Project 3: Decision Trees for Expert Iteration

Rodolfo Cuan-Urquizo

Abstract—In 2017, Artificial Intelligence (AI) AlphaGo, became history as the first agent capable of defeating the Go board game world champion. The accomplishment motivated AI research to explore and improve learning methods. Novel Expert Iteration method improves the performance of Monte Carlo Tree Search by guiding the agent to better decisions in its simulations, instead of using a random policy. This paper presents an implementation of Imitation Learning using Expert Iteration method which utilizes Decision Trees, instead of Neural Networks, as guidance. The proposed method was capable of generating agents which improve by self playing, in the game of OXO.

Index Terms—Reinforcement Learning, Imitation Learning, Board Game, Artificial Intelligence, Decision Trees, Expert Iteration.

1 Introduction

Board games have been played by humans for more than 3 thousand years. The ability to play this type of games has always been associated with intelligence [1]. This is what makes board games an attractive area of application for Reinforcement Learning and Artificial Intelligence. Board games have set rules which define the states and permitted actions, and then a model of the environment can be constructed [2]. The objective of a board playing agent consists on creating an evaluation function which ranks the possible moves [1].

Simple games can be solved by creating a lookup table or using linear function evaluations [3]. However, these approaches can not handle complex games as Othello, Go or Chess, where actions and states can exponentially increasing [1]. Approaches for solving the complex games and creating agents capable of replicating expert players, have been made by combining nonlinear functions (e.g. Decision Trees and Neural Networks) and Temporal Difference Learning [4] [1] [3].

The recent development around Deep Neural Networks and Convolutions Neural Networks, in combination with reinforcement learning, has made possible the creation of more complex agents and also made board games and active and attractive research area [5] [6] [7]. For example, Artificial Intelligence agent, AlphaGo, made history by being the first agent to defeat a world champion in the board game Go [8]. Alternatively, another recent trend consists of developing generic agents able of learning to play different games [9].

The project described in this paper, uses a Monte Carlo Tree Search (MCTS) algorithm with Upper Confidence Bound applied to Trees (UCT), in order to create agents capable of playing the OXO board game. MCTS make use of simulations to choose its actions. These simulations are often run many times with a random actions to generate an optimal policy. This often is time consuming. The objective of this paper is to improve the algorithm by training a Decision Tree Classifier to generalize the optimal policy with information of previous self-played games. The tree intends to guide the MCTS/UCT towards better decisions, instead of the random moves, similarly to [4] Imitation Learning approach.

This paper proceeds as follows: Section 2 presents background information of relevant terms and context. Section 3 explains the methodology taken in the project. The results and evaluation methods are presented in Section 4. Finally in Sections 5 and 6, the results are discussed along a concluding comments.

2 Background

2.1 Reinforcement Learning and Markov Decision Process

Artificial Intelligence classify learning in three categories: Supervised, Unsupervised and Reinforcement Learning. The first two categories require already gathered data in order to model the problem. Contrary to models using Reinforcement Learning, where previous information is not needed. The objective of this kind of learning consists on understating the relationship between actions and situations, this mapping is called *policy*. The agent learns the optimal policy by exploring their environment and obtaining incentives called *rewards* [2]. RL is the closest to how learning works in humans and nature is Reinforcement Learning. This type of learning works in a similar way as humans learn to drive, kids learn not to touch the hot stove and dogs learn to sit and roll.

Markov Decision Processes (MDPs) are a key element for the development and research of Reinforcement Learning and Artificial Intelligence [10]. MDPs framework models the interaction between the main elements of Reinforcement Learning (states, actions and rewards) to find a policy [2] (Fig. 1). MDP can be solve using different methods such as Linear Programming, Iteration methods, Temporal Difference Learning, Imitation Learning and Monte Carlo methods [2] [10] [4].

2.2 Monte Carlo Tree Search and UCT

Monte Carlo Tree Search (MCTS) is an algorithm used for decision making, that uses search trees and random sampling [11]. MCTS is based on the premises that the value of taking an action can be estimated by random simulations and the values can be used to create an optimal policy

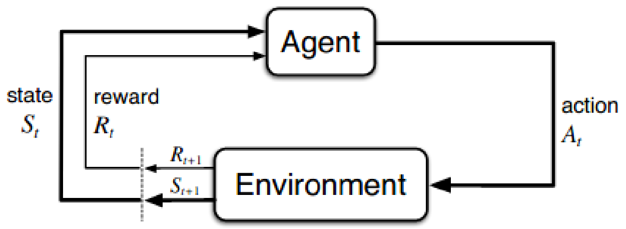

Fig. 1. Markov Decision Process diagram [2]

[11].This method gained popularity and focus in the Artificial Intelligence use in games, after its great performance in the game Go [11].

