Project 3: Decision Trees for Expert Iteration (Reinforcement learning)

February 7, 2020

1 Description

A combination of Neural Networks (NNs) and Monte Carlo Tree Search (MCTS) has resulted in really strong players for a multitude of games.

The basic idea is to create a learning loop with the following steps:

1. Multiple games are played using a standard search algorithm like MCTS.
2. The actions that the player took at each state are kept and used to train a neural network.
3. The neural network is used to enhance/guide the search algorithm.

For this project we are going to implement a simpler version of this algorithm (no neural network included!) using decision trees as the classifier.

2 Tasks

1. Using the code provided here, generate (and collect) sample data for at least one kind of game (e.g., OXO, Othello). Sample data that includes combinations of state values (keep your encoding as close to the game mechanics as possible) and the action that the agent took. For example, for OXOState/Game, the state is the board and who is playing; the action is the action the MCTS/UCT took.
2. Use a machine learning approach (e.g., a decision tree) to learn a fast classifier that is able to predict which action to take, by using supervised learning on the MCTS/UCT actions.
3. (OPTIONAL) If you want to improve performance further, instead of feeding the above classifier with data from the the board directly, use a neural network to learn to predict the next state given an action (this is termed the inverse model) and use the middle layer of the neural network as input to the decision tree. — To do this you might have to modify the way you collect your data.
4. Modify the following part of the MCTS function in the code provided so that, instead of doing a purely random search, it uses the learned classifier and finds out which action to take. Do the best action 90% of the time, while playing randomly 10%:

```
# Rollout - this can often be made quicker using a state.GetRandomMove() function
while state.GetMoves() != []: # while state is non-terminal
    state.DoMove(random.choice(state.GetMoves()))
```

5. Collect the data using this approach. You now have a new set of states and actions — repeat the above procedure for as many iterations as possible.
6. Every 10 iterations of the whole algorithm, have an agent play with all its past self 10 games and record the results — are the agents improving?

3 References

1. [Anthony, Thomas, Zheng Tian, and David Barber. "Thinking fast and slow with deep learning and tree search." In Advances in Neural Information Processing Systems, pp. 5360-5370. 2017.

2. Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert et al. "Mastering the game of Go without human knowledge." Nature 550, no. 7676 (2017): 354.

University of Essex

CE888-7-SP-CO: Data Science and Decision Making
Project 3: Decision Trees for Expert Iteration (Reinforcement Learning)

ARITRA GANGULY
Registration Number: 1906467

University of Essex – Colchester Campus
Department: Computer Science and Electronic Engineering
Assignment 2 – Final Project Report
Dr Haider Raza (Supervisor)
Dr Ana Matran-Fernandez (Teaching staff)

April 10, 2020

Abstract

In recent days, the success in games using Artificial Intelligence agents has been a wonderful phenomenon, especially using the combination of Supervised and Reinforcement Learning. The combined performance of Monte Carlo Tree Search (MCTS) and Neural Networks (NNs) gave an outcome of really strong players for a series of multitude of games. The two classical approaches of Artificial Intelligence in games are high quality of domain knowledge and effective AI behaviour [1]. This project proposes to combine classifiers like Decision Tree along with Monte Carlo Tree Search (MCTS) to play the game of tic-tac-toe (OXO). The main idea in this problem is to try a two-way approach which states that human responses are also two definite types. This paper is organized in the following manner. Firstly, a brief introduction of Artificial Intelligence in games, along with an in-depth explanation of related work done in the past are all described in the *Background* section. The data generation process and the implementation of different techniques are explained in the *Methodology* section. The approach taken to compare the two different agents, as well the various discussions are all described in detail in sections *Experiments* and *Results* respectively. Finally, the *Conclusion* section concludes the paper.

1 Introduction

The application of Artificial Intelligence used in games has been a hot topic of research recently and it has grabbed a lot of attention to the AI Research industry since DeepMind's AlphaGo defeated the world Go Champion in 2016.

