CE888 Assignment 2: Report on Decision Trees for Expert Iteration (Reinforcement Learning)

Rohit Venugopal (1901309)
School of Computer Science and Electronic Engineering
University of Essex
Colchester, UK
rv19514@essex.ac.uk

***Abstract*—The use of Artificial Intelligence in games has always been of interest to the research community and recent successes for creating robust agents can be credited to the approach where both Reinforcement Learning and Supervised Learning are combined. This paper proposed an approach to combine Monte Carlo Tree Search, a popular algorithm, with Decision Trees; a supervised learning model. The primary idea behind this approach is to create an agent by mimicking the Dual-Process Theory, which states that humans possess two distinct types of thinking. The MCTS algorithm shall perform the task of long term planning, while the Decision Tree shall learn about the domain, and help the MCTS by generalising the problem. Experimental results show that a combination of Decision Trees and MCTS yields promising results, and are certainly a step in the right direction. However, more work shall be needed to fully utilise the power of Supervised Learning within MCTS. In addition, in certain situations, the Decision Tree was not able to capture all the intricacies of the domain, which could be solved by using a more powerful model like Neural Networks; though, since they are faster than Neural Networks, use of Decision Tree can help MCTS search more if used in a time-restricted environment.**

I. Introduction

Agents capable of dealing with and handling real world, complex situations is one of the biggest challenges of Artificial Intelligence. However, developing and testing agents in the real world may not always be viable due to various factors like cost, complexity and safety. This is one of the main reasons computer games are considered to be ideal environments to build, develop and test various artificial intelligence agents; since they are faster, safer, easier to modify and have a lower cost of research. As such, games have received a lot of attention from the AI community over the last few years.

One of the most notable achievements was when IBM's Deep Blue [1] defeated the Chess Grandmaster Kasparov in 1997. However, Deep Blue made use of extensive brute force search, to find out the best move possible at a given scenario. In the following years, there was shift in creating agents to play games. Instead of making use of tree search algorithms, like minimax or A*, there was shift to using Supervised Learning. Expert knowledge was used to formalize the rules for various games and create datasets. These datasets were then used to train a supervised model, which an agent would then use to play the game.

More recently, Reinforcement Learning has had a huge success in creating agents. Agents making use of Reinforcement Learning in general showed promising results, however one drawback was the amount of time it took the agent to play and learn the game, before it became proficient. Eck and Wezel [2] in their paper discuss the application of Reinforcement Learning to playing Othello, and [3] proposes an approach which uses TD-Learning to create an Othello agent.

While Monte Carlo Tree Search (MCTS) has been the go to technique for many games in recent years, including Othello [4], the concept of biasing MCTS with domain knowledge to improve its performance, has seen a significant increase in interest ever since AlphaGo [5] defeated the World Go Champion. Research is still going on how to best bias the MCTS algorithm for different games. Biasing the MCTS helps the algorithm perform a more guided search, and as such, domain knowledge becomes necessary. However, if it is biased too much, MCTS will not search enough, and the agent may perform poorly.

This project's main objective is to develop agents that make use of classifiers like Decision Trees to bias the Monte Carlo Tree Search (MCTS) technique, to play the game of Othello. MCTS shall be used to perform a lookahead search of the game, while the Decision Tree classifier shall guide the search of MCTS, and help predict the best move at a particular stage of the game.

This document is structured as follows. First, a brief explanation of the game (Othello), the background necessary, and a thorough discussion of related work is presented in section II. Section III discusses the data generation process, as well as the core techniques used in the project. Evaluation methods to compare various agenets, as well as the discussion of the results are described in sections IV and V respectively. Finally, section VI concludes the paper.

II. Background

A. Othello

Othello, also known as Reversi, is a popular, strategic board game that can be played by two players. The game is played on an 8x8 uncheckered board, and the players make use of a game piece known as a disk. The colour of the disk is either black or white, depending on the player. The initial configuration of the game can be seen in Fig 1. During a player's turn to place their token or disk on the board, the objective is to outflank the opponent's pieces. When a piece is outflanked, the disk is flipped over and changes its colour. The game ends when either both the players pass, or the board is filled with tokens. The goal of the game is for players to try and populate the board with disks of their colour. The player having a majority of their coloured tokens displayed on the board wins.

