**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.hpp

// 更新alpha或者beta的值：

 // 如果当前是玩家0决策，则其最大化自身的分数（提升自己分数下界）

            if (state.active\_player() == 0 and new\_value > alpha){

                // TODO：更新alpha或者beta的值——1行

                alpha = new\_value;

                next\_state\_of[state] = next\_state;

            // 如果当前是玩家1决策，则其最小化玩家0的分数（降低对手分数上界）

            } else if (state.active\_player() == 1 and new\_value < beta){

                // TODO：更新alpha或者beta的值——1行

                beta = new\_value;

                next\_state\_of[state] = next\_state;

            }

            // TODO：剪枝

            if (alpha >= beta)break;

general\_game\_search.hpp

// /\*\*\*\*\*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()])

monte\_carlo\_tree\_search.hpp

// TODO：子结点初始累计收益 values 为模拟得到的值——1行

            values = simulate\_from(next\_state);

/\*\*\*\*\*\*TODO：选择UCT值最大的子结点继续探索\*\*\*\*\*/

selection.submit(value\_sums\_of[child][state.active\_player()]/visit\_count\_of[child]+exploration\*sqrt(2\*log(visit\_count\_of[index])/visit\_count\_of[child]));

/\*\*\*\*\*\*\* TODO： 按平均价值贪心选择（需要参考root\_state的接口）\*\*\*\*\*\*/

selection.submit(value\_sums\_of[child][root\_state.active\_player()]/visit\_count\_of[child]);