Games like Tic-Tac-Toe (OXO), Othello, Chess, Go, etc. have several possible actions or moves that can be played. This possible action of moves increases exponentially as the game goes forward. The chances of winning the game increases highly if we can predict every possible move and it's consequences that may take place in future. But the computation power to predict every possible move increases to a high extent when the moves at every step increases exponentially. For this reason, creating agents which use a purely Supervised Learning algorithm can be really troublesome at times since the agent sometimes requires to create several datasets and rules for the game. On the other hand, Reinforcement Learning has shown a lot of potential to deliver good production in a multitude of games despite it taking time to understand the game properly before attaining full level of expertise. Hence Monte Carlo Tree Search algorithm is used in these games to predict the path (moves) that should be chosen in order to reach the final winning solution. In order to find the right path that will lead to victory, it is necessary to arrange the moves from the present state of the game and it will therefore form a tree when all the moves are connected together which is also known as a search tree. Monte Carlo Tree Search Algorithm searches every possible move that exists after each turn of the game.

For Example: In the game of tic-tac-toe, there are many different combinations for the player to give the move, which can be visualised using a tree representation. The move can be further increased for the next turn evolving the diagram into a tree [2].

There has been a lot of development going on the combination of Supervised and Reinforcement Learning algorithms so that these artificial players can make really good decisions in playing games. These players can be good enough to beat the world human

champions as well and can change the perception of how we think about Artificial Intelligence and its applications in real life.

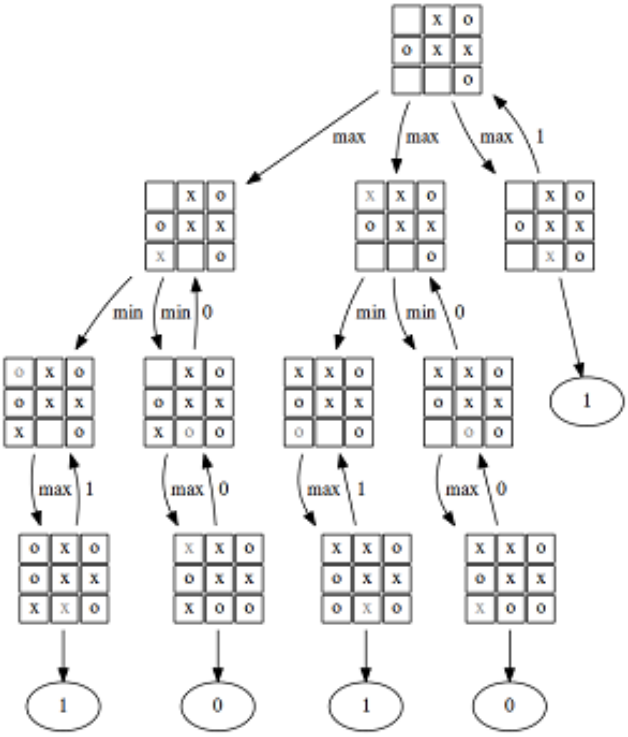

Figure 1: Example search tree for tic-tac-toe [Source]

2 Background

2.1 OXO

The tic-tac-toe which is also known as OXO is a 3X3 grid, pen and paper game, played by two persons. One person plays 'X' and another person plays 'O', who takes turns each time making the space in a 3X3 grid. In order to win, a player needs to successfully place three of their marks in a horizontal, vertical, or diagonal row.

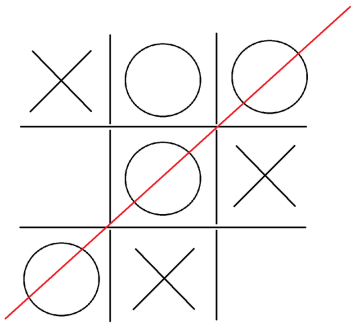

Figure 2: The following OXO example game is won by the second player, O

In general, the best play from both the players leads to a draw. Hence, tic-tac-toe is most often played by young children, which helps them discover the optimal strategy.

2.2 Literature Review

Games using Artificial Intelligence and building smart agents who can mimic the human ability to take perfect decisions has been a breakthrough in AI Research. In general, 2-player board games have accurate rules which both the players are obliged to follow. These rules are easy to formalize with proper details of information, which makes it a very good choice for developing various Artificial Intelligence algorithms.

Out of these, one of the most popular Reinforcement Learning topic is the Monte Carlo Tree Search (MCTS) algorithm which is used to create artificial agents for playing games. MCTS is basically a type of heuristic search algorithm which generally uses tree search based techniques for searching several iterations and providing the best possible move for the player.

