RED INDICATING PREVIOUS SUBMISSION

CS 4701: Practicum in Artificial Intelligence

Spring 2020

Project Proposal

Rodnelps

Team Members: Arthur Tanjaya (amt333), Nicholas Yuwono (nly7), William Sumendap (wls63)

AI Keywords: Game playing (including game tree search), reinforcement learning

Application Setting: Our project is a game playing program for the game Splendor by Space Cowboys. The AI will play a CLI version of this game.

Section 1:

Our project is a game playing program for the game Splendor by Space Cowboys. In doing this, we will create a CLI setup for the game whereby the AI can interact with the game as a human would. We will use heuristic evaluations to determine favourable outcomes and drive the game that way.

Section 2:

We will use heuristic evaluations to determine favourable outcomes and drive the game that way. This means that the AI should be able to balance between short and long term gains in the game and to capitalize cards in the game as the game approaches its end. To that end, we currently have two ideas in mind:

1. Game tree search. Since this is an AI for playing an adversarial, turn-based game, we will be searching through the game tree in order to determine the best move possible (in the minimax/maximin sense). In order to do this, we must have an efficient evaluation function for any particular game state. Initially, this function will be based on (a) the number of gems and points accrued (more is better) and (b) strategies that human Splendor players recommend (i.e. us).
2. However, there is a level of non-determinism to the game, because the order of the cards in the deck is unknown to players. To remedy this, we will allow the AI to tweak its evaluation function based on (a) the expected probabilities of the card(s) to be drawn and (b) its past experiences playing against itself and/or other players (hence the reinforcement learning part).

Section 4:

We will let the game play against various players. We will also let the game play against the official mobile application's built-in AI. Overall, due to the very weak AI that the developers implemented as the mobile app game's AI, our goal is to create a better AI than the default AI. A nice-to-have would be an AI that plays better than an average Splendor player. We plan to recruit (if possible) other 4701 students and also our friends that are willing to test the system. Evaluation of the AI's strength is, of course, by its win-rate.

While total points accumulated at the end of a game is not in itself always a good measure of AI strength (as sometimes the only goal of the AI is to win and the win margin does

not matter), we will use the average number of accumulated points as an evaluation function to test the AI when it plays a game against itself.

Section 5 (TIMELINE):

By February 13

- Finish and submit this document

By February 25 (Up to and during February break)

- Finish setting up the game engine (i.e. implement all the rules of the game), CLI and input verification (i.e. reject illegal moves)

By March 11 (or the week of)

- Code up a rudimentary AI that can interface with the game engine (i.e. it's fine for it to make stupid moves, but not illegal moves)

By March 24

- Finish coding up the game tree search algorithm and evaluation function, even if it's rudimentary (in particular: make sure it does not error out)
- Write up the status report

By April 5 (Up to and during spring break)

- Finish implementing a working AI and have a non-garbage evaluation function
- Begin testing during spring break and surpass the super-sucky default app AI
- Ask around for test volunteers

Up to April 22

- Have AI be good enough to consistently beat us

- Add in reinforcement learning portions of the algorithm

Up to May 5

- Polish up the AI
- Write and finish final report
- Prepare project presentation
- Do last minute things that will inevitably come up

Section 6:

1. Standard Java libraries
2. Labor in the form of friends for testing the AI
3. Online strategy guides for Splendor
4. An actual physical copy of Splendor to play

Since our user interface will be no more than a CLI, we will implement this ourselves.

NEW MATERIAL HERE:

Progress

We decided to explore the use of Monte Carlo Tree Search as a heuristic search tool for our move generating processes as it is what we see as being most in-line with the expectations of this course (limited use of ML, focus on AI strategies). To compare our MCTS results against, we also created a "baseline" AI which produces its moves based on a greedy, goal-focused approach that evaluates each hand locally and deterministically.

