **Random Agent**

This agent chooses its moves randomly. It gets from the game space all the possible actions to do in its turn and chooses randomly one of them, without taking into account how good or bad the move is.

### Greedy Agent

This agent emulates the playfulness of a human novice. It implements the simple strategy of choosing the option that gives it the most points immediately, regardless of how easy the move benefits its opponent. It gets from the game space all the possible actions to take in its turn. Then it selects the best one, taking into account the previous strategy. In case of different options that give it the same points, its preference will be the following one:

- (1) Place a small tile on the subtraction side. This increases the white token number, which is good for getting a hexagonal tile.
- (2) Place a small tile on the addition side. In any case, placing a small tile brings the player closer to the end of the game.
- (1) Place a hexagonal tile.
  - (1) Take a tile. This does not give it any points, so it only ultimately chooses to take this action. Obviously, if it has just taken a tile, the option will be to pass the turn to the other player, as specified by the *iwoki* rules.

### Minimax Agent

This agent applies the strategies that a highly experienced human player has. It has to anticipate its opponent's moves to know the possible future states of the game, from the state it is in, to an horizon (depth). To do so, in each turn it must generate a Minimax search tree with Alpha-Beta pruning and suspension in depth and time. To get the best move in the current turn:

1. The `alpha_beta` function is invoked, that is executed in a recursive way for the Max and Min nodes, from the root node to the leaf nodes. As it is explored, new nodes are generated following these steps:
  - (a) It gets from the game space all the possible actions to be carried out.
  - (b) For each of them, a new lower level node is created, invoking a method that:
    - Makes a clone of the game space.
    - Executes the action on the new game space. This is the new game state associated to the generated node.
2. The tree is branched out recursively until a leaf node is found in two different ways:
  - Reaching the established depth level.
  - Reaching the end of the game in the corresponding branch without having reached the depth level.

The utility function is found by subtracting the scores after the final point tally.

3. Undoing the recursion updates the values of Alpha and Beta until the root node is given a value. At the same time, it allows pruning the branches that will not present better values than those found so far.

The maximum time for deep searching along each of the main branches of the root node is determined. For example, for a game state where the Minimax agent has 6 possible actions and the maximum time set is 10 seconds, if any of those 6 branches has not been fully explored in 10 seconds, it will propagate the best option found so far and follow it by the next branch of the root node. This limits the total search in that turn to a maximum of 60 seconds. As it is a search with suspension, the algorithm does not guarantee to find the best action in each turn, but establishing some suitable parameters of time and depth, as it will be seen in the following sections, the results will be able to be satisfactory.

### **QLearner Agent**

This agent is based on training by playing numerous games. The more it trains, the more it can be expected to perform against its opponents. The way to choose the best option in each turn is the following:

1. The game space provides it with all the possible actions  $a$  in the current state  $s$ .

2. In order to determine the action to be performed:

(a) It obtains from the Q-table the previously stored Q-values ( $s, a$ ) values. Those that do not exist are initialized with a hyperparameter.

(b) In exploration mode, the action to be performed is selected from a weighted random function on the Q-values. These take value previously by the *softmax* method, which obtains a probability distribution. Thus, the values are between 0 and 1 and the sum of all of them is 1.

(c) In exploitation mode, the agent simply selects the action that has the highest Q-value. In case of having several actions with this maximum value, one is taken randomly.

3. The selected action is executed. Each movement in the course of the game is stored in an action-state value list.

4. At the end of the game:

(a) The value of the reward is calculated as the difference between the score achieved by the *QLearner* agent and its opponent, after the final count tally.

(b) The Bellman equation is then applied in an iterative manner to the previous list of action-state values, traced in the reverse direction (that is, first the action-state that has brought to the end of the game and then the rest until the action-state of the beginning). Thus, all Q-values of the movements performed in the game are obtained.

(c) The Q-values are stored in the Q-table.

5. After the execution of the number of games that constitutes the training of the agent, the Q-table is stored in a file in order to reuse the learning in future executions.

## Human Agent

It implements an interface that allows a human player to face any of the other agents.

## Experimental Setup

Each of the items in the experimental phase is explained as follows. Firstly, for the correct interpretation of the results, a 100-game match between the Greedy and Random agents is launched. This way we can have a reference of how good the first one is with respect to the second one when other agents compete against both.

## Perfect Information Experiments

In this experiment, a modification of the *iwoki* will be used in order to make it deterministic and information perfect. To this purpose, the small and hexagonal tiles that have not yet been drawn will not be hidden. This way the agent will know in advance which tile will draw if it chooses this action in its turn and, in the same way, it will also know which tile its opponent could draw in its case. Thus, the agent is deterministic, eliminating completely the randomness once the tiles have been drawn and the game is started. Also, the players' small tiles, as well as the hexagonal ones, will remain visible at all times. Thus, the game will be one of perfect information.

Minimax will play games against Random and Greedy agents. It will use different configurations according to the hyperparameters used, which are:

- Search tree depth: 1, 2, 3 and 4 level values will be used. For depth 1, 2 and 3, 100 games will be executed, which will allow evaluating the results and drawing certain conclusions. Due to the high computational cost, only 10 games will be executed when depth level 4 is applied.
- Maximum search time on each of the main branches of the root node (*maxTime*): A range of values from 0.5 to 90 seconds will be applied. As it will be seen in the next section, it only makes sense to apply this hyperparameter in games against the Greedy agent under *depth* = 3.

After playing all of the above games, the two Minimax settings that are likely to be optimal for playing against a human player will be determined so as to balance their effectiveness with the time it takes to execute their turn. Minimax agent will play 20 games against itself applying these two settings to determine definitively which one is the most appropriate. This will be the one used by the agent to play 5 games against the creator of the *iwoki*.

### **Imperfect Information Experiments**

Initially, 10,000 games will be launched between *QLearner* and Random and Greedy agents in order to establish a reference of how they behave before any training.

Two sub-experiments with *QLearner* will follow. In each of them there will be two phases:

1. In a first step, the model is trained to acquire as much experience as possible. The training aims to give it the knowledge of the best policy to apply in order to determine what action it should take in each state to get the best score at the end of the game. The *QLearner* training will be done with games in exploration mode against the Random agent, because it is the one that explores all the possible actions in each state. If it did it against the Greedy, its training would not be so exhaustive, since it would only learn from the opponent the best actions in the short term.
2. Once the agent has been trained, the next phase is to put its experience into practice. Now the *QLearner* games will be in exploitation mode. The agent will choose in each state the action that will provide him with more benefit in the long term, as it will have learned in the previous training.

In the first sub-experiment, a batch of 100,000 games will be executed in exploration mode between the *QLearner* and Random agents. Once the training is finished, 50,000 games will be played against Random and Greedy agents.

