**Implementation Explanations:**

The MCTS algorithm consists primarily of four distinct steps; Selection, Expansion, Simulation, and Back-propagation. The following implementations each have different strategies for addressing these steps, and these will be detailed below. First, an explanation of the overall design concepts and how the algorithms run will be provided.

Games are played out through a Controller class, which handles the players and the rounds. All implementations initialize and execute when the Controller asks that player to play a card.

A common issue with random simulations and the MCTS is when to randomly assign cards for a play-out. The last implementation addresses this challenge differently from the first two.

In implementations 0 and 1, a single search tree is constructed whenever the player is asked to play a card. This search tree is then explored, with random play-outs being performed at each node. These play-outs are based on opponents receiving random assignments of hands from the pool of remaining cards *at the beginning of each simulation, or at each node.* The results of this tree are then calculated and the best move made is returned.

A node in these trees is equivalent to a play. For example, the root node will have thirteen children at the beginning of a round, because there are thirteen possible cards to play in the hand. This approach is easier to implement and appears somewhat faster – the downside is no trick information is stored within the nodes, and therefore the nodes themselves are much fewer but not as informative.

In contrast, the third implementation constructs multiple search trees when asked to play a card. A search tree is created, and a random deck is dealt *at the root node*, and is used for the rest of the round. The search tree is fully explored, and a recommended card to play is returned. The most common card returned from the various trees is returned as the move to be played.

A node for this tree contains much more information, since the deck being played with is static for that instance of a tree. The nodes can therefore contain information regarding an entire trick, so there is much more of them.

Each implementation is given an average time of 10 seconds to play a game (respond to thirteen hands), for testing purposes. This could be increased in future work, but for comparing these implementations 10 seconds seems sufficient to still reach a significant depth and number of play-outs.

**Implementation 0:**

This is the simplest implementation of the MCTS algorithm for the Hearts application. It is not designed to be fast nor particularly effective – it is simply a basic example of MCTS applied to trick taking card games. As such, is used as a baseline for comparisons between the other algorithms, and is useful for quantifying improvements.

The specific implementation of each method is as follows.

**Selection:**

A basic probabilistic ranking formula is used when selecting a node. An array is constructed of all the possible nodes to select. Aliases of successful nodes are appended to the end of the array based on how little hearts they have managed to collect in their play-outs. The node chosen is selected at random – with more representation of successful nodes, these are chosen more often then others, but a probabilistic element remains to ensure that other nodes are not skipped over. The extra representation in the array is limited to 13 – (average Hearts collected / 2). This is a simple representation, and is expected to perform poorly when compared to UCT.

**Expansion:**

Nodes are expanded based on the remaining plays following the current node. For example, if expanding the node representing the first play, there will be twelve child nodes expanded.

**Simulation:**

A simple face-up evaluation method is used for this implementation. The opponent’s hands are assigned at random based on the pool of remaining cards (as mentioned earlier), and the game is played out using some simple rules. The MCTS player starts the first trick, and all opponents will play their highest card possible unless Hearts (or the Queen of Spades) are present in the trick, in which case they will lay their lowest card. This is again meant to simulate a very naïve player when considering these simulations.

**Back-Propagation:**

The Hearts collected during the play-out and the visit count are recorded on the node, and are added on to all parent nodes.

**Implementation 1:**

This method is similar to the first, with some noticeable improvements to the Selection and Simulation methods.

**Selection:**

Upper Confidence Bound for Trees (UCT) is applied here. The formula is stated below (TO DO, see paper – convert to min/max and explain values of threshold and coefficient).

**Simulation:**

The simulation method is more advanced then earlier, but follows the same execution pattern. The face-up play now considers the other cards currently in the trick, which is particularly useful for the final tricks. In addition, the rules based approach has been improved, with the basic logic below.

If last to play in trick

Play highest card of the suit ELSE

Play Queen of Spades ELSE

Play highest card of suit with fewest remaining cards

If leading the trick

Open with the lowest card of the suit with fewest remaining cards

If 2nd or 3rd in trick

Play highest card of suit that will not win trick ELSE

Play Queen of Spades ELSE

Play highest card of suit with fewest remaining cards

**Implementation 2:**

This last implementation shares many of the same design choices with implementation 1 – the MCTS itself uses the same select, expand, simulate, and back-propagation methods. However, the difference lies in the instantiation of the trees.

When the algorithm is called to make a play, it will construct an array of possible moves (the remaining cards). Each move will be assigned a score based on a number of distinct MCTS operations. While the algorithm still has time, it will construct trees based on random deck deals, explore this tree to some length, and return the play with the highest score.

Since the deals are unique to the trees, as opposed to the nodes, the information stored in the tree is slightly different. Each node contains information on a trick that was played out, as opposed to just a play.

This is a distinctly different style of implementation of the MCTS, but one that shares the same four components of the others.