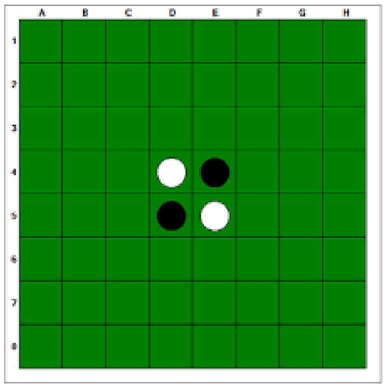

Fig. 1. Othello board and initial configuration

B. Monte Carlo Tree Search

The Monte Carlo Tree Search (MCTS) algorithm, surveyed in [6], is a tree search technique that makes use of stochastic simulations, and can perform well without domain-specific knowledge. Over the years, MCTS has been used a lot in games such as Go [5], Scotland Yard [7] and Magic: The Gathering [8]. MCTS can be applied in any game which is of finite length. MCTS makes use of a time budget to search the game tree for the next best move to play. If a higher time budget is given to the algorithm, it can search more and infer a more relevant answer. This is what gives the MCTS algorithm its anytime property; that is the algorithm can be stopped at anytime, and it will still be capable of providing an answer.

The algorithm runs for a predetermined time budget or for a fixed number of iterations, and in each iteration, there are four phases:

- Selection: In a particular state or node, an action has to be chosen in a way that balances both exploitation and exploration. In exploitation, the move which leads to the best result is chosen, while in exploration, the algorithm searches the state space for moves which might give better results. The balance between both these criteria is achieved by making use of the UCT (Upper Confidence Bound) formula [9].

$$J(a) = V(a) + C\sqrt{\frac{lnN}{n(a)}} \tag{1}$$

  In (1), $n(a)$ is the number of times a node was visited, while N is the number of times the node's parent was visited. C is a tunable parameter that is used to control the trade-off between exploration and exploitation, and $V(a)$ is the approximate value of the node (or how many times the node took action a. The action with the largest value of $J(a)$ is selected at each node.
- Expansion: One of the untried actions is selected at random, and expanded, thereby adding it to the game tree. Due to this, in each simulated game, the tree grows bigger by a node.
- Simulation: This part is also known as Rollout. Here actions are chosen either at random, or by using some domain-specific policy, until the the end of the game to obtain the final game score.
- Backpropagation: The game's final score is backpropagated through the tree, and all nodes that were traversed are updated. Each node calculates the mean scores over all iterations as $V(a)$, which will be used in future iterations.

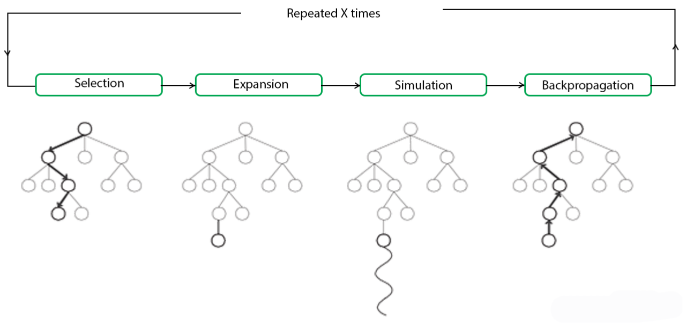

Fig. 2. Monte Carlo Tree Search as seen in [10]

Fig 2 depicts the 4 steps in an MCTS iteration. Since the MCTS algorithm performs a stochastic search, if the algorithm is allowed to run for long periods, optimal results can be found for any given problem. However, this is not always feasible since most games and applications need to work in real time. Due to this, the MCTS technique will have to perform its search in a limited time frame, and the results returned may not always be optimal.

C. Related Works

Developing agents to play games has been an integral part of AI research history. Board games typically provide perfect information and have precise rules, which can be formalized easily; which makes it an ideal choice for the research and development of various AI techniques.

One of the most famous Othello agent is Logistello [11], which made use of game tree searching and efficient pruning, to find the best possible move at any given stage of the game. Sannidhanam et al. [12] analysed the importance and use of heuristics and how it can improve an agent using Minimax algorithm to play the game of Othello.

