Playing Hanabi with a modified Information Set Monte Carlo Tree Search bot

line 1: 1st Peter LLoyd Gent   
Department of Computer Science and electronic engineering  
Chelmsford, UK

plgent@essex.ac.uk

Uzair Walajahi  
Department of Computer Science and electronic engineering  
Colchester  
uzairwalajahi@gmail.com

***Abstract*— This paper is interested in the development of an ISMCTS bot for the purpose of playing the 4-5 player game of Hanabi. The game presents unique challenges that can only be solved by a rule based approach or an ISMCTS approach because of players blind hands. Adapting this through a filtration process for selective rollout has yieled promising results.**

# Introduction (*Heading 1*)

This paper focuses on creating a bot to play the 4-5 player game Hanabi. This paper will cover a literary review on the rules of hanabi, and then cover the background of Information Set Monte Carlo Tree Search (ISMCTS). It will then cover the code that has been implemented, before reviewing the results of developing the ISMCTS bot further.. Finally, the performance of the bot will be evaluated and concluded, along with suggestions for further development.

# Literary review

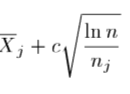
## The card game Hanabi is described with a full ruleset by the online board game retailer regatuljocurilor.ro [1]. It is a 2-5 player game where the goal is to build a full set of fireworks in red, blue, yellow or green suites in numerical order. When the game starts each player starts with either 5 of 4 cards depending on the number of players, and with 8 blue and 3 red counters to one side. A unique part of the ruleset is that in Hanabi players are prohibited from seeing their own cards. Therefore the game revolves around three actions: giving clues to other players on the cards in their hand, which discards a blue counter [2]; discarding cards, which gives back a blue counter to allow any player to tip [2]; and playing cards with the possibility of completing or starting a set in the first case or in the second case discarding a card and depleting a fuze token. If all tokens are expended the game is over as a total failure. It can also end in total success if all five fireworks are completed in order. The third option is the grey area that lies between total success or failure. This occurs if a player draws the final card from the deck, initiating a sudden death round is initiated where all players get an extra turn and then the scores are tallied up.

The biggest challenge for a MCTS approach to the game of Hanabi is the fact that the game world is only partially observable. This is because each players hands are hidden respectively for each player. This is where information set monte carlo tree search (ISMCTS) [3] can be applied to work around problems of partially observable moves or hidden information that this paper is approaching the problem of Hanabi. For this purpose a ISMCTS approach has been selected.

# Background

## In Walton-Rivers lectures on reinforcement learning monte carlo tree search is covered as one of the techniques for reinforcement learning [4]. A reinforcement learning algorithm such as MC and MCTS is different from supervised or unsupervised learning because it balances long term rewards with short term results through rewards given to the algorithm for successful outcomes. In the model for reinforcement learning a bot makes actions on the environment and then observes the change before planning a move to do next. At each stage the bot receives a reward that is accumulated so the bot attempts to maximise reward over the long term [4]. With Monte carlo methods this means looking down an avenue of possibilities by simulating a particular set of outcomes until a terminal node is reached. This returns a reward at the end, which back propagates to the root node [5]. In total there are 4 actions that any MCTS must perform and these are: Select; expand; simulate; and backprop [5];