MCTS represent states as nodes. The parent node is the actual state and its children the possible states the agent can transit. This makes leaf nodes the terminal states of the game.

The algorithm is divided in four main steps [11]:

1) Selection: A policy runs down the tree, starting from the root node, to select an adequate node to expand.
2) Expansion: Child nodes are added, in the selected node, to expand the tree.
3) Simulation: Simulation games are run iterative transiting through the children nodes of the added child.
4) Backpropagation: When a simulation reaches a terminal state, the result its back-propagated through the nodes to retain the information gained.

Simulations use a roll-out or default policy to choose which children node to explore next [11]. Random choosing is the simplest policy.

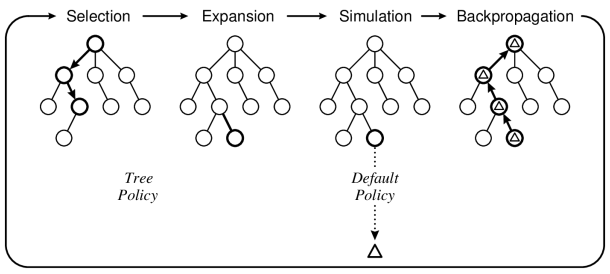

Fig. 2. Graphical representation of MCTS algorithm steps [11].

Additionally to the roll-out policy, MCTS uses a different policy for choosing the children node in which the simulations will take place, called tree policy. The tree policy determines how the tree is constructed by selecting the node to be explore. Tree policies can be deterministic (UCB1, UCB1-Tuned, UCB-V and UCB-Minimal) or stochastic (e-greedy, EXP3, Thompson Sampling) [12]. [12] compares and details several of the policies. UCB1 being the most popular. This policy models the node selection as a multi-armed bandit problem. The usage of MCTS and UCB1 create the most popular MCTS algorithm, called Upper Confidence Bound applied to Trees (UCT) [11]. After all children nodes have been explored, UCT is used to calculate the next node to explore again. The method consider the number of times the node has been explored and the results obtained [13] [14]. The two factors create a balance trade-off between exploration and exploitation, the most important dilemma in Reinforcement Learning. The UCT equation is an extension from UCB1 equation and is presented below [15].

$$UCT(x) = x.wins + \sqrt{\frac{2 * \log x.parent\_visits}{x.visits}}$$

2.3 Learning Methods

The following two learning methods are commonly use by agents to solve complex problems as board games.

2.3.1 *Temporal Difference Learning*

Reinforcement Learning, by its own, usually fails when dealing with complex problems. The main reason being the temporal delay between the action taken and knowing the final outcome, this problem is called *Temporal Credit Assignment* [3] [16]. Temporal Difference Learning targets the problem by creating rewards in intermediate states, decreasing the temporal delay between the action and a reward.

TD-Gammon was one of the first computer agents that played BackGammon, it used this method along a Multi Layer Perception nonlinear function [3]. Other board games solved using Temporal Difference Learning are presented in [16] [17].

2.3.2 *Learning from Demonstration*

Learning from demonstrations is a promising field that can be applicable to complex problems. This type of learning is divided in two trends, Imitation Learning and Inverse Reinforcement Learning. Both methods use two agents: the expert and apprentice [18]. The former provides information in form of labelled data. The apprentice agent uses this information to generalize the expert's policy by imitating it [4] [18]. Learning from demonstration methods weakness is the dependency to an expert agent, which some problems can not provide one.

Novel algorithm, Expert Iteration [4] present a solution to the need of a expert policy by creating an agent capable of playing Go board game. The apprentice guides the expert's decision by generalizing the data obtained by its previous played games by the expert. A tree search algorithm is used as an expert (e.g. Monte Carlo Tree Search, α - β Search, and greedy Search); and Deep Neural Networks are used as apprentices [4].