For the next sub-experiment, it is opted to modify the *iwoki* in order to dramatically reduce the space of possible states. The small and hexagonal tiles will be randomly drawn at the beginning, but a seed is set up so that they will always be the same in all the games played. Those drawn throughout the game will be randomly seedless. As in the previous experiment, the *QLearner* agent is trained with 100,000 *QLearner* vs. Random games and then the same matches are performed (50,000 *QLearner* vs. Random and 50,000 *QLearner* vs. Greedy).

With regard to the hyperparameters, a Cartesian Grid Search (that is, all possible combinations) With regard to the hyperparameters, a Cartesian Grid Search (that is, all possible combinations) is performed with Learning Rate ( $\alpha$ ) and Discount Factor ( $\gamma$ ) in order to establish their appropriate values. The first step is to assign them values divisors of 10 (1, 0.1, 0.01, 0.001 and 0.0001) and execute a batch of 5,000 games for every pair. In each case, it is observed is performed with Learning Rate ( $\alpha$ ) and Discount Factor ( $\gamma$ ) in order to establish their appropriate values. The first step is to assign them values divisors of 10 (1, 0.1, 0.01, 0.001 and 0.0001) and execute a batch of 5,000 games for every pair. In each case, it is observed how many Q-values are generated throughout the games and what percentage of them are updated with a new different value from the initial one. It can be seen that the smaller  $\alpha$  and  $\gamma$  become, the lower the number of Q-values generated is (it ranges around 140,000–150,000) and

the percentage of updated Q-values increases (it ranges around 30%-44%). Not only these values are relevant but also how the evolution is. That is, how close to a state of convergence the evolution of the updated Q-values is after all those 5,000 games. The conceptual meaning of this approach is that small values of  $\alpha$  and  $\gamma$  imply that the system learns less, but the policy learned in each case seems to be more optimized. In contrast, the higher  $\alpha$  and  $\gamma$  are, the more the agent will learn but with a not so optimized policy. For *QLearner's* training, the aim is to obtain as many Q-values as possible, without the rate of updated Q-values being too low. It is relevant that  $\alpha$  has much more influence on the results than  $\gamma$ . Taking into account that the final experiments will reach 100,000 games in exploration mode and another 50,000 in exploitation mode, it is concluded that the most appropriate range of values for both hyperparameters is between 0.1 and 1. A new Cartesian study is performed with new batches of 5,000 games to adjust hyperparameters to the optimal values. The final setup is a Learning Rate ( $\alpha$ ) of 0.4 and a Discount Factor ( $\gamma$ ) of 0.8 for all cases.

## Results

First of all, agents Greedy and Random were faced off in 100,000 games. The results shown that the Greedy Agent completely outperformed the Random one by winning all the times, taking an average of 0.16 seconds to finish and an average difference score of 9.73 points.

## Minimax Analysis

This subsection describes the results of the Minimax agents against the other agents, as previously explained. [Table 1](#) summarizes the results of confronting the Minimax agent vs the Greedy and Random ones.

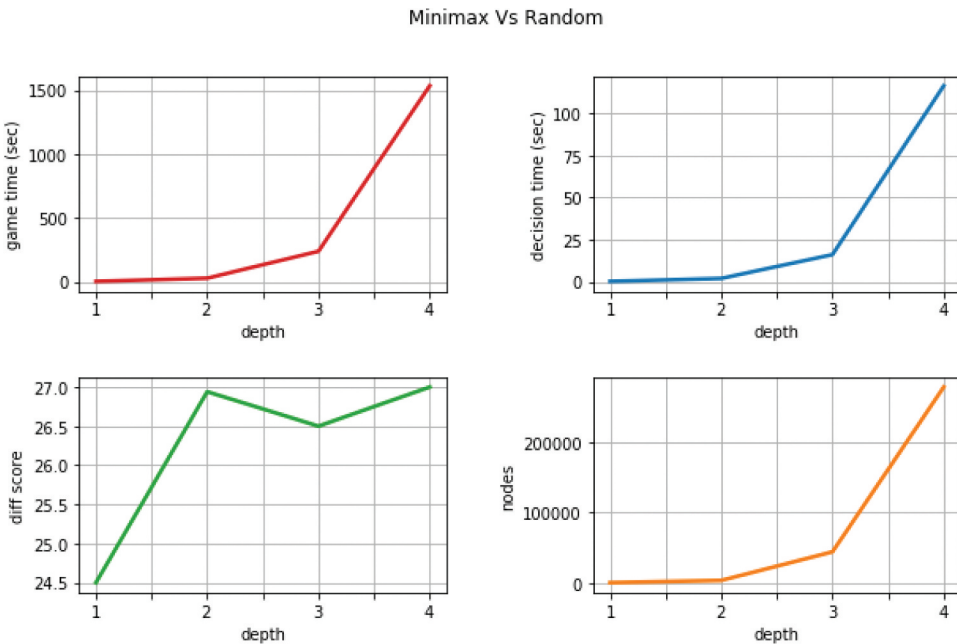
The results shown in [Table 1](#) provide a series of general conclusions. On one hand, the average number of turns is irrelevant, since in all cases are similar and they are independent of the specified parameters. In addition, it is observed that whatever the value of the hyperparameter *depth* is, the Minimax agent always beats the Random one with a big difference of scores. We can conclude that Random is far below the level of Minimax. As expected, Minimax games with *depth* = 1 against Greedy are very balanced. We can see that the number of wins of each one is very similar (47 to 49) and that there is only 0.7 points difference on average. It is confirmed in an experimental way something that was known on a theoretical level: that the Minimax search using a tree of *depth* = 1 is equivalent to the Greedy search.

The average values while the depth increases can be seen in [Figure 1](#).

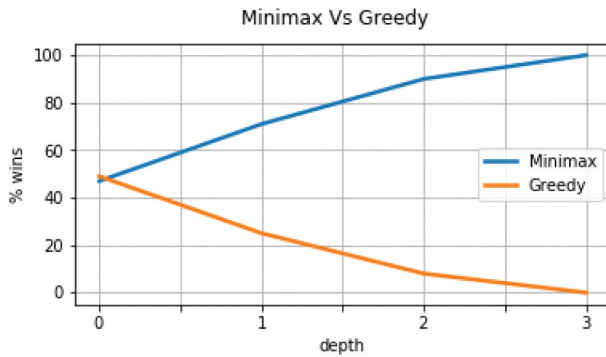
It is worth emphasizing the information seen in [Figure 2](#) about the evolution of the games won by Minimax versus Greedy, while increasing the search