Within the MCTS approach, we also tried multiple strategies in our preference for picking child nodes. Specifically, we created 4 different MCTS AIs, namely:

MCTSv0AI, which picks its children randomly (we expect this to be the worst MCTS AI on average)

MCTSv1AI, which picks the child with the most number of expected wins

MCTSv2AI, which picks the child according to Kocsis and Szepesvári's exploitation vs exploration balance formula which can be found on (https://en.wikipedia.org/wiki/Monte_Carlo_tree_search) and can be seen below.

the game tree the move for which the expression $\frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}}$ has the highest value. In this formula:

- w_i stands for the number of wins for the node considered after the i-th move
- n_i stands for the number of simulations for the node considered after the i-th move
- N_i stands for the total number of simulations after the i-th move run by the parent node of the one considered
- c is the exploration parameter—theoretically equal to $\sqrt{2}$; in practice usually chosen empirically

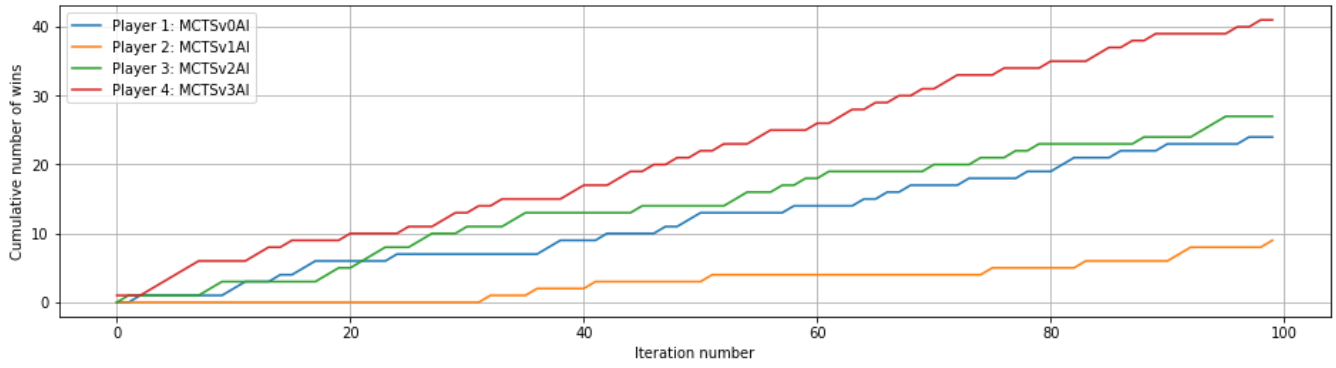
(We expect this to be the strongest MCTS AI on average)

MCTSv3AI, upon the authors of this paper actually playing the game, we realized that card-reservation is a move seldom used and is often counterproductive. Hence, we repeat

MCTSv2AI and enforce an additional condition whereby we exclude card reservation from the choice of moves that our AI can make.

Comparison Between MCTS Results

Being the evidence-driven scientists we are, we let these four MCTS approaches play against each other in a 4-player game and observe their progress. We have imposed a limitation whereby each move of each AI has to be taken in at most 1 second to reduce computational load (early termination is one of the advantages of using MCTS). We then let these simulations run 100 games and observe the results which we can see in the graph below.