3 Methodology

3.1 MCTS/UCT Algorithm

The main focus of the project is towards the improvement of the Monte Carlo Tree Search supported by Upper Confidence Bound applied to Trees (MCTS/UCT) algorithm. For this reason, it was opted to use an already implemented version of this algorithm, instead of building one from scratch, and apply the Decision Tree to guide the tree search. The algorithm was developed in 2012, by Peter Cowling, Ed Powley and Daniel Whitehouse from the University of York.

The script presents different board game options to test the algorithm as Othello, Nim and OXO. The latter game was chosen to be used in this project for its simplicity and well-known rules. The script can be found **here**.

As stated by its authors, the python script is not optimal as it intention is to be clear and simple for learning purposes. This makes it ideal for developers and researchers to modify and test improvements in the algorithm. The project presented in this paper is developed on top of the script. The project uses python 2.7, SciKit-learn and Numpy libraries. The developed script is available **here** or in the next url: https://github.com/fitocuan/CE888_2020/tree/master/Assigment .

3.2 Data and Agent Generation

A total of 10 agents were created, the differentiation among them is the data the decision tree was trained on. Each agent is trained with data gathered from its previous agent games.

The DT used is from the SciKit-Learn library and initialized with the parameters presented next:

```
DecisionTreeClassifier(class_weight=None,
     criterion='gini',
     max_depth=None,
     max_features=None,
     max_leaf_nodes=None,
     min_samples_leaf=1,
     min_samples_split=2,
     min_weight_fraction_leaf=0.0,
     presort=False,
     random_state=None,
     splitter='best')
```

Initially, the first agent is trained by gathering data from 200 games (around 1000 instances) played by the 0th agent, an agent using a random Roll-Out policy. The first agent will now use a Roll-Out policy guided by the trained DT. Then, the data set is expanded by 50% with the new agent's self-played games. The second agent is trained with this data set. In the same way, the remaining agents are generated. This method is proposed in [4] as exponential Expert Iteration. The exponential increment lets the data set to grow and use recent data while not consuming much time in each training step (Figure 3). The game used in this project is simple, therefore, the data set generated is relatively small in size. Exponential Expert Iteration may not be needed in this case, nevertheless for future work and other applications it can be handy.

Each instances of the data set consists on 10 features and a label, the board game positions, the player making the move and the move it took (Table 1).

	Pos0	Pos1	...	Pos8	Player	Move
0	0	0	...	0	1	0
1	1	0	...	0	2	4
2	1	0	...	0	1	1
3	1	1	...	0	2	8
...	0	0	...	2	1	5

TABLE 1
Example of Game Instances

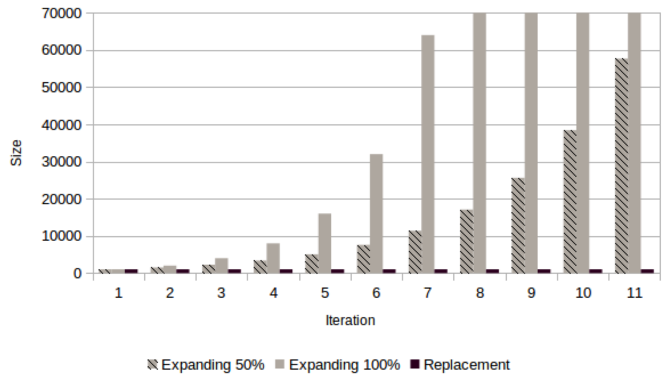

Fig. 3. Data Set size expanding demonstration

3.3 MCTS Simulations

100 simulations were made by the MCTS algorithm throughout the agent generation. After creating the first agent the Roll-Out policy stops being random and start using the trained DT. Both players in the simulations uses the same DT, making many of the outcomes to be draws. When back-propagating the result, draws are granted a reward of 0.5. For this reason, in the simulations it is proposed for the players to use different policies; The main player uses the DT and the adversary, the random policy. This will potentially increase the number of successful simulations and games.