**Table 1.** Results for Minimax. An overline over a column header indicates average values.

Depth	maxTime	#Games	Wins	Losses	Draws	<u>Time (sec)</u>	<u>Dif.Scores</u>	<u>Turns</u>	<u>Decisiontime</u>	<u>Nodes</u>
Minimax Vs Random										
1	–	100	100	0	0	2.97	24.5	13	0.22	339
2	–	100	100	0	0	27.73	26.94	14	1.97	3.498
3	–	100	100	0	0	238.8	26.5	14	16.13	44.2
4	–	10	10	0	0	1534.8	27.2	13	116.4	278.857
Minimax Vs Greedy										
1	–	100	47	49	4	3.87	0.7	12	0.3	455
2	–	100	71	25	4	44.81	5.29	12	3.54	6.315
3	–	100	90	8	2	699	7.5	13	50.27	140.505
4	–	10	10	0	0	2907	9.67	12	188.4	548.321
Minimax ( <i>depth</i> = 3) Vs Greedy										
3	0.5	100	46	49	5	9.81	–1.05	12	0.77	1.719
3	1	100	41	57	2	15.2	–1.88	12	1.18	2.8
3	5	100	49	43	8	58.39	0.69	13	4.43	11.563
3	10	100	48	52	0	113.4	–0.82	13	8.52	13.545
3	20	100	60	28	12	192.6	1.36	13	13.96	37.979
3	30	100	60	28	12	264.6	3.28	13	19.7	53.063
3	40	100	56	32	12	361.8	5.2	13	27.67	51.473
3	50	100	76	20	4	403.2	7.8	13	29.9	88.872
3	60	100	60	38	2	475.8	6.94	14	33.52	97.015
3	70	100	82	16	2	480	9.17	13	36.67	97.429
3	80	100	80	19	1	570.6	10.7	14	39.89	118.672
3	90	100	95	1	4	591	16.6	13	44.43	122.166

**Figure 1.** Average values in Minimax vs. Random games by increasing *depth*.

depth. With *depths* of 1, 2, 3 and 4, Minimax wins 47%, 71%, 90% and 100% of games, respectively. The difference in scores between the two grows in a linear trend, as can be seen in Figure 3. As a counterpart, increasing the depth means increasing exponentially the number of nodes generated in the search tree and, therefore, the time that the agent takes to decide the action to be carried out.



**Figure 2.** Minimax wins against Greedy by increasing *depth*.



**Figure 3.** Average values in Minimax vs. Greedy games when increasing *depth*.

This can be seen graphically in [Figure 3](#). It is confirmed that the complexity of the Minimax algorithm exponentially grows in time and space as the depth of the search tree increases.

Focused on a match between the best Minimax configuration and a human, *depth* = 4 is a value that implies too much time waiting for Minimax to decide the best move, even though the winning percentage against Greedy is 100%. That is the reason why we have chosen to run Minimax games with different *maxTime* values against Greedy, only with *depth* = 3. The decision time to be improved is 50.27 seconds (Minimax vs. Greedy with *depth* = 3), without losing the effectiveness of more than 90% of won games (9 wins and 2 draws).



The average values follow a linear growth trend as the *maxTime* parameter is increased, as shown in Figure 4.

In Table 1 it can be seen that when *maxTime* = 90 seconds the maximum percentage of wins of Minimax versus Greedy is reached and the time it takes the agent to decide its move in each turn is 44.43 seconds. That is, the time is reduced by 11.62%. While the experiment seems to have improved the win percentage, a much larger number of games would conclude that *maxTime* = 90 seconds would improve the time but would never become more effective in terms of win percentage, since unlimited time explores the entire tree down to depth level 3 through all the branches of the root node. Therefore, it seems that the optimal configuration of the Minimax agent to play against a human, balancing efficiency with time, is *depth* = 3 and *maxTime* = 90 seconds.

Thus, two different Minimax agent instances were also performed. The parameters were set to:

- Minimax agent 1: *depth* = 3 and unlimited timeout.
- Minimax agent 2: *depth* = 3 and *maxTime* = 90 seconds.

The result of the 20 games were 10 victories for Minimax agent 1 and 2 draws, with an average duration of 22.19 minutes and an average difference of score of 2.50. The average duration is close to the sum of the value of Minimax vs. Greedy (11.65 minutes) and the one using *maxTime* = 90 seconds (9.85 minutes).

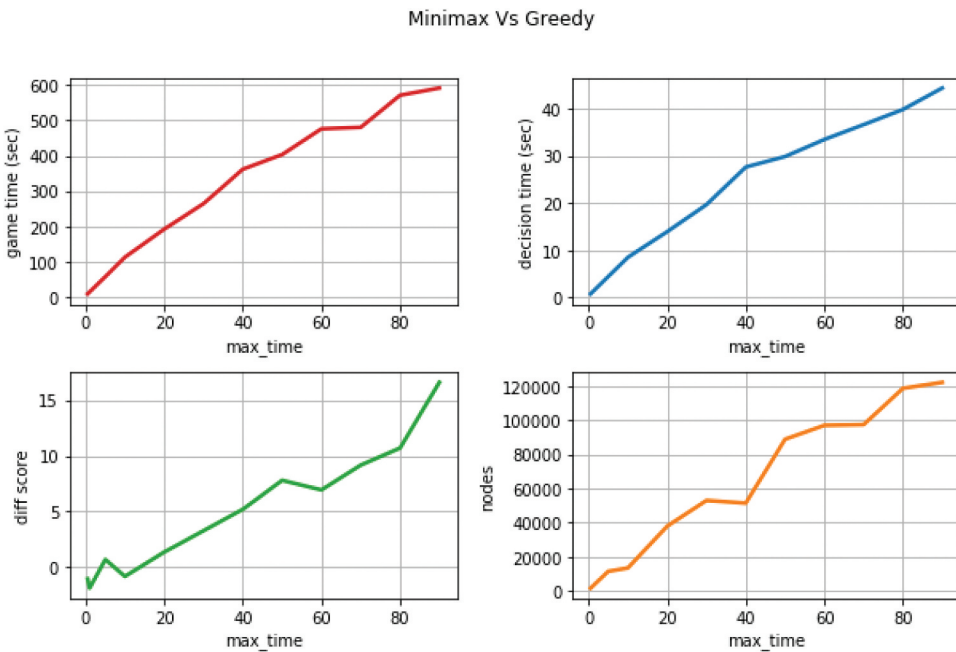


Figure 4. Average values in games Minimax vs. Greedy by increasing *maxTime*.

**Table 2.** Final Scores of Minimax vs. Human games.

Minimax	Human	
Game 1	48	43
Game 2	51	47
Game 3	59	54
Game 4	51	45
Game 5	65	62

Finally, five games were played between the *iwoki*'s author and the Minimax Agent 2. The results can be seen in Table 2. Throughout the game, the Minimax agent determines its movement once it gets the depth established as a parameter ( $depth = 3$ ) or when it reaches the maximum time in any of the branches of the root node ( $maxTime = 90$  seconds). Minimax tends to take the lead and avoids at all costs that the human places hexagonal tiles, maintaining higher score than the human at all times. However, the human, who is able to play with an expert strategy, can keep a close scoring difference. When reaching the last turns of the game, the Minimax movement is determined when the algorithm gets to the leaf node before reaching the depth limit and before spending the established time limit. This implies that the Minimax's decision will undoubtedly be the optimal one. It anticipates the human and disrupts any strategy he tries to implement. That is the reason why Minimax is always better than the human, no matter how expert he is. It is worth mentioning that if the human player had not been the creator of the *iwoki*, the scoring difference between the two players would have been greater in favor of Minimax.