Player 1: MCTSv0AI wins: 24

Player 2: MCTSv1AI wins: 9

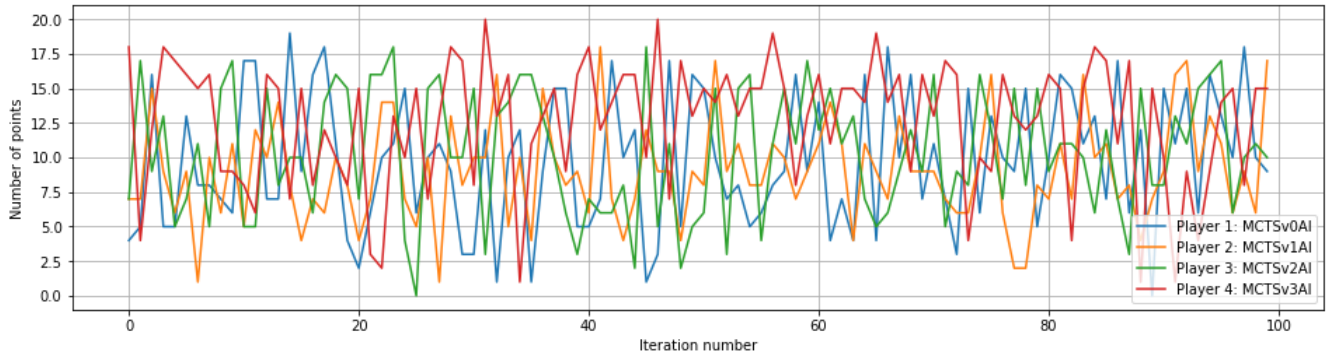
Player 3: MCTSv2AI wins: 27

Player 4: MCTSv3AI wins: 41

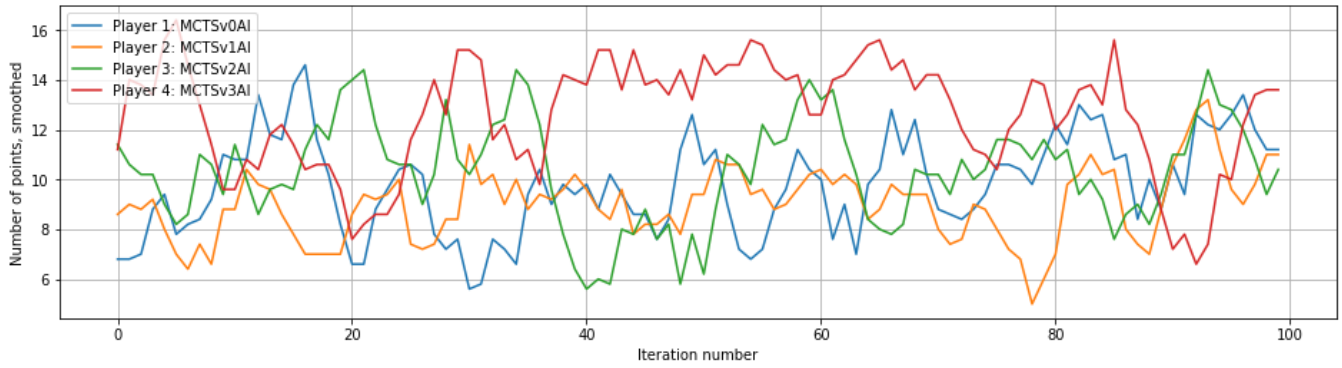
Seen on the graph above, we have MCTSv3AI as our clear winner with a 41% win rate. This was surprising as we have imposed an artificial condition to the AI and our other MCTs approaches (especially v2) should have fared better by recognizing that the reserve move should

seldom be used but can be used occasionally to gain a slight advantage. V3's handicap should, therefore, put it at a disadvantage compared to v2.

Apart from win rates, we also took a keen interest in the different game metrics each AI might have. We start off by observing the number of points that each AI gets in each game. While this is certainly an imperfect proxy for determining win-rate due to the AI's goal of maximizing wins and not its point count, we sought to see if there is any correlation between the number of points an AI gets and its win rate. The graphed result can be seen below