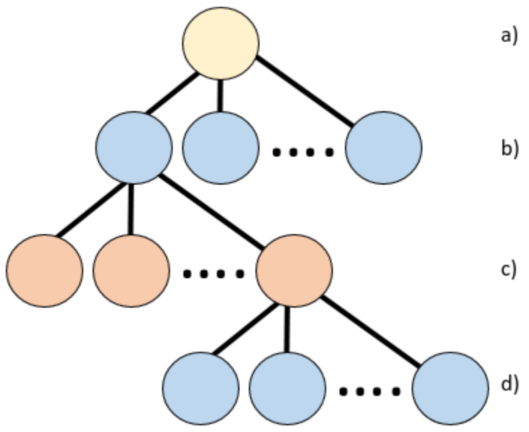

Fig. 4. Simulation example with alternating policies. (a) Starting state of simulation (b) Adversary turn using random policy (c) Main player using Decision Tree policy (d) Second turn of adversary

4 Results

4.1 Agent Improvement

Two experiments were developed to measure the improvement of each generated agent. For each of the experiments the conditions were used:

- The nature of OXO game makes the game to occasionally result in draws, for this reason the agents are evaluated by playing 250 games.
- When the number of MCTS simulations is high, both agents' polices will converge similarly. To take

advantage of new policy the number of simulations are reduced to 30.
- The data set was expanded by 50% for each new agent.

The steps taken before each of the experiments are the following:

1) Generate data set from playing 200 games of 0th Agent vs 0th Agent.
2) Create new agent trained with the data set.
3) Gather new information by playing the new agent against it self, for 200 games.
4) Expand the data set with the new data.
5) Repeat steps 2-4 until the 10 agents are created.
6) Perform the experiment.

The first experiment consisted in matching each agent to play against an agent with a random Roll-Out policy (0th Agent). The second experiment, similar to the first one, match the last generated agent (10th Agent) against the previous 10 (1th Agent to 10th Agent). Both experiments results are displayed in Figures 5 and 6.

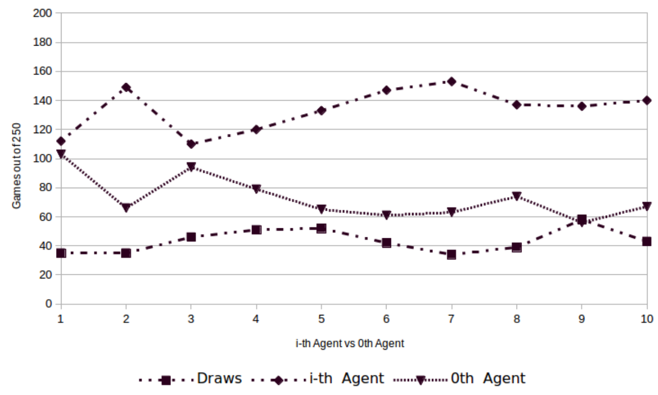

Fig. 5. Graph displaying the performance of 10 agents throughout 250 games.

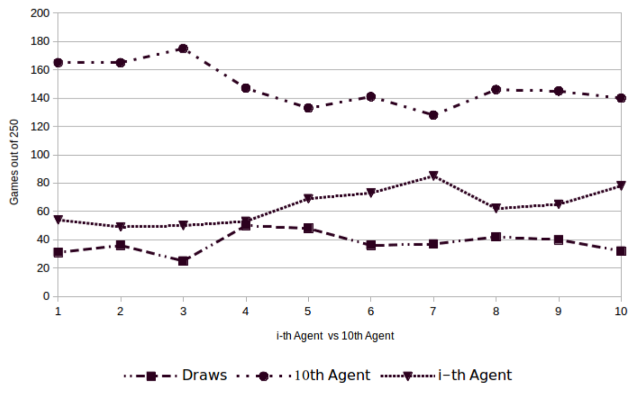

Fig. 6. Graph displaying the performance of 10 agents throughout 250 games.

4.2 Data Expansion

As described in the methodology section, the data set was expanded exponentially between agents. Three sets of 10 agents were generated; 2 which expanded the data set by 50% and 100% each step, and 1 with total replacement of the data set. Similarly to the previous section, the three sets of generated agent were match versus agents using a random Roll-Out policy and alternating policies (Fig. 7).

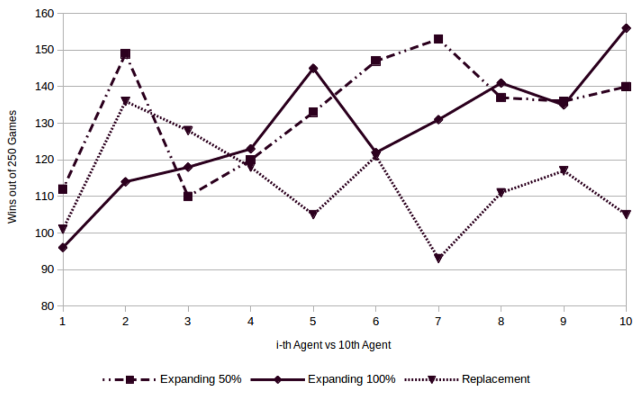

Fig. 7. Graph displaying the performance of three sets of 10 agents throughout 250 games with different data set expansion methods.