### Reinforcement Learning Analysis

The results of the batches of the different sub-experiments specified in the Section 4.2 are shown in Table 3.

The first 10,000 games of *QLearner* vs. Random and *QLearner* vs. Greedy show how the *QLearner* agent behaves in exploration mode, without any previous training. It is necessary to explain that the total Q-values are all the tuples (state, action) that are generated in all the games. That is, there will be a Q-value for each of the possible actions in each state (game turn) of each of the games played, as long as it has not been generated before for this state-action, while the updated Q-values are those that, having been previously generated, are updated with a different value from the initialization one.

According to the results of Table 3, and following the metrics illustrated in the figures of the previous section, a series of conclusions can be drawn. Firstly, the average number of turns and the average duration of the games are not relevant, since their values are very similar in any of the typologies of the

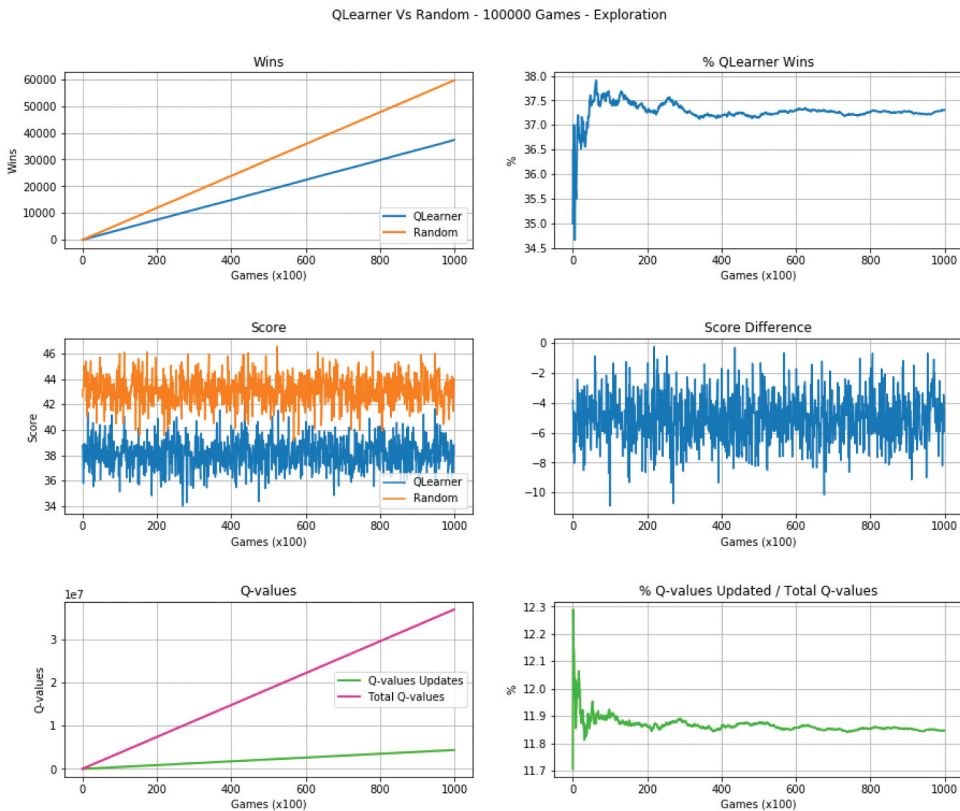
**Table 3.** Results for Reinforcement Learning Experiments. An overline over a column header indicates average values. The type of the games played are in Exploration (R) or in Exploitation (T) modes.

Versus	Type	#Games	Qlearner	Rival	Draws	$\overline{Dif. Scores}$	$\overline{Time}$ (secs)	$\overline{Turns}$	Total Q-values	Updated Q-values	Q-file Size
Random	R	10000	3706	5996	298	-5,11	0,21	13	3686206	437932	465.40 MB
Greedy	R	10000	24	9970	6	-32,68	0,19	12	4266728	453224	540.51 MB
Tiles randomly drawn at the beginning											
Random	R	100000	37305	59536	3159	-5,02	0,24	13	36967241	4379700	4,652.88 MB
Random	T	50000	18671	29829	1500	-5,07	0,25	14	55442209	2185249	-
Greedy	T	50000	112	49841	47	-32,64	0,24	13	58167883	2276841	-
Same tiles drawn at the beginning											
Random	R	100000	49532	48205	2263	-0,2	0,2	13	25965300	9767824	3,238.99 MB
Random	T	50000	22777	26208	1015	-2,62	0,24	13	39591285	4625747	-
Greedy	T	50000	41607	8386	7	-3,6	0,21	12	28707442	16570374	-

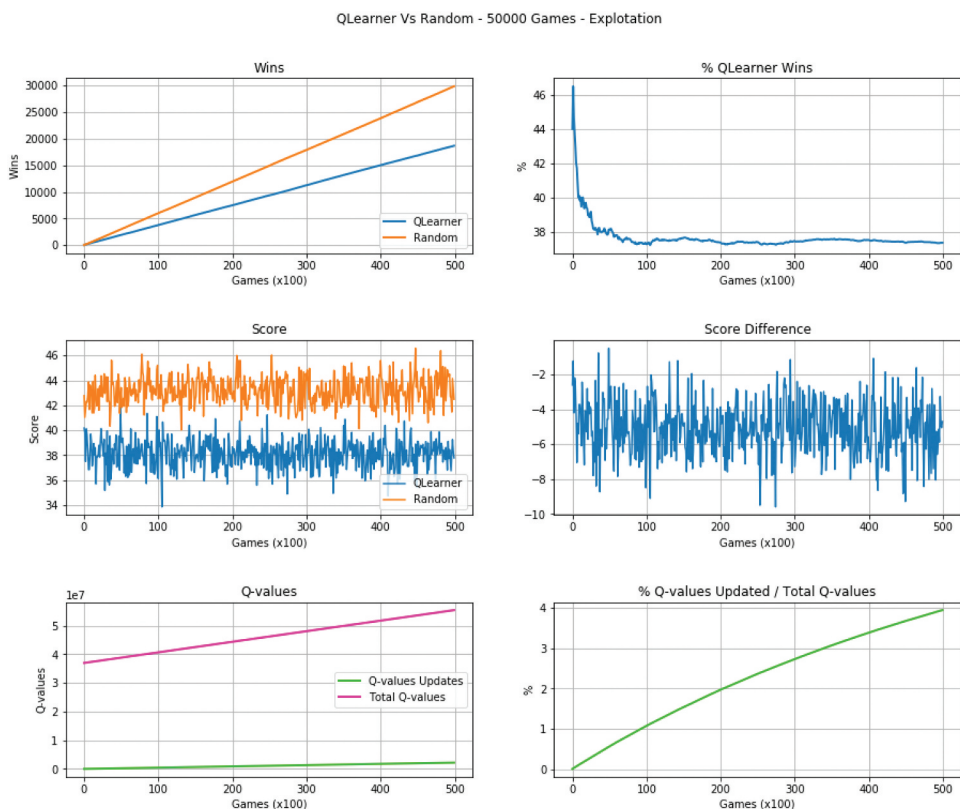
games and for all the agents. In addition, the 10,000 initial games that the *QLearner* agent plays against Random do not present relevant information either. We can see that the percentage of games won by the first one is around 37% of the total, with a mean difference in scores that is not big at all (5.11). However, the other instances executed against agent Greedy show the low effectiveness of the *QLearner* when it has not yet been trained, as it only wins 24 of the 10,000 games (0.24%) and the mean difference in scores is 32.68.