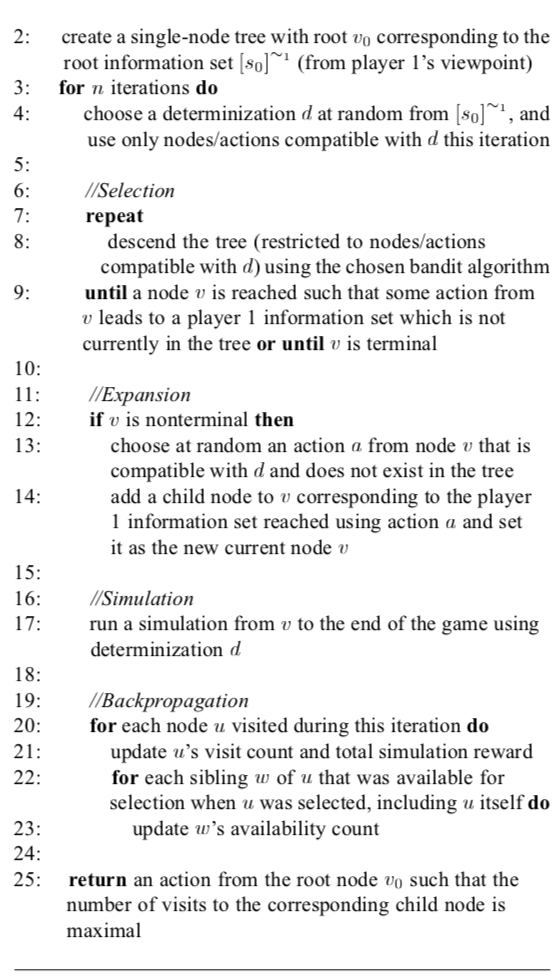
when beginning a new MCTS the search skips the select stage and instead expands one of the nodes at random. It then simulates a possible outcome for one of these avenues as described. On the next iteration it then uses select with upper confidence bound (UCB) to balance greedy actions of taking an already explored set of nodes with a virgin node that hasn’t been explored before. The UCB value is calculated from:



The UCB is calculated every time the search needs to traverse the tree. The first variable represents the reward for traversing down a particular set of the tree which is balanced by a constant that increases the reward of exploring new unseen nodes. The value N represents the number of nodes explored. When both sides of a tree have been explored the UCBs are calculated for both sides and then the child node with the highest UCB score is picked. In ISMCTS the process is different.

Firstly, In ISMCTS instead of states, sets of information are considered as possible nodes [3]. In cowling’s paper the first set of approaches he takes are labelled “determinized UCT” approaches, with two varieties: single observer and multi observer. For the purpose of this paper single observer ISMCTS (SO-ISMCTS) is sufficient for a Hanabi use case. In this case each node in the tree correspond to information sets from the root player’s point of view, and edges correspond to actions.