4.3 Alternating Policies

Two sets of 10 agents were generated, one using alternating polices and the other without it. Both sets expand their data set by 50% and are tested using the same methodology as the previous experiments. Figure 8 displays improvement in performance on using alternating polices throughout the learning step of the agents.

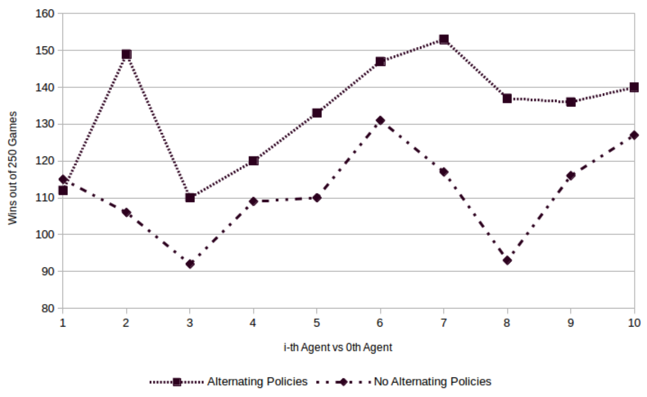

Fig. 8. Graph displaying the performance improvement using alternating policies.

5 Discussion

5.1 First Turn Bias

It was observed that the player with the first turn had more possibilities of winning. This was proven by matching two agents with the same random Roll-Out policy, for 250 games. The agent with the first turn won 46% of the games, while its adversary 9.2%. Figures 5 and 6 its evident the line which represents the agent with the first turn is the one with the better performance. However, the bias was neutralize by not swapping the first turn between the agents through out their 250 matches. 0th agent will always have the second turn in all of its matches.

5.2 Agent Improvement

The methods proposed are capable of generating agents that improve by self-playing, demonstrated with graphs in Figures 5 and 6.

For the first experiment resulted in a increasing tendency, expected for the line representing the i-th agents. The last generated agent is expected to win more versus the 0th agent, than the first agent generated. As observed in the results, the last agent won 140 times and the first one 112 (Fig. 5).

In Figure 6, the line representing the 10th agent wins is decreasing, while its adversary performance increases. This is expected as the agents are capable of choosing better decision as the they are being generated. The last agent is able to win 165 times against the first agent, but 140 against the 10th agent (it self).

Although the overall performance has an improving tendency, the agents are not constantly improving. Some agents perform worst than its predecessor. The cause of the phenomenon is not known, but it might be related to a generalization problem.

5.3 Data Expansion

In Figure 7, both agents sets using the expanding method demonstrate improvement; Contrary, the set which replace the data set do not presence a tendency. It is concluded that an expansion is needed in the data set to presence improvement in the agents. If the agent uses information from all of its previous agents it stops over-fitting from happening.

Expanding the data set by 100% generated a slightly better performance compared to 50%. The expanding factor depends entirely from the characteristics of the board game. A OXO game grants few instances of information each game, however, it is a simple game and does not need big data sets to approximate a playing function. In this case expanding by 100% or 50%, do not require big processing power. Although the more information the better, for complex games its better to use an expanding method to avoid processing demanding data sets. For example in [4], they use the same method but for Hex board game, a more complex game. As they generate more information for each game, their agents expand by 10%.

5.4 Alternating Policies

Graph in Figure 8 demonstrate how this method affect the overall result of the set. The only difference between the sets is the usage of the method. The set of agents using alternating polices method, always had an improving tendency and more number of wins. Opposed to sets that do not use the method, they presence no tendency. Alternating Policies method was used for all of the previous experiments, without it none of them would have demonstrated improvement in their agents.

6 Conclusion

In Modern eras, technologies grants large amounts data sets. Artificial Intelligence and Machine Learning algorithms make use of this information to successfully accomplish tasks. Artificial Intelligence research can be divided in developing new methods and in improving exiting ones. In this project the already existing MCTS algorithm was used. Although the algorithm accomplished successful results, improvements were proposed.

