Exercise 8
Monte Carlo Tree Search

In this exercise, we will implement a simple version of Monte Carlo Tree Search for the game Connect4. We use a simple environment implemented in `env.py`. Compared to vanilla environment implementations, we provide three additional functions:

1. `env.action_mask() -> List[bool]` Returns a boolean mask for all legal actions in a given state.
2. `env.get_state() -> Dict[str, Any]` and `env.set_state(state: Dict[str, Any]) -> None` Sets/gets the current environment state.

Programming Tasks:

1. **Preliminaries** First, get familiar with the environment implementation inside `env.py`. Try to run it to see an example execution of the different functionalities. Take note that two different players interleave their actions to interact with the environment. If the action taken by either agent ends in a win, or draw, or has been an invalid move, the environment returns 1, 0, and -1 respectively.
2. **UCT** The win and visitation counts are saved inside `np.ndarray` data structures, where every node carries all the statistics for its children in two array. Inside the class `MCTSNode`, based on the `self.child_value_sum` and `self.child_num_visit` attributes, implement the functions:
   (a) `child_values() -> np.ndarray` Returns the evaluation of first term of the UCT formula (i.e., the value/win ratio for every child).
   (b) `child_exploration() -> np.ndarray` Returns the evaluation of the exploration term of the UCT formula.
   (c) `uct() -> np.ndarray` Combines the two and calculates the UCT.
3. **MCTS Phases** Implement the four phases of MCTS inside the `MCTSNode` class. In detail, implement the functions `selection`, `expansion`, `simulation`, and `backpropagation` inside the `MCTSNode` class.
   Regarding the two-player nature of the game, the only thing you have to take into special account is that the tree alternates between player and opponent nodes. This means, during *backpropagation*, you have to alternate the win indicator (i.e. if the player loses, it's a win for the opponent, and vice versa).
4. **Evaluation** Play around with the implementation and see how it performs. We implemented a random agent for you to test out, if MCTS is implemented correctly (it should always win). Also, you can swap out the opponent with a second MCTS agent and see, how they compete against each other. Also, try out different hyperparameters, e.g., the exploration constant.