DPRL Assignment 3

I. Description of MCTS

A. How to explore

Our implementation of MCTS for a game environment consists of four main steps: selection, expansion, rollout and backpropagation.

- Selection: Starting from the root node, which represents the current state of the game, the algorithm recursively selects the child node with the highest value, until it reaches a node that is not fully expanded or is terminal (i.e., the game is over). The value of a node is calculated by a formula that balances the exploitation of nodes with high average reward (Q/N) and the exploration of nodes with low visit count (N). The formula also includes a parameter that controls the degree of exploration (explore_value).
- Expansion: If the selected node is not terminal, the algorithm expands it by generating one or more child nodes, each representing a possible action from the current state. The algorithm then randomly chooses one of the newly created nodes to proceed to the next step.
- Rollout: The algorithm simulates a random playout from the chosen node until it reaches a terminal state. The simulation can use a simple or a sophisticated policy, depending on the problem domain. The outcome of the simulation is a reward value that indicates the quality of the chosen node
- Backpropagation: The algorithm propagates the reward value back to the root node, updating the visit count (N) and the total reward (Q) of each node along the path. This way, the algorithm learns from the results of the simulations and improves its estimates of the node values

B. How to update the tree

By repeating these four steps for a given amount of time or iterations, the algorithm builds a search tree that approximates the optimal decision tree for the problem. The best action to take from the current state is then the one that corresponds to the child node of the root with the highest value.

- It starts from the root node, which represents the current state of the game.
- It selects a node to expand or simulate by following the UCT formula, which balances exploration and exploitation. Exploration means choosing nodes that have not been visited or have low visit counts, while exploitation means choosing nodes that have high win ratios or rewards.
- If the selected node is not terminal (i.e., the game is not over) and not fully expanded (i.e., it has unvisited children), it expands the node by creating a child node for each legal action in the game state.
- It then simulates a random playout from the selected or expanded node until a terminal state is reached. The playout is done by choosing actions randomly from the legal actions in each state.
- It evaluates the outcome of the playout and assigns a reward to the terminal node. The reward is usually +1 for a win, -1 for a loss, and 0 for a draw.
- It updates the information of the nodes along the path from the terminal node to the root node. The information includes the visit count (N) and the total reward (Q) of each node. The Q value is updated by adding the reward of the terminal node, while the N value is incremented by one.
- It repeats the above steps until a predefined time limit or a number of simulations is reached.
- It returns the best action to take according to the search results. The best action is usually the one that corresponds to the child node with the highest average reward (Q/N) or the highest visit count (N).

II. Visualization of game

A. Tree graph

First, we use a binary tree to represent the process of decision since we can only place circles on the left-most column and the right-most column. Second, we start from the root. The state of left-child node means the player of this round chooses the put its circle on the left-most column and vise versa. Lastly, a branching will be end at a terminal state (node) when the game is over. That is, one of the players wins or they draw.

B. Edges

As you can see, there are two different colors of edge, blue (us) and green (our component). It identifies the round for the color who takes action. Besides, for the blue one, our policy is based on choose the one with higher probability to win, which is attached to the edge. On the other hand, the green one's policy is randomly choice.

C. Result

The whole tree is presented at the FIGURE 3. And it really match our decision policy implemented in the code file. Apparently, for the first step, we can see the winning probability will be 0.998 as long as we go to the right child. Correspondently, the game presented in the FIGURE 1 also shows that the blue circle is put on the right-most column, which proves our policy.

III. Convergence process

A. Plot

First, we choose to plot the winning probability as our indication to converge. The project ask us to always win the game, that is, the winning probability should be 1 in the end. Thus, if our policy always wins the game, we meet both requirements, convergence and the final winning probability showed being 1.

Second, in order to show the efficiency of our optimal policy, we also travers other trajectories from the tree root to terminals. That is, we also show other converge processes but some of them may fail. Thus, we distinguish our policy in red line (the process of this trajectory is also shown in FIGURE 1) and make it compared to others.

B. Results

The result in FIGURE 2 show our policy converged. Besides, we also attach the process of playing the game.

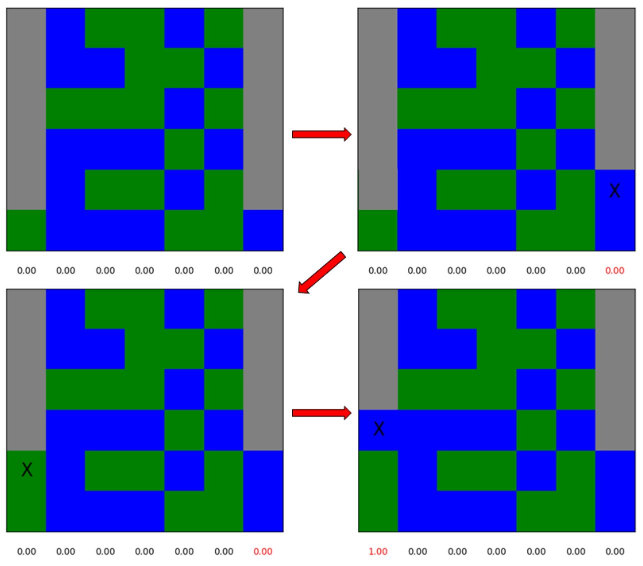

FIGURE 1. Diagram of one game under our policy.