The Monte Carlo Tree Search (MCTS) algorithm is one of the most widely used and popular technique for creating agents. MCTS is a tree-based search technique, which can search the game tree for as many iterations as needed, and provide the best possible move for a player at any given stage of the game. One of the main advantages of MCTS is its 'anytime' property. That is, the algorithm can be stopped at anytime, and it will still provide a solution to the given problem.

Nijssen [4] developed an AI agent which used MCTS to play the game of Othello. Generally, in the Simulation or Rollout phase of MCTS, the algorithm makes use of random moves. Robles et al. [13] tried to incorporate domain knowledge of Othello, to improve the MCTS algorithm. They employed Temporal Difference Learning (TDL) to capture domain knowledge and use it to bias the MCTS algorithm. The biased MCTS agent won most of the games when it played against pure MCTS agents and Minimax agents.

Benbassat and Sipper [14] proposed a combination of evolutionary algorithm and MCTS to play Othello. The current board configuration was provided to the evolutionary algorithm to select best possible moves, and help guide the MCTS algorithm. Jain, Verma et al. [15] implemented an agent which made use of the eXtended Classifier System (XCS) framework. The XCS framework formalized rules, which evolve over time, as the agent keeps playing games. It was shown to offer advantages over other reinforcement learning approaches due to its adoption of the rule based format.

AlphaGo [5] combined Deep Neural Networks with Reinforcement Learning, with the Neural Network acting as the guide for the MCTS algorithm. A similar approach was also done by [16], where MCTS was combined with Convolutional Neural Network.

Liskowski and team [17] developed an agent for Othello, that utilized a convolutional neural network, that took as input the current board configuration, and provided the best move as output. The agent was able to win against all existing Othello agents, including Edax [18], which is the best open source agent to play Othello. Soemers et al. [19] implemented a combination of MCTS and a Linear Function Approximator (LFA). The LFA was the used to guide the MCTS search. While this technique did not show any remarkable improvement or exceptional results, it offered a few advantages over the typical MCTS and Deep Neural Networks model, such as faster training period, better interpretability, and a lower utilization of computer resources.

III. Methodology

A. Generating Data

The pure MCTS agent was made to play 30 games of Othello with itself in order to generate data. Each 10 games out of the 30 games, used 500, 300 and a 100 iterations in the MCTS algorithm. Since a lesser number of iterations means that the MCTS algorithm doesn't search or explore as much, if an agent uses lesser iterations than its opponent, it has a higher probability of losing the game. Due to this reason, both instances of the MCTS agent (player 1 and player 2), used the same number of iterations, so that they have an equal chance at winning the game.

The generated data included features such as current configuration of the board, the player number and the next best possible move. Each of the position on the Othello game board is recorded as a feature in the dataset, and encoded as a number; 0 if empty, 1 if player 1 has a token in that position, and 2 if player 2 has a token in there.

Due to the fact that an 8x8 board is used for the game of Othello, there are effectively 64 positions on the board. However, since 4 of them come under the initial configuration of the game, the multi-class classifier developed only needs to predict the best move from among 60 classes.

The generated data can be viewed in the GitHub repository for the Data Science Lab, by clicking here.

B. Creating Agents

MCTS agents were first used to play the game and also generate the necessary data. The main objective of this project is to build a Supervised Classifier that can learn from the domain knowledge of the game and use it to bias the Monte Carlo Tree Search (MCTS) algorithm, and try to improve its performance.

With the generated dataset, a Decision Tree Classifier was built, which took as input the board configuration and the player number, and gave as output, the best possible move at that stage in the game. The Decision Tree classifier tried to incorporate domain knowledge of the game to try and predict the best action to take.

Due to the large size of the Othello board, and the encodings, the Decision Tree built was only able to achieve accuracies of up to 6%. This trained Decision Tree model was then used to replace the rollout function of the MCTS algorithm. The move predicted by the Decision Tree is first checked to make sure that the move is legal, i.e., it should be possible to play that move. If it is legal, then the the agent shall play the move, however, if it is not a legal move, then the agent will play a random move. By using a Decision Tree in the Simulation or Rollout phase, the search done by the MCTS agent became more guided, and the moves selected were not on a random basis. This could result in an increase in the performance of the MCTS algorithm if the Supervised model is able to capture the necessary domain knowledge. However, if the model is not able to learn properly, it could hinder the MCTS algorithm, as the algorithm will not explore as much. This could result in a poorer performance by the combined agent.