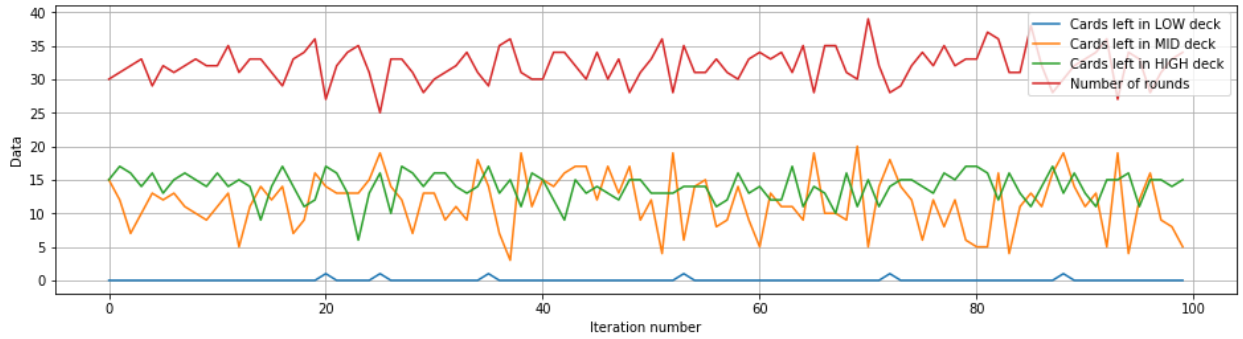


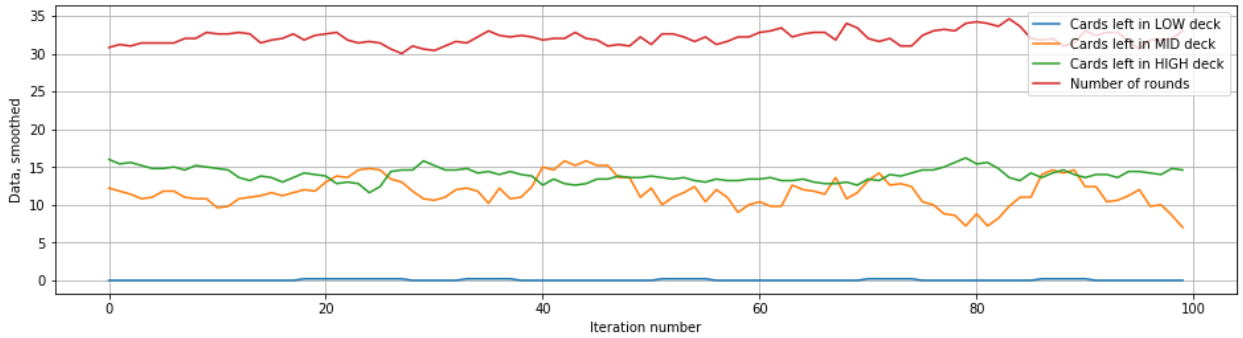
As the graph was difficult to decipher, we smoothed the points



In playing Splendor in person, we notice that it is a generally favorable strategy to take up low deck cards in the early game to build a gem reserve to be able to purchase the more expensive cards in the future. It is also sometimes a favorable move to take a low-deck card even in the later games when moves are more limited in choice due to the difficulty in purchasing

more expensive higher deck cards or due to limitations in the number of coins up for grabs. We would, therefore, take up low deck cards to increase our gem reserve. Based on empirical observation, one would then usually end the game with between 7-12 low deck cards in each hand. So we sought to see the number of cards on average for our AIs





To our surprise, we can see that it is exceedingly rare for low deck cards to be taken up in favor of mid or high deck cards. This result is indicative of a flaw in our MCTS strategy as taking the low deck cards should occur much more frequently than the current rate.

Comparison With Baseline

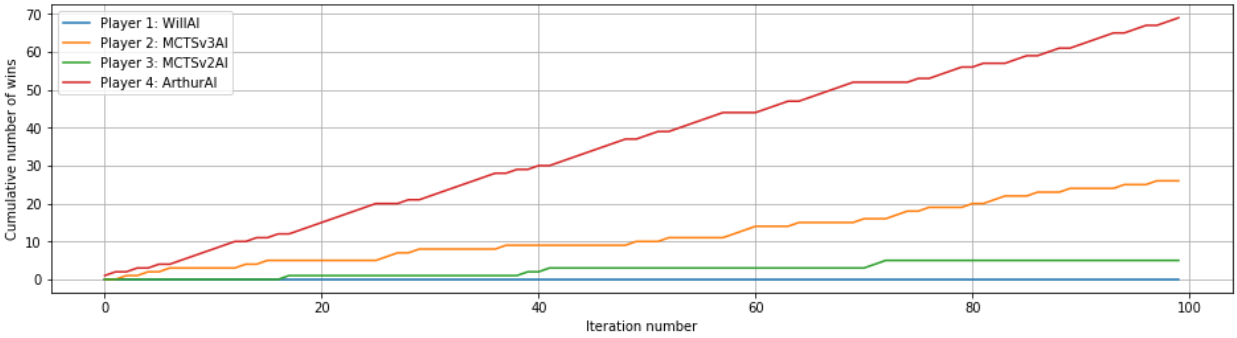
As intended, we compared our results with our baseline (a greedy algorithm) and sought to see the performance improvements our MCTS can offer. To do so, we run our simulations

again with 4 different AI. Two of these AIs rely on mere greedy approachest whilst the other two are MCTSv3AI and MCTSv2AI.