Lee [3] proposed an enhanced version of the Monte-Carlo Tree Search algorithm to play the game of Tic-Tac-Toe. He refers that Monte Carlo Tree Search is a top search algorithm which has been implemented successfully in many board games, especially on the game known as Go. Byung-Doo Lee evaluates the performance of Monte-Carlo Tree Search by playing against each other in the game of Tic-Tac-Toe, and observes that the player who gives the first move, always has an immense advantage over the player, who plays second. Byung-Doo Lee tries to detect the justification that why the player who plays the first move is superior to the player who plays the second despite the fact that the best game result should be a draw. Monte-Carlo Tree Search is a statistical algorithm based on the repeated random sampling, it cannot sufficiently tackle a tough situation that needs a strategy or a technique, especially for the second player. For this, Byung-Doo Lee propose a strategic MCTS (S-MCTS) and show that the S-MCTS player never loses a Tic-Tac-Toe game.

Lee [4] proposes in his research journal Monte-Carlo Tree Search Applied to the Game of Tic-Tac-Toe, that Monte-Carlo Tree Search (MCTS) gave birth to strong computer Go games programs such as MoGo and CrazyStone which defeated human Go professionals played on the 9 X 9 board. Prior to implementing MCTS into computer Go, Byung-Doo Lee tried to measure the winning rate of three board positions which are, center, corner and side, in Tic-Tac-Toe playing as the best first move. The experimental result revealed that the center is the best, a corner the next and a side the last as the best first move.

Fu [5] in his paper explains about the simulation optimization perspective for AlphaGo and Monte Carlo Tree Search. He explained a really good example about the Tic-Tac-Toe working mechanism in his paper (Page 663). Fu mentioned that if each player follows an optimal policy in the tic-tac-toe game then it has a very high chance of resulting in a draw. In his theory the game has a possible combination of 255,000

out of which only few are unique after accounting for symmetries and 765 moves are different in terms of move for both players. The optimal strategy for either of the player can be explain in a small paragraph but for a computer to understand it requires hundred lines of code in any programming languages.

Fogel [6] proposes the use of evolutionary programming to create Neural Networks that are capable of playing tic-tac-toe. He mentioned that all the intelligent systems are evolutionary and there have been three main efforts in simulating evolution which are basically evolutionary programming, generic algorithms and evolution strategies. His research focuses on the use of evolutionary programming for adapting the design and weights of a multi-layer feed forward perceptron in the context of machine learning. As well as, in order to achieve advance level of game-play in the tic-tac-toe game without the use of heuristics algorithms, it is necessary to evolve the structure and weights of a single hidden layer perceptron. The conclusion from his work states states about the high significance of the specific mutual operations, the importance for credit assignment procedures and the power and dominance of the evolutionary search.

Benbassat and Sipper [7] implemented a combination of genetic programming methods along with MCTS algorithm to intensify the level of play games especially in zero-sum, deterministic board games of Othello. The research uses the MCTS algorithm to evolve the evaluation function. Along with the MCTS, the current board setup was given to the evolutionary algorithm to select the best possible set of moves as well as helping the MCTS algorithm. After the research was conducted, it is shown that MCTS with Evolutionary Algorithm or EvoMCTS performs relatively better than the MCTS alone itself, after testing it into two different board games. At last the author proposes that further modifications can also be done with the EvoMCTS algorithm therefore helping the algorithm to be fluent with heavier board games.

2.3 MCTS

The main concept of Monte Carlo Tree Search algorithm is to build a domain search tree by performing successive random play-outs [7]. MCTS algorithm recognizes the best combination of moves out of a series of possible sets of moves by the process of Selection, Expansion, Simulation and Back-propagating the nodes in the tree to find out the final solution. This method iterates repeatedly until it reaches the final solution and hence learns the policy of the game [2]. This approach has proved useful in generating effective board game players and is responsible for the great improvement in level of play seen in games with a high branching factor [7]. MCTS is also a leading approach to many general 2-player board games as well [8]. See the visualization of the MCTS steps in Figure 3.