Concerning the first of the sub-experiments, *QLearner* training is an exercise similar to the first 10,000 games but with a higher volume. [Figure 5](#) shows that the trend in terms of wins and score difference is the same. However, we can find relevance in the number of Q-values generated (about 37 million), being only 11,85% those updated along the 100.000 games. Once trained with exploration instances, the behavior of the *QLearner* agent in exploitation mode maintains the trend: the games against Random agent follow the same line as in the training, with no relevant changes. This can be seen in [Figure 6](#).

Regarding the matches against Greedy, there is no improvement compared to the initial 10,000 games, as it only manages to win 112 of the 100,000, (0.22%) and the difference in scores is maintained. [Figure 7](#) graphically



**Figure 5.** Metrics of the first training of *QLearner* vs. Random.



**Figure 6.** *QLearner* vs. Random metrics after the first training.

confirms this. The learning is nearly 37 million of Q-values, which implies a 4.54 GB Q-file, plus about 50% more Q-values in the exploitation phase. Despite this, no signs of improvement are found in either Random or Greedy matches. During the experiment it has been tried to increase the number of training games, but the consumption of computational resources makes it unfeasible to obtain good results.

In order to reach some interesting conclusions in this work, the previously mentioned modification of the *iwoki* is put into practice, which consists of always drawing the same tiles at the beginning of the games. With this measure the space of possible states is reduced considerably. The second sub-experiment adopts this modification and from now on the results take another direction. The Table 3 shows that throughout the 100,000 training games there is a negligible difference between *QLearner* and Random. Practically 50% of wins and a mean difference in scores that can be considered insignificant (0.20). 25,965,300 Q-values are generated, 29.76% less than in the first experiment, and the percentage of updated Q-values remains around 39% of the totals. Graphically, the evolution can be seen in Figure 8. After such training, 50,000 instances of exploitation are launched against the two opponents, raising some new considerations.

QLearner Vs Greedy - 50000 Games - Exploitation

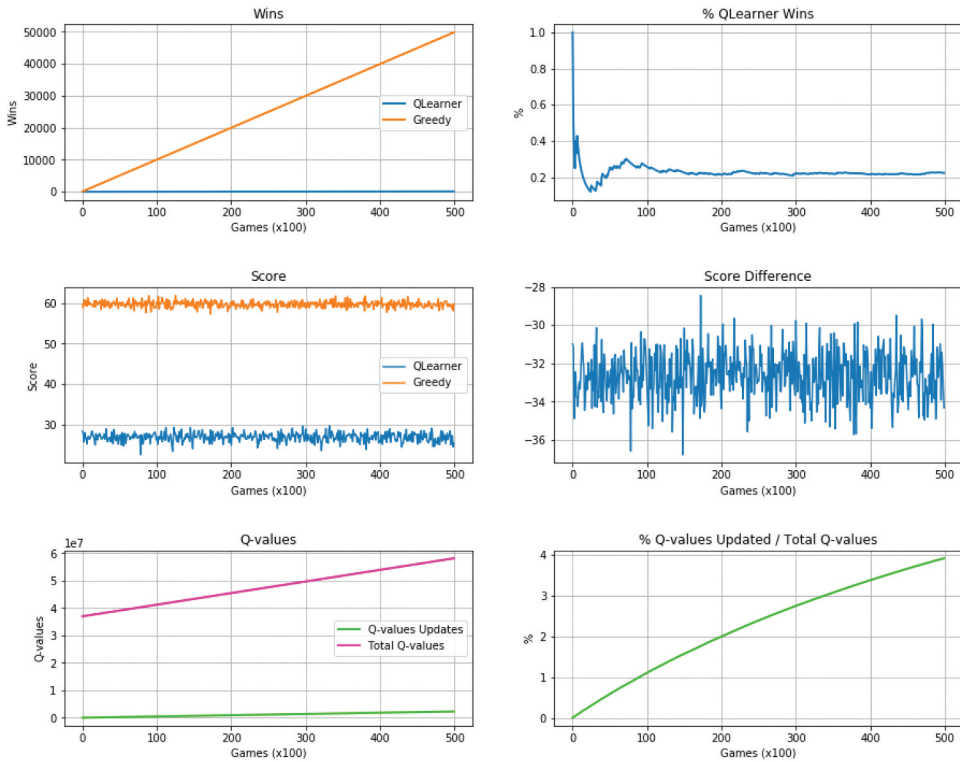
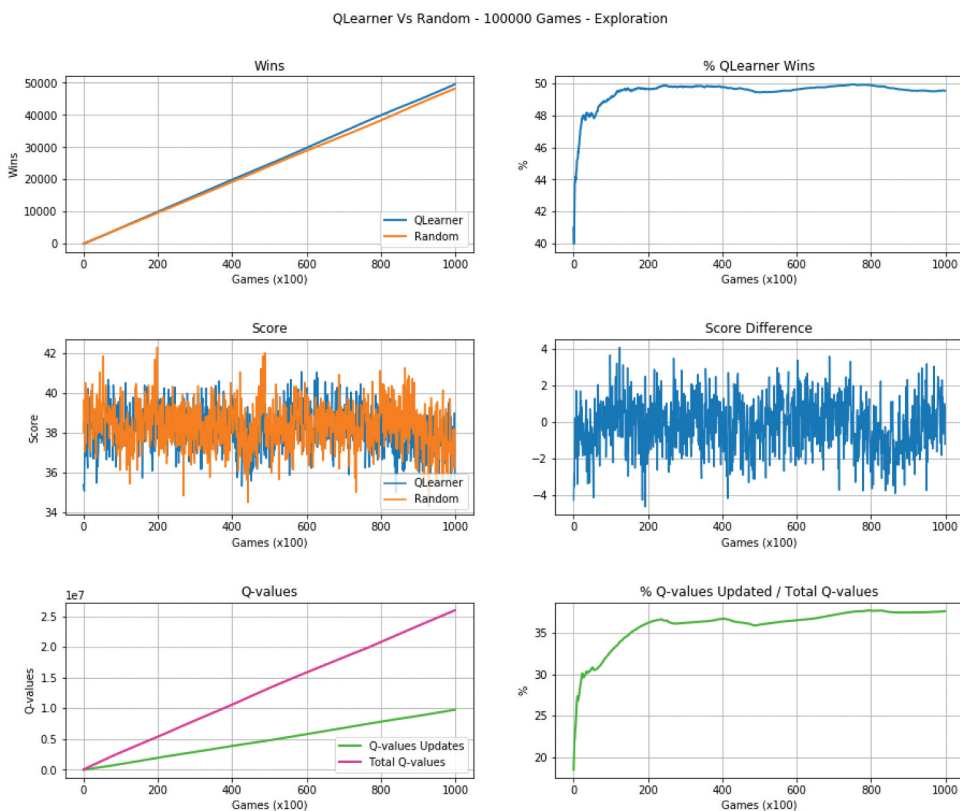