Once the combined MCTS and Decision Tree agent was built, it played against the pure MCTS agents for another 30 games, and more data was generated. This new data was used to train the next generation of the agent. This process continued for 10 iterations, resulting in 10 generations of the MCTS and Decision Tree hybrid agent.

IV. Experiments

A. Evaluating the Decision Tree

The generated dataset was first split into a train and test sets, prior to building the decision tree. A decision tree was then built by training it on the train set, with 5-fold cross validation. GridSearch was used to tune the parameters of the Decision Tree and build the best tree. The trained Decision Tree model was then tested and evaluated on the test set of the data, before being saved and used by the agent.

B. Testing the Agent's Performance

The decision tree created was used to modify the Rollout or Simulation phase of the MCTS algorithm, so that the agent would incorporate domain knowledge, and take more informed decisions.

After each iteration of the MCTS and Decision Tree agent was built, it was made to play a predetermined number of games with the pure MCTS agent. The overall wins, losses and draws were then recorded.

C. Running a Competition

A competition was designed so as to evaluate the performance of every agent. A competition basically consisted of a predefined number of rounds, and in each rounds, a predefined number of games. For the competition, a common pool of players was created, with the pool consisting of all 10 generations of the hybrid agent, and the pure MCTS agent.

In each round, 2 players are chosen randomly and then made to play with each other for a certain number of games. An average score is assigned to all agents, based on the number of wins, and the number of games they played in total. A win by an agent, gives them a score of +1, while a draw gives them a score of +0.5. The pseudocode for the competition can be seen in Algorithm 1

Algorithm 1: Algorithm for the Competition

```
Result: playerScores and playedGames for each agent
Initialization of numberOfRounds
Initialization of numberOfGames
while not end of numberOfRounds do
    Choose two random players from common pool of
      agents
    while not end of numberOfGames do
        Update playedGames
        Play match between two players
        Record score
        Update playerScores
    end
end
```

V. Results and Discussion

The decision tree was trained on the generated data and on evaluation, it achieved accuracies of up to 6%. There can be multiple reasons for this. Firstly, the decision tree is tasked with capturing the domain knowledge of the game, and predicting the best possible action to take at any stage of the game. This is not a trivial task, and the decision tree has to predict the correct class among 60 classes. Another reason is that, it could be possible that using more features in data, and generating more data could help achieve the decision tree model achieve a better accuracy score. However, even a very small accuracy achieved by the decision tree can be useful in biasing the MCTS algorithm, since it is capturing the domain knowledge in certain aspects.

Once 10 generations of the Hybrid (MCTS and Decision Tree) Agent was built, each of the agent played 30 games with the pure MCTS agent, and the results were recorded. This can be seen in Fig 3.

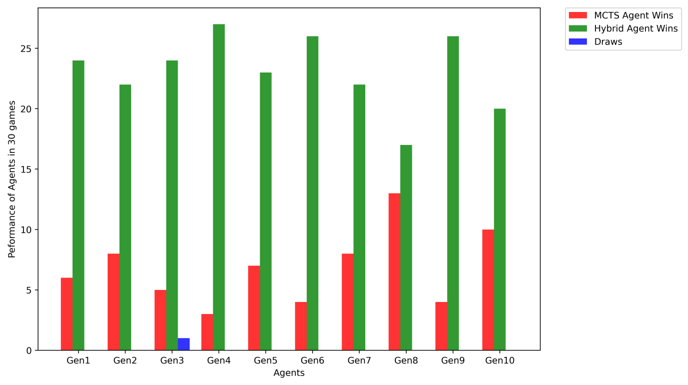

Fig. 3. Comparison of Hybrid Agents to pure MCTS agents

As it can be observed in Fig 3, the certain generations of the Hybrid agent performs really well when playing with the MCTS agents, however in certain generations, the performance drops a lot. One of the possible reasons for this is that, the Decision Tree could not efficiently capture the domain information of the game. Due to this, usage of the Decision Tree model could hinder the performance of the MCTS algorithm, since it will not explore as much, and instead take actions which were wrongly predicted.

To get a better idea of the performance of the various agents, the competition described in Section IV was used. The competition consisted of 150 rounds, and each round containing 1 game; so overall, a 150 games were played by various agents.