Principal of Operation:

- Selection: "*MCTS algorithm starts from root node R, then moves down the tree by selecting successive child nodes until a leaf node L is reached. In this section, a node on the tree is selected which has the highest possibility of winning. The node selected is searched from the current state of the tree and since the selected node has the highest possibility of winning, the path is also most likely to reach the solution faster than other path in the tree.*" [2][9]
- Expansion: "*If L is not a terminal node (it does not terminate the game) then create one or more child nodes according to available actions at the current state (node), then and select the first of these new nodes M.*" [9]
- Simulation: "*Run a simulated rollout from M until a terminal state is found. The terminal state contains a result (value) that will be returned to upwards in the is back propagation phase. The states or nodes in which the rollout passes through are not considered visited.*" [9]
- Back-propagation: "*After the simulation phase, a result is returned. All nodes from M up to R will be updated by adding the result to their value and increasing the count of visits at each node.*" [9]

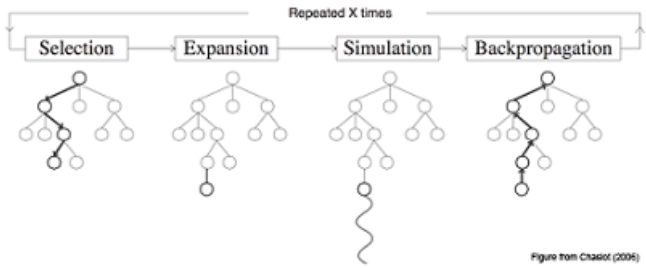

Figure 3: Four phases of MCTS algorithm [1].

2.3.1 *The UCT Algorithm*

One of the better-known variants of MCTS is the Upper Confidence Bounds applied to the Trees (UCT) algorithm [10]. UCT uses the following formula:

$$s_q(c) = \frac{W(c)}{n(c)} + C\sqrt{\frac{log(N(q))}{n(c)}} \quad (1)$$

where:

- $s_q(c)$ is the score of child c of node q.
- n(c) is the number of simulations of move c.
- N(q) is the number of simulations of state q.
- W(n) is the sum of scores for simulations of node n (in games this is often the number of won simulations).
- the constant C controls the compromise between exploitation of good moves and exploration of new moves.

In order to choose a move from game state q, UCT preforms an *argmax* operation as follows:

$$\underset{c \in children(q)}{argmax} \; s_q(c) \tag{2}$$

3 Methodology

3.1 Generating Dataset

Using the source code provided here, the data was generated by letting pure MCTS agents play 100 games of OXO. The dataset consisted of features such as player identification i.e. player 1 or player 2, the current state of the board and the best move that can be played. The player identification is represented by the value 1 and 2 for player 1 and player 2 respectively. Each position on the board is recorded as a feature and identified as a number i.e the value 0 identifies that the position is empty, the value 1 identifies that player 1 has a token there and similarly the value 2 identifies that player 2 has a token there. Since the OXO game is played on a 3X3 grid, hence there are only 9 positions to be marked which are numbered from 0 to 8. The value of the best move will lie between value 0 to 8 after every alternate move. So in order to create a classifier to predict the best move, we need to create a multi-class classifier for 9 classes. In addition too, it is necessary for both the agents (players) to use equal number of iterations while generating data, so that both the players have an equal chance of winning. If any one player has less number of iterations from the other player then there is a high possibility for that player in losing some series of games since less number of iterations will not help that agent to explore more combinations. Finally after generating sample data of the OXO game for 100 iterations, the resultant data is stored inside a .csv file provided here. The .csv data consist of 8 columns and approximately 1750 rows from the iteration of 100 games.

3.2 Techniques Implementations

The project will begin by using the existing MCTS agents to play many games of tic-tac-toe. The MCTS agents are required to generate datasets which are necessary for the upcoming stages. The main agenda of this project is to implement a decision tree classifier that is able to predict the best possible moves from the given game state in order to prejudice the Monte Carlo Tree Search algorithm, and by improving its performance. By prejudicing the MCTS algorithm, the decision tree classifier will replace the rollout function which is now able to predict the best results and can choose the most appropriate move, rather than doing purely random selection, hence will deliver results in a more guided and efficient way. The project will be done till a series of iterations and each iteration, previous agents shall play the game and generate data, and this data will be used to build the next classifier for the next agent. At the end, each of the agents shall be analysed to see the impact of prejudicing the MCTS algorithm with a classifier, as well as to see the level of improvement amongst the various agents.

