**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()]