NERS 544 Final Project Proposal: Monte Carlo Tree Search Applied to Simple "Filler" Game

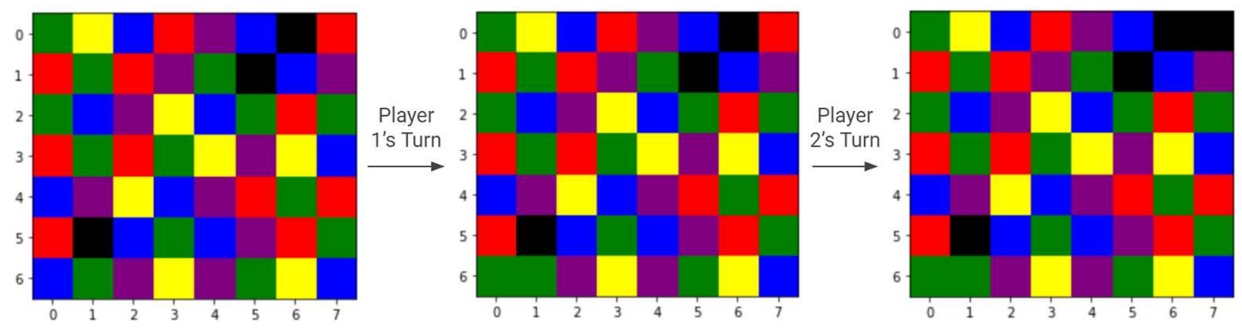
Sumit Basak, Arham Jain, Daniel Manwiller

March 26, 2021

We are proposing a project to apply the Monte Carlo Tree Search (MCTS) algorithm to create an AI for a simple game called Filler. This game provides a simple yet scalable complexity for applying the MCTS algorithm. Details of the game, the MCTS algorithm, our proposed implementation, and how we will evaluate our AI are provided in the following sections.

Filler Game:

This game consists of a rectangular board with tiles of different colors as shown in Figure 1. In its standard form, the board size is an 8x7 tiled grid and has 6 distinct colors. This is a 2-player game and the objective of the game is to capture more tiles than your opponent. Player 1 starts with the tile in the bottom left corner and Player 2 starts with the tile in the upper right corner. The players alternate turns and on each turn, the player chooses to change his color to one of the four remaining colors not already occupied by either the player or his opponent on that turn. Each player captures new tiles by changing his color to be the same color as any tiles touching the tiles he has already captured. When a player changes his color, all his captured tiles change color as well. Since neither player is ever able to choose the same color as his opponent, a tile can only be captured once. The strategy in this game is to strategically pick your color to maximize your captured tiles not only in the current move but also in future moves by boxing off regions or expanding toward regions of same colored tiles. A player might also strategically choose his color to keep his opponent from choosing that color and delay his opponent from capturing a large number of tiles.



a) Initial game board setup b) Player 1 changes his color to green c) Player 2 changes his color to black

Figure 1: Shows the initial game board and one round of game play. Note that Player 2 chooses black to capture the tile which sets him up for choosing blue next turn and capturing two tiles. Choosing black also blocks Player 1 from choosing it on his next turn and setting him up on the following turn to capture two purple tiles.

This game provides a simple setup and gameplay while also having a relatively complex strategy given the need to look many turns in the future to determine the optimal move. The complexity of the game can be increased by increasing the board size from the standard 8x7 grid to be as large as desired. This flexibility allows the standard game play to provide an interesting and relatively simple application for the MCTS algorithm while the ability to scale the board size to be much larger provides a very reasonable application for the MCTS algorithm where an exhaustive decision tree or other simple decision tree methods would not be feasible.

MCTS Algorithm:

The MCTS algorithm provides a heuristic search algorithm for some defined decision process and has been applied to many board games such as checkers, chess, and most famously, the game of Go [1]. While there are many different decision tree search algorithms, MCTS leverages the power of random Monte Carlo sampling to efficiently play out simulated games to the end many times to build a weighted decision tree. On each run, a game state is chosen based on the current game state and the outcomes of previous runs. Once a game state is selected, a game is played out with random choices. The decision tree is then updated based on the outcome of each iteration. The algorithm can then be run with a specified number of iterations before the decision with the highest weight is selected. The MCTS algorithm can be broken down into 4 distinct phases as outlined in Figure 2.

