对抗搜索算法的实验分析

实验零：补全代码，并将填空内容写在实验报告中

alpha_beta_serach.py

```
for action in action_space:
    next_state = state.next(action)
    new_value = self._search(next_state, alpha, beta)

    if state.active_player() == 0 and new_value > alpha:
        # TODO: 更新alpha 或者beta 的值——1 行
        alpha = new_value
        self._next_state_of[state] = next_state
    elif state.active_player() == 1 and new_value < beta:
        # TODO: 更新alpha 或者beta 的值——1 行
        beta = new_value
        self._next_state_of[state] = next_state

    # TODO: 剪枝
    if alpha >= beta:
        break
```

general_game_search.py

```
for action in action_space:
    next_state = state.next(action)
    cumulative_rewards = self._search(next_state)
    # TODO: 如果某个动作会导致当前玩家的分数提高，则切换到该动作
    if (
        cumulative_rewards[state.active_player()]
        > best_cumulative_rewards[state.active_player()]
    ):
        best_cumulative_rewards = cumulative_rewards
        self._next_state_of[state] = next_state
```

monte_carlo_search.py

```
def _sample_path(self, state: GameStateBase, exploration: float)
-> np.ndarray:
    index = self._state_to_index[state]
    node = self._tree.node_of[index]
    self._visit_count_of[index] += 1

    # 还可以扩展
    if node.n_children < state.n_actions():
        next_state =
state.next(state.action_space()[node.n_children])
```

```
        child = self._tree.create_node()
        self._tree.add_as_child(node, child)
        self._state_to_index[next_state] = child.index
        self._index_to_state[child.index] = next_state
        self._visit_count_of[child.index] = 1
        # TODO：子结点初始累计收益 values 为模拟得到的值——1 行
        values = self._simulate_from(next_state)
        self._value_sums_of[child.index] = values
    elif node.n_children > 0:
        selection = MaxSelection()
        selection.initialize(node.n_children, -float("inf"))
        for i in range(node.n_children):
            child = node.child(i).index
            # TODO：选择 UCT 值最大的子结点继续探索
            selection.submit(
                self._value_sums_of[child][state.active_player()]
                / self._visit_count_of[child]
                + exploration
                * sqrt(
                    log(self._visit_count_of[index]) /
self._visit_count_of[child]
                )
            )
        next_state =
state.next(state.action_space()[selection.selected_index()])
        values = self._sample_path(next_state, exploration)
    else:
        values = np.array(state.cumulative_rewards(),
dtype=np.float64)

    self._value_sums_of[index] += values
    return values

def select_action(self, iterations: int, exploration: float):
    root_state = self._index_to_state[0]
    for i in range(iterations):
        self._sample_path(root_state, exploration)

    root = self._tree.root
    selection = MaxSelection()
    selection.initialize(root.n_children, -float("inf"))

    for i in range(root.n_children):
        child = root.child(i).index
        # TODO：按平均价值贪心选择
        selection.submit(

self._value_sums_of[child][root_state.active_player()]
            / self._visit_count_of[child]
        )

    return root_state.action_space()[selection.selected_index()]
```