The first two AIs are designed as such:

> WillAI chooses a card at random and then locks the card as a target. The AI will then attempt to get that card by taking the tokens (or free cards) necessary to get the target card. If the target card is lost to another player (either purchased or reserved by another player) it finds another target card.
>
> ArthurAI by choosing a card to immediately purchase at (if possible, with higher tier cards given priority but randomly chosen within the same tier). If no card purchase is possible, the AI takes 3 distinct coins at random (if possible), if that is also not possible, it takes 2 coins at random. Lastly, if all those are not possible, we shall reserve a card at random.

The results of the simulations are shown below:



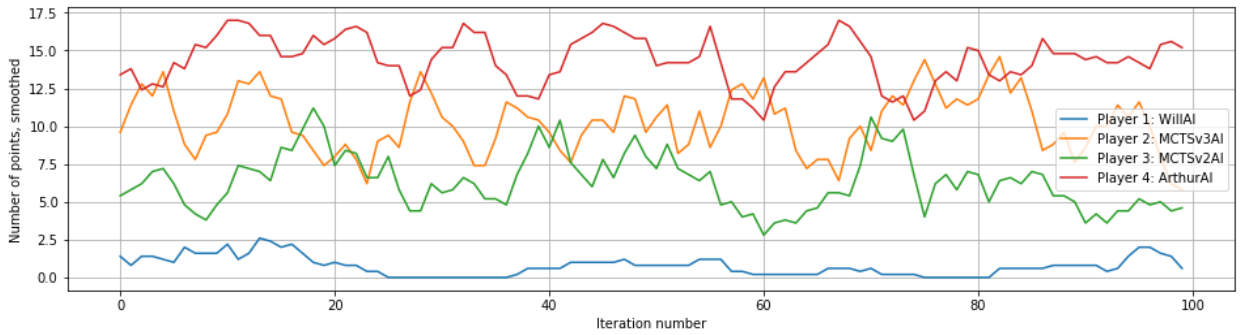
Player 1: WillAI wins: 0

Player 2: MCTSv3AI wins: 26

Player 3: MCTSv2AI wins: 5

Player 4: ArthurAI wins: 69

As seen on the graph above, the ArthurAI and its greedy approach win an overwhelming number of games (69%). This result is a rather disappointing one for our MCTS as our arguably more "complex" approach has lost to one that is based on local optimality. Looking at the number of points too, it is clear that ArthurAI takes the lead.



We, however, are quite pleased that our MCTS results consistently outperformed the hone-in greedy approach as implemented with WillAI and this shows that MCTS can offer some promising results.

Lastly, while we initially also intended for the game to be played against the Splendor App's AI, the lack of an API and the difficulties surrounding automating this process prevented us from doing so in a time-efficient manner and thus we decided only to compare with a greedy algorithm (which is also home the App's AI works although the exact approach is slightly different)

Conclusion and Next Steps

In our next milestone, we will explore more options within our MCTS approach. We remain committed to the approach as we believe that it should be able to yield results better than our greedy approach. The unfavorable outcomes of our MCTS approaches could be because of the 1 second time limit given to our MCTS which makes the outputted moves less optimal than possible. This could also be why MCTSv3AI has an advantage over MCTSv2AI as there are fewer moves to consider and therefore more options can be explored in the Monte Carlo approach.

To fix this, we are considering increasing the time limit. If we still find our MCTS approach too computationally expensive (as our current 1 second limit translates to approximately 15 minutes of runtime per 100 games on a Macbook Pro), we can resort to cloud computing (AWS, Google Cloud) as a means to reduce runtime whilst also exploring more options within the MCTS children.