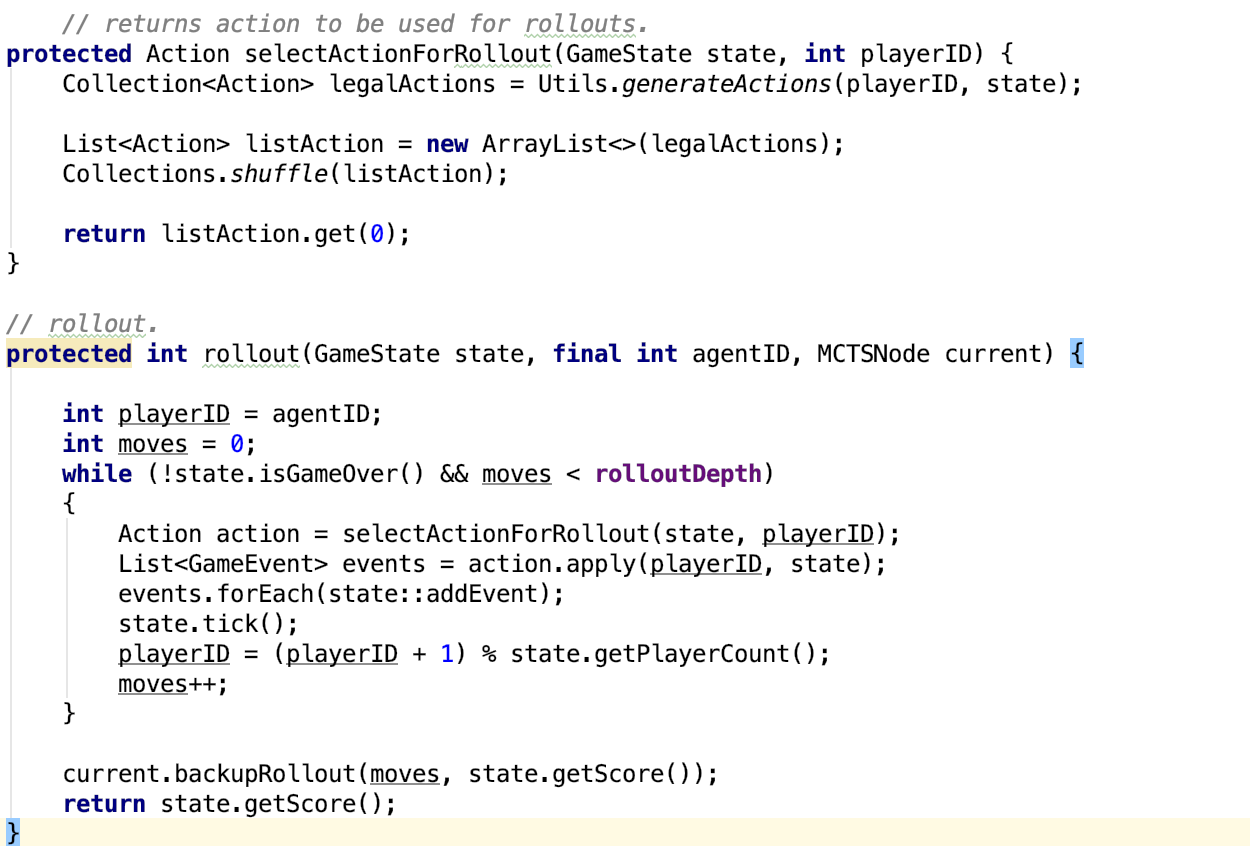
Here is the pseudo code for SO-IMCTS implemented in this paper [3]:



The bot has been implemented by walton-rivers et al has rather poor time performance. This is because it looks at any of the possible moves to make for telling another player rather than taking precedence for moves that the player can make to complete a suite.