From simple games as OXO to more complex as Chess or Go, board games are present and known in cultures all around the world. This is the reason for research and development around this types of games is attractive. For people not familiar with Artificial Intelligence is more understandable to associate intelligence with tasks and activities they consciously perform. This is why accomplishments for creating agents capable of playing against humans are more astonishing to the general population.

References

[1] I. Ghory. *Reinforcement Learning in Board Games*. Tech. rep. 2004.

[2] R. Sutton and A. Barto. *Reinforcement Learning : An Introduction*. 2nd. Cambridge, MA, USA: MIT Press, 2017.

[3] G. Tesauro. "Temporal Difference Learning and TD-Gammon". In: *Commun. ACM* 38.3 (Mar. 1995), pp. 58–68. ISSN: 0001-0782. DOI: 10.1145/203330.203343. URL: https://doi.org/10.1145/203330.203343.

[4] T. Anthony, Z. Tian, and D. Barber. *Thinking Fast and Slow with Deep Learning and Tree Search*. 2017. arXiv: 1705.08439 [cs.AI].

[5] M. Lu and X. Li. "Deep reinforcement learning policy in Hex game system". In: *2018 Chinese Control And Decision Conference (CCDC)*. 2018, pp. 6623–6626.

[6] Y. Chen and Y. Li. "Macro and Micro Reinforcement Learning for Playing Nine-ball Pool". In: *2019 IEEE Conference on Games (CoG)*. 2019, pp. 1–4.

[7] D. Draskovic, M. Brzakovic, and B. Nikolic. "A comparison of machine leaming methods using a two player board game". In: *IEEE EUROCON 2019 -18th International Conference on Smart Technologies*. 2019, pp. 1–5.

[8] D. Silver, J. Schrittwieser, K. Simonyan, et al. "Mastering the game of go without human knowledge". In: *Nature* 550.7676 (2017), pp. 354–359.

[9] M. Rezende and L. Chaimowicz. "A Methodology for Creating Generic Game Playing Agents for Board Games". In: *2017 16th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*. 2017, pp. 19–28.

[10] L. Littman, L. Dean, and L. Kaelbling. *On the Complexity of Solving Markov Decision Problems*. 2013. arXiv: 1302.4971 [cs.AI].

[11] C. B. Browne, E. Powley, D. Whitehouse, et al. "A Survey of Monte Carlo Tree Search Methods". In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1 (Mar. 2012), pp. 1–43. ISSN: 1943-0698. DOI: 10.1109/TCIAIG.2012.2186810.

[12] P. Perick, D. St-Pierre, F. Maes, et al. "Comparison of different selection strategies in Monte-Carlo Tree Search for the game of Tron". In: vol. 1. Sept. 2012, pp. 242–249. ISBN: 978-1-4673-1193-9. DOI: 10.1109/CIG.2012.6374162.

[13] X. Li, L. Hou, and L. Wu. "UCT algorithm in Amazons human-computer games". In: *The 26th Chinese Control and Decision Conference (2014 CCDC)*. May 2014, pp. 3358–3361. DOI: 10.1109/CCDC.2014.6852755.

[14] Pierre-Arnaud P. Coquelin and R. Munos. "Bandit algorithms for tree search". In: *arXiv preprint cs/0703062* (2007).

[15] Y. Fan. "Bandit Algorithms in Game Tree Search: Application to Computer Renju". In: *University of British Columbia, University of British Columbia2012* (2012).

[16] Y. Osaki, K. Shibahara, Y. Tajima, et al. "An Othello evaluation function based on Temporal Difference Learning using probability of winning". In: *2008 IEEE Symposium On Computational Intelligence and Games*. Dec. 2008, pp. 205–211. DOI: 10 . 1109 / CIG . 2008 . 5035641.

[17] M. Szubert, W. Jaskowski, and K. Krawiec. "Coevolutionary Temporal Difference Learning for Othello". In: *2009 IEEE Symposium on Computational Intelligence and Games*. Sept. 2009, pp. 104–111. DOI: 10.1109/CIG. 2009.5286486.

[18] B. Piot, M. Geist, and O. Pietquin. "Bridging the Gap Between Imitation Learning and Inverse Reinforcement Learning". In: *IEEE Transactions on Neural Networks and Learning Systems* 28.8 (Aug. 2017), pp. 1814–1826. ISSN: 2162-2388. DOI: 10 . 1109 / TNNLS . 2016 . 2543000.