For each game, two players were randomly chosen from the common pool of agents, and were made to play with each other. If an agent won, it was awarded a score of 1, and a score of 0.5 was given if the game resulted in a draw. However, comparing the scores of different agents will not give an accurate idea of the performance, since different agents might play different number of games. Due to this, the score of each agent was divided by the number of games played by the agent. This resulted in an average percentage of wins of each player in the competition.

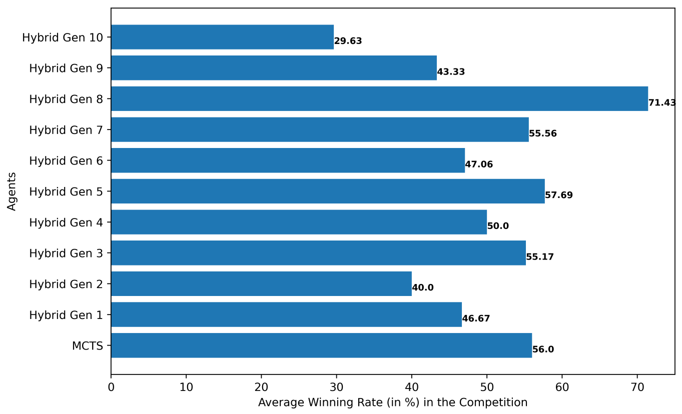

Fig. 4. Results of the Competition

The average win rate of each of the agents can be seen in Fig 4. The pure MCTS agent has an average win rate of around 56%. The first generation of the hybrid MCTS-DT agent does not perform very well in comparison. This could be due to the fact the Decision Tree was not able to learn well. Generation 2 also notices a drop a performance. Generations 3, 5 and 7 manage to achieve a similar win rate as that of the MCTS agent in the game. There are fluctuations in the win rates of the different generation of agents, however, an increasing trend can be noticed. The best performing agent is Generation 8 of the Hybrid agent, with the player achieving a win rate of around 71.43% in the competition.

However, on analyzing Fig 3, we can see that the Generation 8 agent has a similar number of wins, as compared to the MCTS agent, and this is much lower than the other generations. This could possibly mean that while the other hybrid generations were able to achieve more wins when playing with pure MCTS, they could not perform as well when playing against itself or against other generations of the hybrid agent. On the other hand, Generation 8, managed to beat MCTS, as seen in Fig 3, and also managed to beat most of the other generation of hybrid agents, which can be observed in Fig 4.

After the 8th Generation, a decline in the performance of the Generation 9 and 10 can be noticed. One of the possible reasons is that the Decision Tree biases the Hybrid agent too much, and it is not able to perform as well against its counterparts. That is, while it proficient enough to win against the pure MCTS agent, it is not good enough to win against itself or against the previous generation of hybrid agents.

Table I shows the competition scores of the best 5 agents, as well the number of games won by them (out of a total of 30 games), when played against the pure MCTS agent.

VI. Conclusions

Othello is a strategic game, and while it not considered as difficult to play as Go, it is still one of the hardest strategic games with a big learning curve, which might take humans years to master. However, since it is a board game, and it does not have any hidden information, like Poker or Hanabi, the rules of the game can be easily formalized, and the board configuration for the game can be encoded.

TABLE I
Performance of the best 5 agents from the Competition

Best Agents	Competition Winning Rate	Number of games won in 30 games against MCTS Agent
MCTS-DT Gen 8	71.43%	17
MCTS-DT Gen 5	57.69%	23
MCTS	56%	13
MCTS-DT Gen 7	55.56%	22
MCTS-DT Gen 3	55.17%	24

Many agents have been created over the years to play this game, utilising techniques such as Tree Search with Pruning [11], Temporal-Difference Learning (TD-Learning) [3] and different variations of the MCTS algorithm.

This project tried to draw inspiration from the dual-process theory, according to which humans have the capability to think for both long term and short term tasks. The project implemented a Monte Carlo Tree Search (MCTS) for an agent, and used it to generate data. The data was then used to build a classifier, which could learn the domain knowledge, and then help guide the search of the MCTS agent.

The MCTS agent mimicked the human process of planning, while the Decision Tree classifier used knowledge of the game to try and generalise the task, and make immediate decisions.