# IIII Techniques implemented

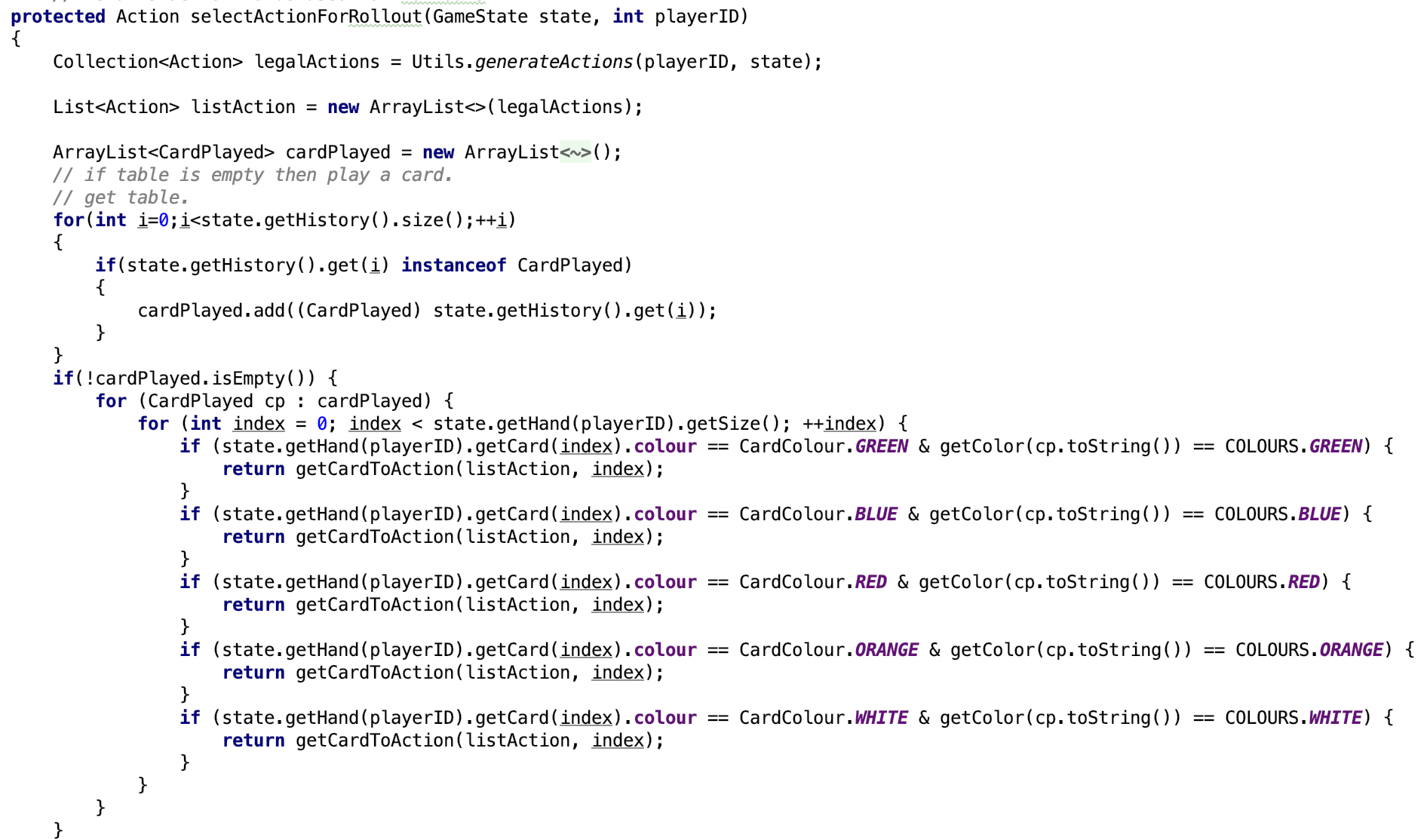
In the original code there are two functions responsible for simulation or ‘rollout’. The first of these two ‘selectActionForRollout’ gets a list of possible actions and gets a random (shuffled) action to be explored in the rollout process.



#### fig 1. code for random rollout [6]

The rollout process then looks through the random actions, applying an event each time to consider what the next action will be. This process is repeated for each of the players in the game at the beginning of each players turn. The main area for improvement in this area is to not use random nodes and instead look for nodes that would complete information for another player or complete or add to a suite on the table.

In the process of creating a solution the main problem has been that it is nigh on impossible to get a picture of what cards lay on the table. This has hindered development of a solution that improves upon the original solution in the aforementioned way as it’s impossible to match cards in the players hand with that on the table. The only middle solution is to look through the history of plays in the game, which has proven cumbersome and time consuming to do and can yield incorrect plays that went into the discard pile because they did not match the cards on the table.