4 Experiments

The data generated was used to train the decision tree classifier which was then used in the simulation phase of the Monte Carlo Tree Search technique. More data was generated and this process continued for 10 iterations. To compare the performance of the various agents implemented, each of the agents was made to play around 100 games with the MCTS agent. After each game, the wins, losses and draws were recorded. The scores recorded should give a general idea of the performance of every agent with comparison to the MCTS algorithm. Along with this the itermax parameter plays a crucial role in the performance of the agents and result of the game. The itermax is basically the number of iterations to be performed starting from the rootstate. The higher the value of itermax, the higher the search is going to conduct for the optimal best move. The agent with a higher itermax value has a high probability of winning the game. The optimal criteria of a best move for a normal tic-tac-toe after experimentation is as follows:

1. Make a winning move, if available.
2. Block opponent's win, if available.
3. Take the center square.
4. Take a corner square (choose randomly).
5. Take a side square (choose randomly).

5 Results

To make a combination of reinforcement learning and supervised learning, Decision Tree Classifier is implemented in order to prejudice the MCTS algorithm. The new agent is trained on the previous data and then it is used to generate more data by playing games. This procedure continues and more classifiers are implemented on each iteration which are again trained on their previous data. This iteration continues till 10 times. After successfully executing the iterations 10 times by playing the existing MCTS agents with all the new agents we have recorded the result in the table given below and analysed the result in the form of graph shown in Figure 4.

Performance of Agents			
List of Iterations	MCTS Agent	MCTS + DT Agent	Draw
Iteration 1	19	1	80
Iteration 2	21	1	78
Iteration 3	18	2	80
Iteration 4	24	2	74
Iteration 5	16	0	84
Iteration 6	15	3	82
Iteration 7	18	0	82
Iteration 8	35	2	63
Iteration 9	21	0	79
Iteration 10	21	0	79

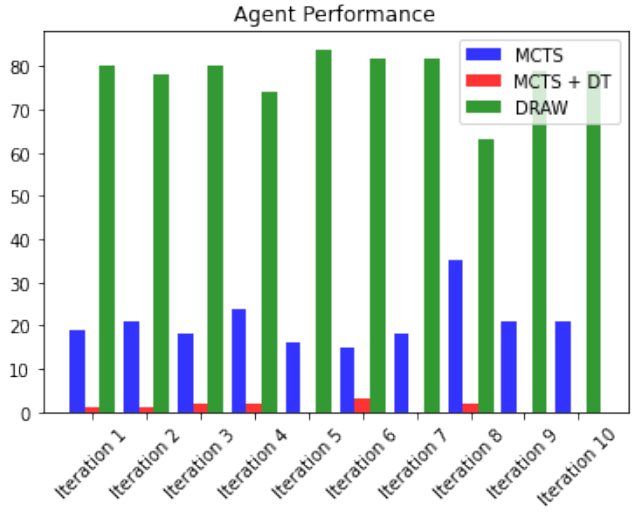

Figure 4: Performance of Agents

By looking at the above data, a statement can be made that the combination of MCTS and Decision Tree is not as impressive as expected. Unlike more complex games such as Go, Othello or Chess; tic-tac-toe (OXO) is a much smaller game with no criteria or strategy of winning. The optimal solution for both the players generally results in a draw in the game, which can be seen in the above table and visually in Figure 4. Despite this, we can see a slight increase in the number of wins for the combination of MCTS and Decision Tree until Iteration 6. After the 6th iteration, the performance of the combine agent sees a drop in performance because of the fact that Decision Tree is restricting the search of MCTS.

6 Conclusion

The game of OXO or tic-tac-toe is a very basic and straightforward game with simple rules and no hidden information. It is not rated with the same level of difficulty as of Go and Othello, but still it consists of various strategies which should be undertaken in order to win. Though, it is a bit easier for an agent to play because of the simplicity of the game and the straightforward rules with no secret information but learning the game could take some time for the agent.