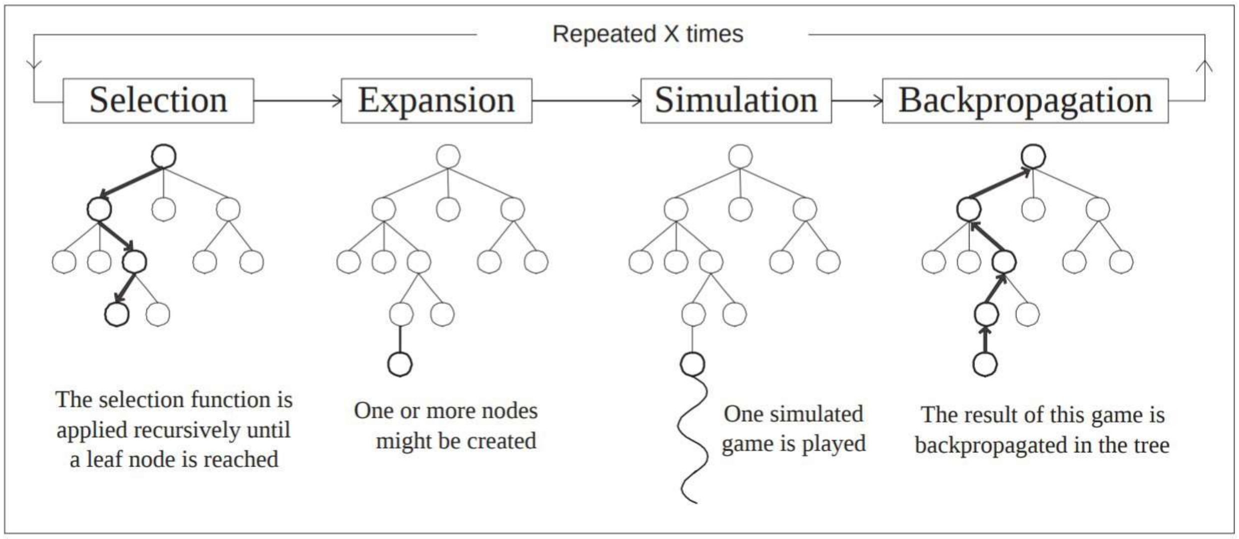

Repeated X times
Selection
Expansion
Simulation
Backpropagation
The selection function is applied recursively until a leaf node is reached
One or more nodes might be created
One simulated game is played
The result of this game is backpropagated in the tree

Figure 2: MCTS Algorithm. [2]

Phase 1: Selection

During this initial phase, the game state to explore is chosen by traversing the decision tree of previously explored states but with a specific weighting on each node. This selection process allows the exploration of game states to favor those that are more likely to be successful. The node weighting incorporates the number of successes for previous trials passing through that node and the number of total trials passing through that node. This allows the algorithm to have a tunable parameter for how often it expands on more likely outcomes to explore vs. exploring less tested outcomes. The weights for each node are then updated at the end of each iteration. The selection for each iteration is based on traversing down the tree until a leaf node is reached by always choosing the child node with the highest weight. Figure 3 gives the equation for node weighting.

$$\frac{w_i}{s_i} + c\sqrt{\frac{\ln s_p}{s_i}}$$

- w_i : this node's number of simulations that resulted in a win
- s_i : this node's total number of simulations
- s_p : parent node's total number of simulations
- c : exploration parameter

Figure 3: Node weighting equation. [3]

Phase 2: Expansion
This is a fairly straightforward step in which we add a randomly chosen child to the leaf node chosen in the selection phase. This node represents a specific game state and is what is used in the simulation phase.

Phase 3: Simulation
During this phase, the chosen game state from the expansion phase is then simulated till the end of the game using random moves by each player. No part of this simulation is stored in the search tree. Only the result of winning or losing matters for updating the tree in the backpropagation phase.

Phase 4: Backpropagation
The result from the simulation is then backpropagated through the decision tree to update all relevant nodes affected by this result. This involves updating the weights on all nodes that are direct ancestors of the node added during the expansion phase as well as that new node itself. These new weights are then used to select a path to explore in the selection phase during the next iteration.

The MCTS algorithm is run for a set number of iterations and the move with the highest likelihood of success, as determined by the Monte Carlo simulations, is then chosen as the AI's next move. This algorithm provides an efficient way to simulate game outcomes and determine the best move for games where an exhaustive search or other simpler decision tree search algorithms aren't feasible. For the game of Filler, an exhaustive decision tree search would be quite feasible for the standard board setup. However, this game provides a simple application for this algorithm and provides the flexibility to scale the complexity by increasing the board size and thus making MCTS a very reasonable decision method.

Proposed Implementation:

We plan on implementing the game play and the MCTS algorithm from scratch using python code. This involves writing the code to set up the board, updating the board based on the chosen move for each player, and properly displaying the board at each turn. This also involves writing the code for the MCTS algorithm with its 4 phases and running the algorithm for a given number of iterations to find the best move. We then would have to write code to test our MCTS based AI vs. a greedy approach to evaluate the performance of our AI and show that it has been implemented properly. We also have an additional goal to implement a user interface for a 1-player gameplay against our AI.

While implementing all of this from scratch will require a fair amount of coding, the game and the MCTS algorithm are both fairly straightforward in their structure. We believe this should be a feasible project and one that demonstrates our knowledge of Monte Carlo methods and how to apply them to interesting problems. MCTS is not related to any area of research for any of the group members and to the best of our knowledge, it has never been applied to the game of Filler.

Evaluation and Deliverables:

To evaluate our implementation of an MCTS based AI, we plan on setting our implementation to play the game of Filler against a greedy approach where it always selects the color that will gain it the most tiles on that move. Given the nature of the game, a model that looks ahead to the potential outcome of future moves should be far superior to a greedy approach. However, due to the random nature of the board generation and its relatively small size, it is possible that the greedy approach could win because the board was set up in its favor. So, we also plan on testing our MCTS based AI against a greedy approach on a much larger board to see how much better it can perform. This larger board is also a much more reasonable application for the MCTS algorithm as an exhaustive search or other simpler decision tree methods would not be feasible for a large enough board.

Our deliverables will be as follows:

- All relevant code for implementing the Filler game, MCTS algorithm, and evaluating our AI vs. a greedy approach.
- The results for evaluating our MCTS based AI vs. a greedy approach for both the standard board size and a much larger board size.
- An application for playing 1-player Filler against our AI to test how a human performs against our AI.
- A report expanding on this proposal with our results and any other steps we encounter while implementing our MCTS based AI.

References:

[1] "AlphaGo: The story so far", Deepmind, 2021. [Online]. Available: https://deepmind.com/research/case-studies/alphago-the-story-so-far. [Accessed: 23- Mar- 2021].

[2] Guillaume Chaslot et al. "Monte-Carlo Tree Search: A New Framework for Game AI." In:AIIDE.2008 (cit. on p. 12).

[3] M. Liu, "General Game-Playing With Monte Carlo Tree Search", Medium, 2017. [Online]. Available: https://medium.com/@quasimik/monte-carlo-tree-search-applied-to-letterpress-34f41c86e238. [Accessed: 23- Mar- 2021].