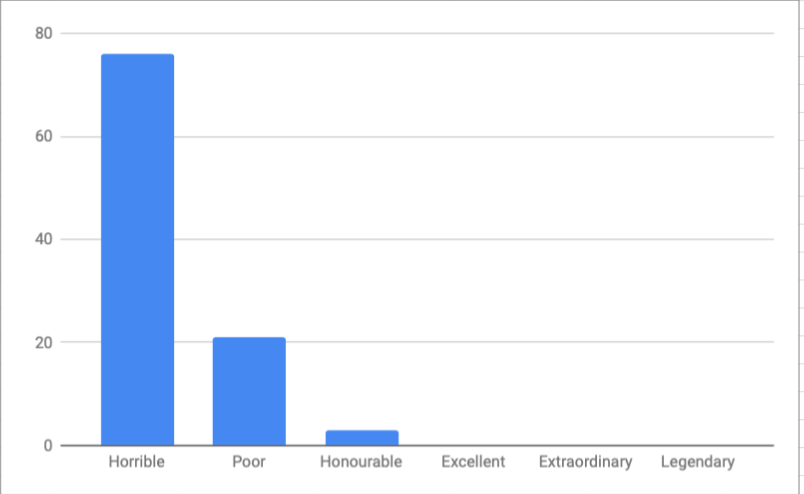


#### Fig 2. filtering returned actions with a check for suite on table and in hand.

Unfortunately the pressures of time has meant that a non optimal solution has been created (above). With further development it might be possible to filter the selected nodes for rollout with a higher fidelity or granularity in terms of behaviours for discarding cards, or playing cards with a reasonable certainty that the cards will fit those on the table. It could also be possible to implement a rule based agent for moves that involve the players hand and MCTS for the rest or vica versa.

# V Experimental Study

In the comparison 100 games with the original MCTS were run and the majority of the performance ranged between Horrible and poor with 76 of the games fitting into the Horrible rating as a near or total failure (see fig4). In only a minority of games a poor or honourable ratings were achieved.



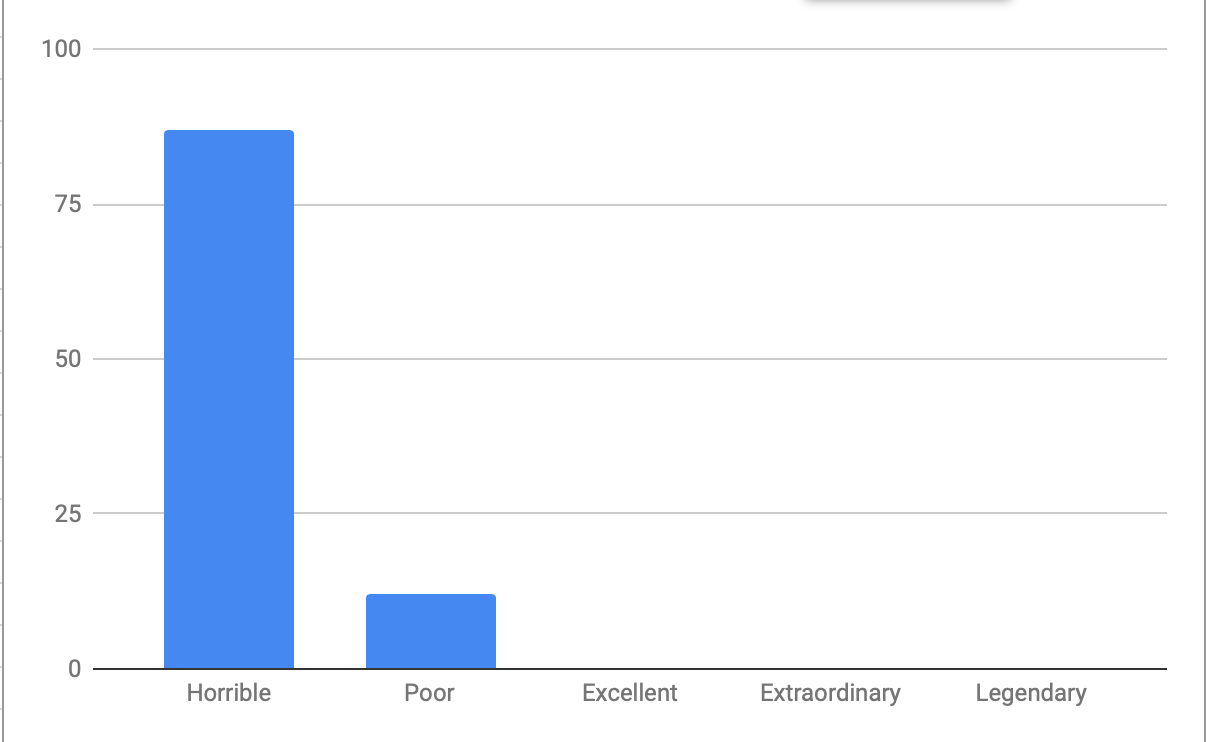
#### fig 3. performance of ten runs of original solution