Among the various generations of the hybrid MCTS-DT agents, we can see fluctuations in the performance, but in general, there is an upward trend. The competition designed helped give an approximate idea as to the capabilities and proficiency of the agents. The best performing agent in the competition was Generation 8 of the hybrid MCTS-DT model, which had a win rate of around 71.43%.

However, there were a few generations of the hybrid MCTS-DT agents which did not perform well. One of the main reason behind this could be the fact that the Decision Tree did not learn well enough, and as a result, it biased the MCTS search too much. This could lead to an overall poorer performance.

Using a Neural Network as a classifier instead of Decision Tree could have resulted in a better performance, since Neural Networks are known to be Universal Function Approximators. However, the MCTS-DT approach offers one major advantage over MCTS and Neural Networks, which is mainly speed of running, and lower resource consumption.

For future work, Neural Networks can be used in place of Decision Trees, and if possible even a combination of Neural networks and Decision trees (such as an inverse model). This could possibly result in a huge improvement in the performance the agent.

References

[1] M. Campbell, A. J. Hoane Jr, and F.-h. Hsu, “Deep blue,” *Artificial intelligence*, vol. 134, no. 1-2, pp. 57–83, 2002.

[2] N. J. van Eck and M. van Wezel, “Application of reinforcement learning to the game of othello,” *Computers & Operations Research*, vol. 35, no. 6, pp. 1999–2017, 2008.

[3] S. van den Dries and M. A. Wiering, “Neural-fitted td-leaf learning for playing othello with structured neural networks,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 11, pp. 1701–1713, 2012.

[4] J. Nijssen, “Playing othello using monte carlo,” *Strategies*, pp. 1–9, 2007.

[5] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, “Mastering the game of go without human knowledge,” *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[6] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, “A survey of monte carlo tree search methods,” *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[7] P. Nijssen and M. H. Winands, “Monte carlo tree search for the hide-and-seek game scotland yard,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 4, pp. 282–294, 2012.

[8] P. I. Cowling, C. D. Ward, and E. J. Powley, “Ensemble determinization in monte carlo tree search for the imperfect information card game magic: The gathering,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 4, pp. 241–257, 2012.

[9] L. Kocsis and C. Szepesvári, “Bandit based monte-carlo planning,” in *European conference on machine learning*, pp. 282–293, Springer, 2006.

[10] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, “Monte-carlo tree search: A new framework for game ai.,” in *AIIDE*, 2008.

[11] M. Buro, “Logistello: A strong learning othello program,” in *19th Annual Conference Gesellschaft für Klassifikation eV*, vol. 2, Citeseer, 1995.

[12] V. Sannidhanam and M. Annamalai, “An analysis of heuristics in othello,” 2015.

[13] D. Robles, P. Rohlfshagen, and S. M. Lucas, “Learning non-random moves for playing othello: Improving monte carlo tree search,” in *2011 IEEE Conference on Computational Intelligence and Games (CIG’11)*, pp. 305–312, IEEE, 2011.

[14] A. Benbassat and M. Sipper, “Evomcts: Enhancing mcts-based players through genetic programming,” in *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, pp. 1–8, IEEE, 2013.

[15] S. Jain, S. Verma, S. Kumar, and S. Aggarwal, “An evolutionary learning approach to play othello using xcs,” in *2018 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8, IEEE, 2018.

[16] T. Anthony, Z. Tian, and D. Barber, “Thinking fast and slow with deep learning and tree search,” in *Advances in Neural Information Processing Systems*, pp. 5360–5370, 2017.

[17] P. Liskowski, W. Jaśkowski, and K. Krawiec, “Learning to play othello with deep neural networks,” *IEEE Transactions on Games*, vol. 10, no. 4, pp. 354–364, 2018.

[18] R. Delorme, “Edax (othello program),” 2004.

[19] D. J. Soemers, É. Piette, and C. Browne, “Biasing mcts with features for general games,” in *2019 IEEE Congress on Evolutionary Computation (CEC)*, pp. 450–457, IEEE, 2019.

Appendix

The development and testing of the project took place on Google Colaboratory. All the files related to this project, such as the generated data, various generations of the Decision Tree, and the program files (data generation and Competition), can be found in the GitHub Repository of the Data Science and Decision Making Lab. The repository can be accessed by either clicking here or by following this url: https://github.com/Rohitv97/Data-Science-CE888.