Figure 7. Q-Learner vs. Greedy metrics after the first training.

The games against Random agent find little difference in their reference values. Therefore, the change from exploration to exploitation does not imply a significant alteration in the evolution of these values. The graphs in Figure 9 confirm this trend. When playing against Greedy the result is different. The graphs in Figure 10 show that the performance of the Q-Learner agent has improved to the point that after 16,200 games, it always wins. It controls Greedy's actions, which always chooses the best move in the short term on every turn, to learn the policy it should follow to get the most cumulative score.

### Conclusions

In the course of this work, *iwoki* and its characteristics have been made known, which has made it possible to model and implement its components has been carried out. Based on the study of the state of the art, we have defined which agents would be part of the experiments, according to the nature of the *iwoki*, and we have proceeded to develop them. During the experimental phase, a series of games have been carried out that have put the different agents into competition. It has been determined how good the Minimax agent is with

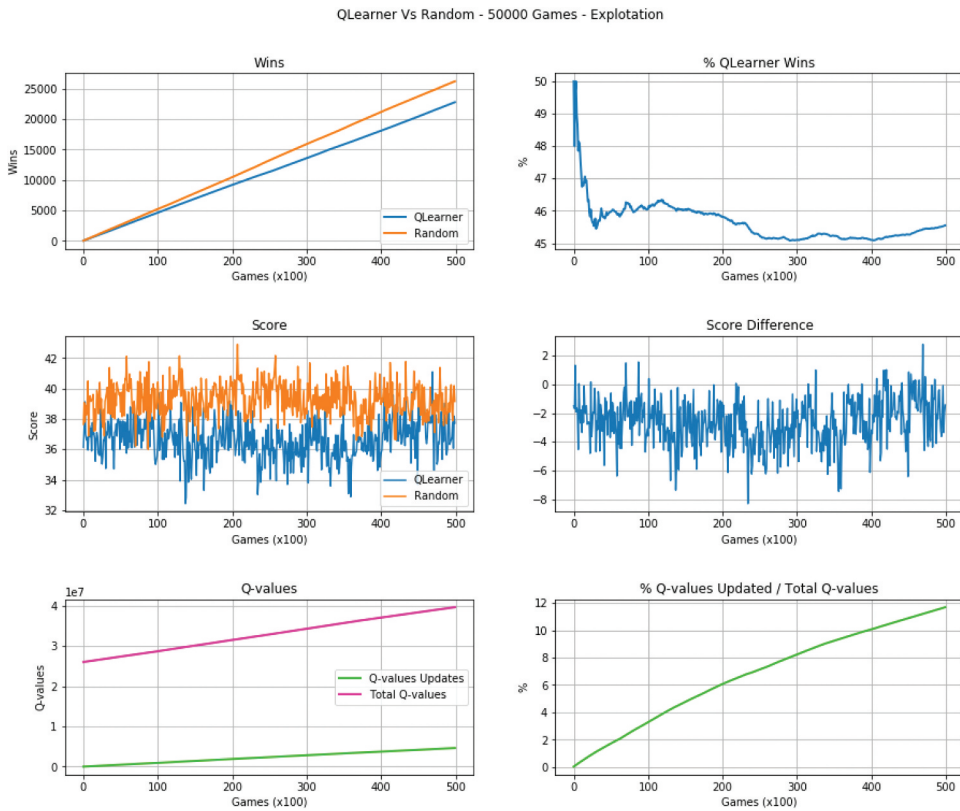


**Figure 8.** Metrics of the second training of *QLearner* vs. Random.

respect to Random and Greedy. We have also been able to observe the *QLearner* agent's ability to learn by training it with Random and then demonstrating its acquired knowledge against Greedy. Finally, we can see how the Minimax agent was able to beat the creator of the *iwoki*, who at the same time, is also the creator of the winning algorithm.

The results obtained cover the twofold purpose set out at the beginning of the work. The first one is to find the appropriate implementation for the agents of the two developed research lines, Minimax and Reinforcement Learning, and to provide them with the optimal configuration according to the experimental results. Moreover, we expect to present the *iwoki* as a new contribution to the field of research in artificial intelligence applied to games, establishing a starting point from which to develop a diversity of agents with different technologies. The source code of the simulator, and all the implemented agents described in this work, are available in Github: <https://github.com/videgain/iwoki>.

Consequently, some lines of future work are identified as a result of what is presented in this paper. There is the possibility of implementing the different variants of Alpha-Beta pruning (Knuth and Moore 1975) available in the



**Figure 9.** QLearner vs. Random metrics after the second training.

literature. But also, since the original *iwoki* is of imperfect information, as the small tiles are not visible to the rest of the players, it could be developed as a Bayesian game. This way it can be taken into account the probability that an opponent has certain tiles, knowing the number of times they have been played previously. Moreover, it would be appropriate to develop a Monte Carlo Tree Search method (Chaslot et al. 2008), or even more advanced agents, such ones used with Deep Reinforcement Learning (van Hasselt, Guez, and Silver 2016) or agents optimized using Evolutionary Algorithms (Fernández-Ares et al. 2017). This opens the door to a new line of research to study this specific case and model a neuronal network that will allow the problem of *iwoki* to be treated. Other possible lines of future work are to study if it is feasible to implement a Recurrent Neural Network (RNN) (Deng, Zhang, and Shu 2018) for a new Vanilla agent. The RNN would be similar to the one used for language models when obtaining a sequence of words and returning a probability distribution of the next word. The hidden state includes the text seen so far at each time step.