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Horrible | Poor | Honourable | Excellent | Extraoridinary | Legendary |
| 76 | 21 | 3 | 0 | 0 | 0 |

#### fig 4. Table of results for regular monte carlo search in original solution

##### 

##### 



#### fig 5. new solution performance.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Horrible | Poor | Honourable | Excellent | Extraoridinary | Legendary |
| 87 | 12 | 1 | 0 | 0 | 0 |

#### Fig 6. performance table for new solution.

In the new solution the performance is actually marginally worse after filtering actions for the select function. This could yet be improved by filtering further for card numbers and not just a suite for the rollout and simulation. Also yet to be implemented is a way to select a discard action always when the amount of information or blue counters is 0 to yield more information for players, more routinely.

The main problem encountered for MCTS in both cases (new and old solutions) is that there is a three strikes and you are out rule. With this it appears that MCTS performs poorly because it isn’t careful enough to play cards that fit with the picture on the table, meaning that most games appear to result in a horrible rating in both instances. The MCTS implementation also appears to be disjointed in a sense. This is because the majority of the actions that MCTS can perform have no effect on the information that the player receives about their cards or plays to that matter.

##### Conclusion

In conclusion there is much room for improvement to the new solution. It appears that further filtration of the selected nodes for rollout has not proved to be as successful as it initially appeared, however due to a tight time constraint this could be forgiven as a result of running out of time with the filtration process. With further development it may be possible to increase the performance of the solution by filtering further and actually reworking the rollout process as a whole or by using a combination of rule base and MCTS solutions.

##### References

[1] *Regatuljocurilor.ro*, 2018. [Online]. Available: http://www.regatuljocurilor.ro/product\_extra\_files/HanabiRules-EnglishTransofFrench-1page.pdf. [Accessed: 29- Nov- 2018]

[2] Roll for crit, 2015.

[3] P. Cowling, E. Powley and D. Whitehouse, "Information Set Monte Carlo Tree Search", *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 120-143, 2012.

[4] J. Walton-Rivers, "Reinforcement Learning", The University of Essex, 2018.

[5] J. Levine, *Monte Carlo Tree Search*. 2017.

[6] J. Walton-Rivers, "src/main/java/com/fossgalaxy/games/fireworks/ai/mcts/MCTS.java · master · IGGI students / hanabi", *GitLab*, 2018. [Online]. Available: https://git.fossgalaxy.com/iggi/hanabi/blob/master/src/main/java/com/fossgalaxy/games/fireworks/ai/mcts/MCTS.java. [Accessed: 12- Dec- 2018].

##### 

##### 

##### 

##### Bibliography:

[1] J. Walton-Rivers, "Reinforcement Learning", The University of Essex, 2018.

[2] J. Walton-Rivers, "Reinforcement Learning II", The University of Essex, 2018.

[3] Fullstack Academy, *Monte Carlo Tree Search (MCTS) Tutorial*. 2018.

[4] P. Cowling, E. Powley and D. Whitehouse, "Information Set Monte Carlo Tree Search", *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 120-143, 2012.

[5] J. Levine, *Monte Carlo Tree Search*. 2017.

[6] Roll for crit, 2015.

[7] *Regatuljocurilor.ro*, 2018. [Online]. Available: http://www.regatuljocurilor.ro/product\_extra\_files/HanabiRules-EnglishTransofFrench-1page.pdf. [Accessed: 29- Nov- 2018]

[8] J. Walton-Rivers, "src/main/java/com/fossgalaxy/games/fireworks/ai/mcts/MCTS.java · master · IGGI students / hanabi", *GitLab*, 2018. [Online]. Available: https://git.fossgalaxy.com/iggi/hanabi/blob/master/src/main/java/com/fossgalaxy/games/fireworks/ai/mcts/MCTS.java. [Accessed: 12- Dec- 2018].