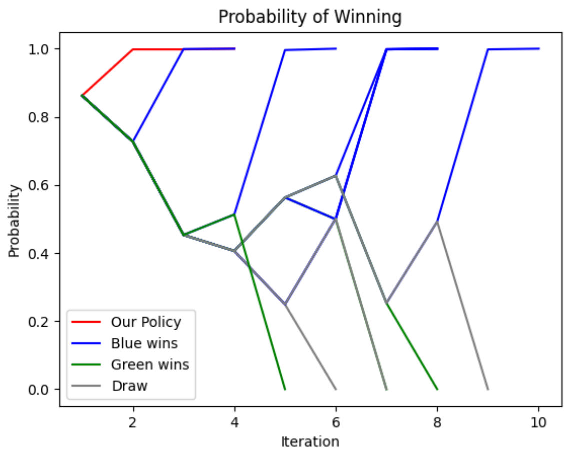

FIGURE 2. Diagram of convergence process

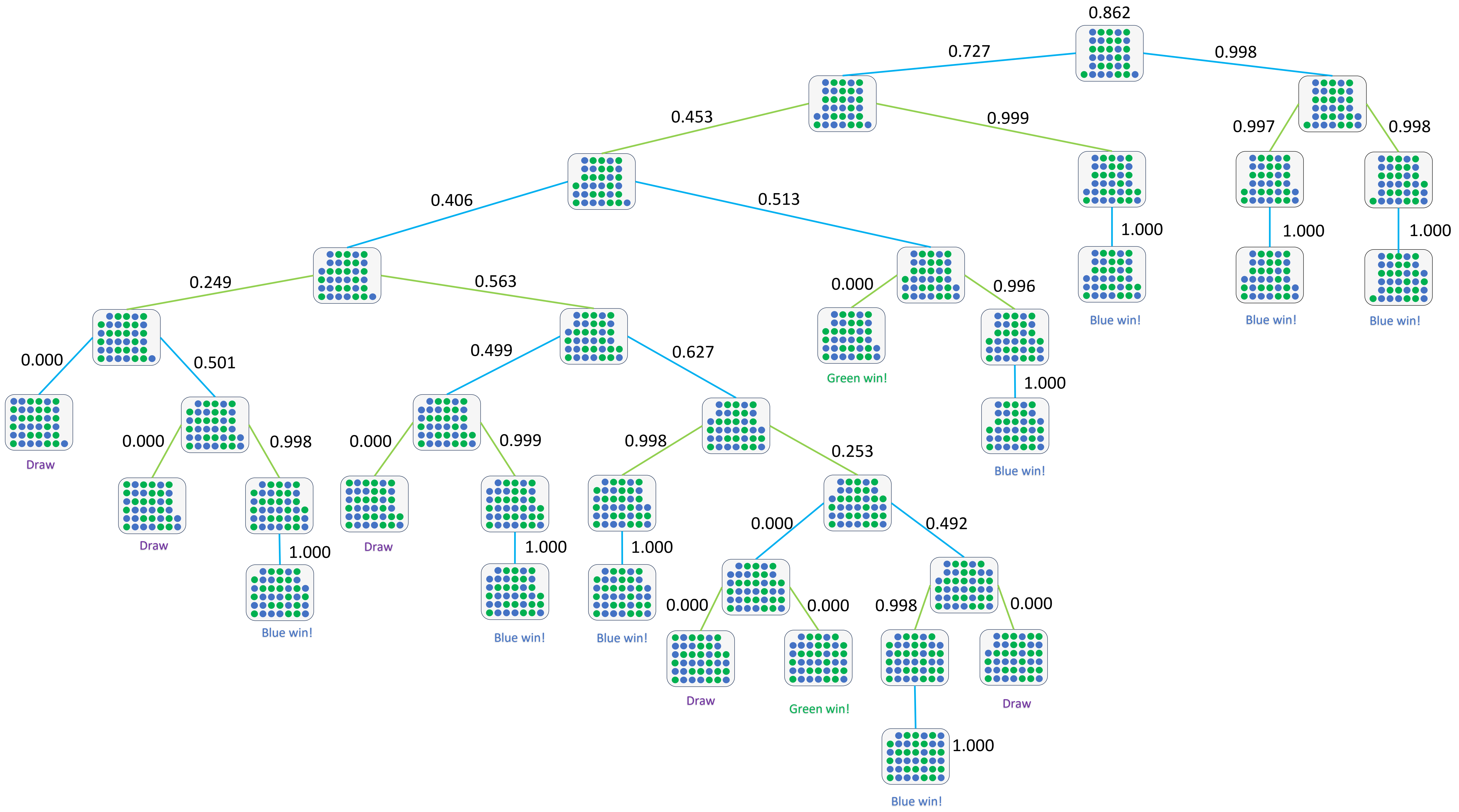

FIGURE 3. Visualization of game