Finally, it would be interesting to establish a graphic environment that makes it easier for a human player to execute his/her moves without having to reproduce the moves with the physical version of the game, and also to



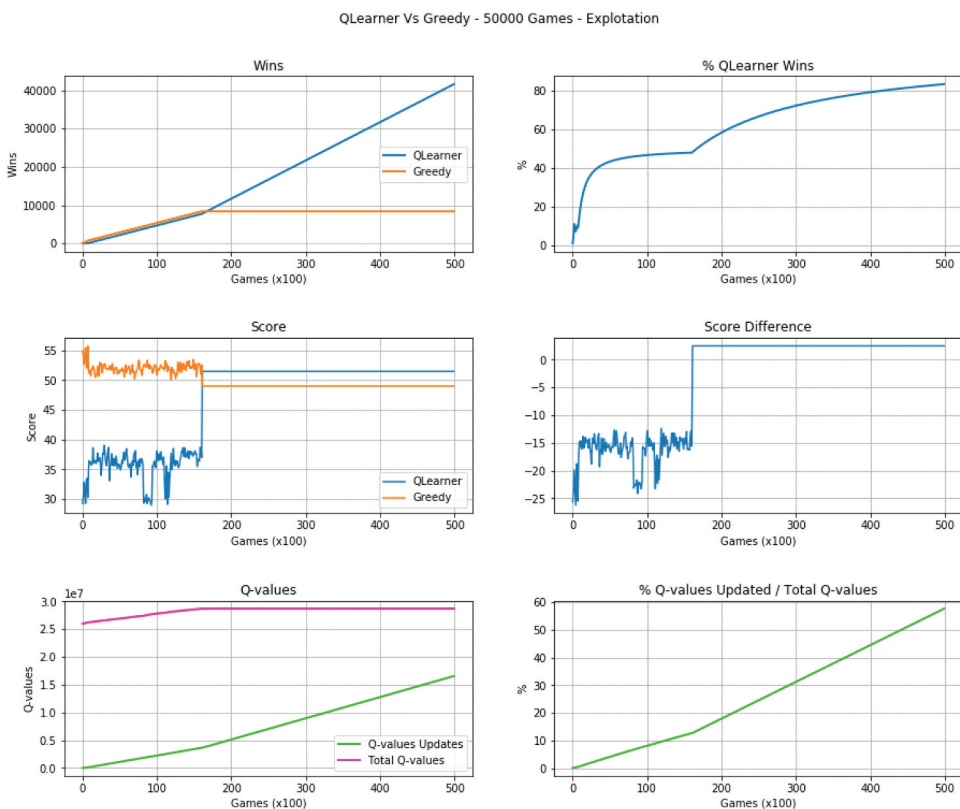


Figure 10. *QLearner* vs. Greedy metrics after the second training.

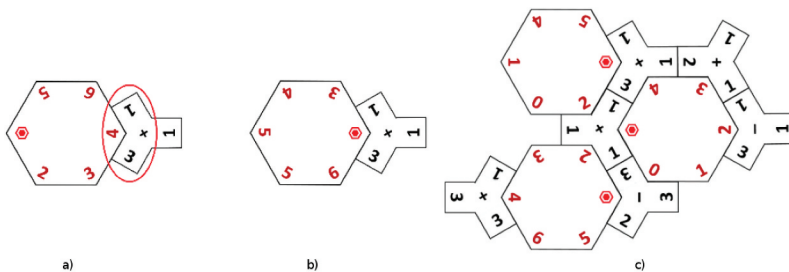
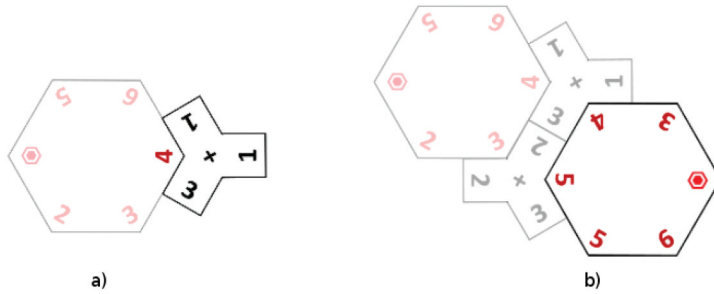


Figure 11. Examples of tile placement. a) Correct placement: the small tile is placed on numbers. b) Incorrect placement: the small tile does not match with any other number, and fits only on the wildcard. c) Correct placement: in each case, the small tiles covering the wildcards also cover a red number from another hexagonal tile.

extended the game to up to 6 players, just like the original *iwoki*.

### Acknowledgments

This work has been partially funded by project TIN2017-85727-C4-2-P (Spanish Ministry of Economy, Industry and Competitiveness). The first author would like to thank his wife and



**Figure 12.** Examples of score calculation. A) Small tile placed: 4 points added to the score. b) Hexagonal tile placed: 9 (4 + 5) points are added to the score.

daughters for their support, especially his youngest daughter, who inspired him to invent this game as a gift for her First Communion.

## Funding

This work was supported by the Ministerio de Economía, Industria y Competitividad, Gobierno de España [TIN2017-85727-C4-2-P].

## ORCID

Santiago Videgáin  <http://orcid.org/0000-0002-8072-859X>

Pablo García Sánchez  <http://orcid.org/0000-0003-4644-2894>

## References

- Andersen, K. T., Y. Zeng, D. D. Christensen, and D. Tran. 2009. Experiments with online reinforcement learning in real-time strategy games. *Applied Artificial Intelligence* 23 (9):855–71. doi:10.1080/08839510903246526.
- Bai, A., F. Wu, and X. Chen. 2015. Online planning for large markov decision processes with hierarchical decomposition. *ACM Transactions on Intelligent Systems and Technology*. 45:1–45. 6 (4):28. doi:10.1145/2717316.
- Chaslot, G., S. Bakkes, I. Szita, and P. Spronck. 2008. Monte-carlo tree search: A new framework for game AI. In *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference, October 22-24, 2008, Stanford*, ed. C. Darken and M. Mateas. California: USA. The AAAI Press, pp. 216–217.
- Daskalakis, C., P. W. Goldberg, and C. H. Papadimitriou. 2009. The complexity of computing a nash equilibrium. *SIAM Journal on Computing* 39 (1):195–259. doi:10.1137/070699652.
- Deng, H., L. Zhang, and X. Shu. 2018. Feature memory-based deep recurrent neural network for language modeling. *Applied Soft Computing* 68:432–46. doi:10.1016/j.asoc.2018.03.040.
- Fernández-Ares, A., A. M. García, P. García-Sánchez, P. A. Castillo, and J. J. M. Guervós. 2017. Analysing the influence of the fitness function on genetically programmed bots for a