Previously there had been a lot of research conducted on Artificial Intelligence in games and researcher implemented several approaches like Minimax, Brute-Force Techniques, along with normal MCTS and combination of MCTS with Neural Network or with Generic Programming, etc. to build agents that can play games. This project we are using the combination of MCTS and Decision Tree Classifiers to help guide the search.

Initially it is expected that the resulting agent i.e. the combination of MCTS and Decision Tree will surpass the existing MCTS agent and are also able to win against humans. But after conducting the research thoroughly, it can be noticed that the combination of MCTS with Decision Tree isn't able to win as many games and therefore resulting in a weak agent. The resulting MCTS is still able to win a lot of games against the classifier and the majority of games resulting in a draw. It is because of MCTS nature of searching through all possible move combinations, and selectively picking is the best possible move whereas in case of Decision Tree Classifier, the best move is made after selectively searching few combinations of moves. Though the combination of MCTS and Decision Tree is not as impressive, it is still believed that the combination of reinforcement learning with supervised learning can still produce remarkable results and is the future of artificial intelligence in games.

References

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

[2] S. SHARMA, "Monte carlo tree search," in *Towards Data Science*, 2018.

[3] B.-D. Lee, "Enhanced strategic monte-carlo tree search algorithm to play the game of tic-tac-toe," *Journal of Korea Game Society*, vol. 16, no. 4, pp. 79–86, 2016.

[4] B.-D. Lee, "Monte-carlo tree search applied to the game of tic-tac-toe," *Journal of Korea Game Society*, vol. 14, no. 3, pp. 47–54, 2014.

[5] M. C. Fu, "Alphago and monte carlo tree search: the simulation optimization perspective," in *2016 Winter Simulation Conference (WSC)*, pp. 659–670, IEEE, 2016.

[6] D. B. Fogel, "Using evolutionary programing to create neural networks that are capable of playing tic-tac-toe," in *IEEE International Conference on Neural Networks*, pp. 875–880, IEEE, 1993.

[7] 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.

[8] H. Finnsson and Y. Björnsson, "Simulation-based approach to general game playing.," in *Aaai*, vol. 8, pp. 259–264, 2008.

[9] Z. SALLOUM, "Monte carlo tree search in reinforcement learning," in *Towards Data Science*, 2019.

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

Appendix

GitHub Link: https://github.com/gangulyaritra/CE888-7-SP-CO/tree/master/Assignment

CE888: Data Science and Decision Making

Assignment 2 Feedback Sheet

Registration Number: 1906467

Comments

Abstract and Style: does it capture the essence of the paper? Does the style follow a recognised journal?

*The style follows an IEEE template as required.

The abstract is too long and it should not include references or the organisation of the paper (that's something you can include at the end of the introduction instead). There's no mention of the results or conclusions reached in the project.*

Introduction: is the project motivated adequately?

The motivation is adequate, but it is missing references to sustain some of the claims made.

Background: is the background reading extensive?

The background section consumes too much space of the paper with a relatively low number of references included. Too much space is given to explain MCTS.

Methodology: what methods is the project using?

The methodology (combined with the Experiments section) gives enough information about the methods and procedures used in the project, but it's not specified whether the itermax parameter was kept the same for both players or not.

Experiments: what is the experimental setup?

*The experiments explore only the effect of the DT vs MCTS, but not subsequent iterations of the DTs versus each other.

The table at the end of page 4 is showing the same information as the figure that follows it, so you should include only one of them.

You mentioned in your literature survey that the first player in OXO has a higher chance of winning. Did you explore the effects to start first/second in the games? It might explain the results in figure 4 if the MCTS player was always the first one to play.*

Discussion/Conclusion: final remarks and lessons learned

The discussion and conclusions are in line with the results obtained, but there are no suggestions for future work.

How much depth was achieved in the project? Has the project question been sufficiently answered?

The experiments are not as exhaustive as expected, but the paper represents a decent effort to answer the question asked by the project brief.

How well organised, documented and coherent is the code base?

*The code is organised in folders. The main code base is provided as a series of notebooks, leading to code repetition.

There is no clear readme file explaining what's included in each of them or how they are linked.

Unused code has not been removed (e.g., Nim, Othello).*

Any quality control present (e.g., unit tests)?

Prints.