- real-time strategy game. *Entertainment Computing* 18:15–29. doi:10.1016/j.entcom.2016.08.001.
- García-Sánchez, P., A. P. Tonda, A. J. F. Leiva, and C. Cotta. 2020. Optimizing hearthstone agents using an evolutionary algorithm. *Knowledge-based Systems* 188: doi: 10.1016/j.knosys.2019.105032.
- Harsanyi, J. C. 1967. Games with incomplete information played by “bayesian” players, i–iii part i. the basic model. *Management Science* 14 (3):159–82. doi:10.1287/mnsc.14.3.159.
- Kaelbling, L. P., M. L. Littman, and A. W. Moore. 1996. Reinforcement learning: A survey. *The Journal of Artificial Intelligence Research* 4:237–85. doi:10.1613/jair.301.
- Kalles, D., and P. Kanellopoulos (2008). A minimax tutor for learning to play a board game. In *Proceedings of the AI in Games Workshop, 18 th European Conference on Artificial Intelligence, Patras, Greece*, pp. 10–14.
- Kebriaei, H., M. N. Ahmadabadi, and A. Rahimi-Kian. 2014. Simultaneous state estimation and learning in repeated cournot games. *Applied Artificial Intelligence* 28 (1):66–89. doi:10.1080/08839514.2014.862774.
- Kemeny, J. G., and J. L. Snel. 1960. *Finite Markov Chains*, 24–38. Chapter 2 ed. Princeton, N.J.: Van Nostrand.
- Knuth, D. E., and R. W. Moore. 1975. An analysis of alpha-beta pruning. *Artificial Intelligence* 6 (4):293–326. doi:10.1016/0004-3702(75)90019-3.
- Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. 2013. *Playing atari with deep reinforcement learning*, vol. abs/1312.5602. CoRR.
- Osborne, M. J.. 2004. *An introduction to game theory*, vol. 3. New York: Oxford university press
- Pearl, J. 1980. SCOUT: A simple game-searching algorithm with proven optimal properties. In *Proceedings of the 1st Annual National Conference on Artificial Intelligence, Stanford University, CA, USA, August 18-21, 1980*, ed. R. Balzer, 143–45. AAAI Press/MIT Press, Cambridge, USA.
- Russell, S. J., and P. Norvig. 2010. *Artificial Intelligence - A Modern Approach. Third ed.* Pearson Education, London, England.
- Silver, D., A. Huang, C. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al.. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529 (7587):484–89. doi:10.1038/nature16961.
- Song, X. 2012. Experience generation in tic-tac-toe for general game learning. L. Barolli, F. Xhafa, S. Vitabile, and M. Uehara. ed., *Sixth International Conference on Complex, Intelligent, and Software Intensive Systems, CISIS 2012*, 1–5. Palermo, Italy: IEEE Computer Society. July 4-6, 2012
- Sutton, R. S., and A. G. Barto. 2018. *Reinforcement learning: An introduction*. MIT press, Cambridge, USA.
- Turing, A. M. 1953. Digital computers applied to games. In *faster than thought*, ed. B. V. Bowden, 286–310. London, England: Sir Isaac Pitman & Sons LTD.
- van Hasselt, H., A. Guez, and D. Silver. 2016. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12- 17, 2016*, ed. D. Schuurmans and M. P. Wellman, 2094–100. Phoenix, Arizona: USA.
- Von Neumann, J., and O. Morgenstern. 2007. *Theory of games and economic behavior (commemorative edition)*. Princeton university press, New Jersey, USA.
- Yannakakis, G. N., and J. Togelius. 2018. *Playing Games, Chapter 3*, 91–150. Cham: Springer International Publishing.

Zagal, J. P., J. Rick, and I. Hsi. 2006. Collaborative games: Lessons learned from board games. *Simulation & Gaming* 37 (1):24–40. doi:10.1177/1046878105282279.

## Appendix A. Iwoki Rules

This appendix specifies *iwoki* rules adapted for only two players.

The *iwoki math* is a game to combine the calculation of simple mathematical operations with the spatial perception of two-dimensional objects. Players fit together small and hexagonal tiles, endeavoring to match the result of simple addition or subtraction on the smaller tiles with the numbers on the hexagonal ones. As the game progresses, the options to choose the best strategy will increase. The winner will be the one who gets the highest score after the final point tally.

It is formed by the next elements:

- 48 Small tiles: In each of the three outer corners there is a number from 1 to 4 and a plus (+) or minus (-) sign in the center. Each face has the same numbers, but with the opposite sign.
- 20 Hexagonal tiles: In the vertices there is a number from 1 to 6 or a wildcard, which is represented by a red hexagon. The same values are on both sides, so the order shown on one side will be the opposite on the other.

Before starting, each player randomly chooses 9 small and 3 hexagonal tiles. The small tiles are not seen by the opponent, but the hexagonal tiles will remain visible at all times

### *Appendix A.1. Course of play*

The starting player lays one of his/her tiles on the table. If he/she chooses to start with a hexagonal tile, the highest number on it will be noted on his/her score line. If he/she chooses to start with a small tile, the result of the addition or subtraction that suits him/her best will be noted on his/her score line.

Then, for the rest of the turns, each player will try to place a tile of either type, taking into account the following:

- The result of the addition or subtraction of the small tiles must match the red numbers of the adjacent hexagonal ones.
- The wildcard can be matched with any addition or subtraction, but it can only be matched when the small tile also covers another red number on a different tile.

Figure 11. Shows these examples.

When a player places a tile, the score sheet is filled in by adding the number or numbers in red. Wildcards count as zero. See Figure A.12 for examples.

Small tiles are drawn in the following cases:

- When it is not possible to place tiles of either type, a small one will be drawn and an attempt made to place it. If this is not possible, it becomes the next player's turn. Only one small tile can be drawn each turn.
- If no tiles can be placed and there are no more to draw, it becomes the next player's turn.
- A small tile can also be drawn voluntarily, even if it is possible to place one. Right after drawing it is mandatory to place one of the tiles, either small or hexagonal.

The player will draw a hexagonal tile each time he/she has made a total of three of the following moves:

- Placing a small tile using the subtraction side instead of the addition side.
- Drawing a small tile, either because he/she does not have placement options or because he/she does so voluntarily.
- Passing the turn to the other player in the event that he/she cannot place any tiles.

Every time one of these actions are performed by a player, he/she will place a white token to notify the other players how close he/she is to getting a new hexagonal tile. Once a player

generates three white tokens, he/she will draw a hexagonal tile and remove the white tokens, returning it to empty.

*Appendix A.2. End of the game*

When one of the players runs out of small tiles, regardless of hexagonal tiles, the other player will get a chance to make his/her last move.

The final point tally will then be performed for both players: the final score of the player who has run out of small tiles will be the score he/she has accumulated on the scoring sheet up to that point. On the contrary, the score accumulated by the other player must be subtracted from the highest number on each of the small tiles he/she has left.

The winner of the game will be the one who gets the highest score after the final point tally.