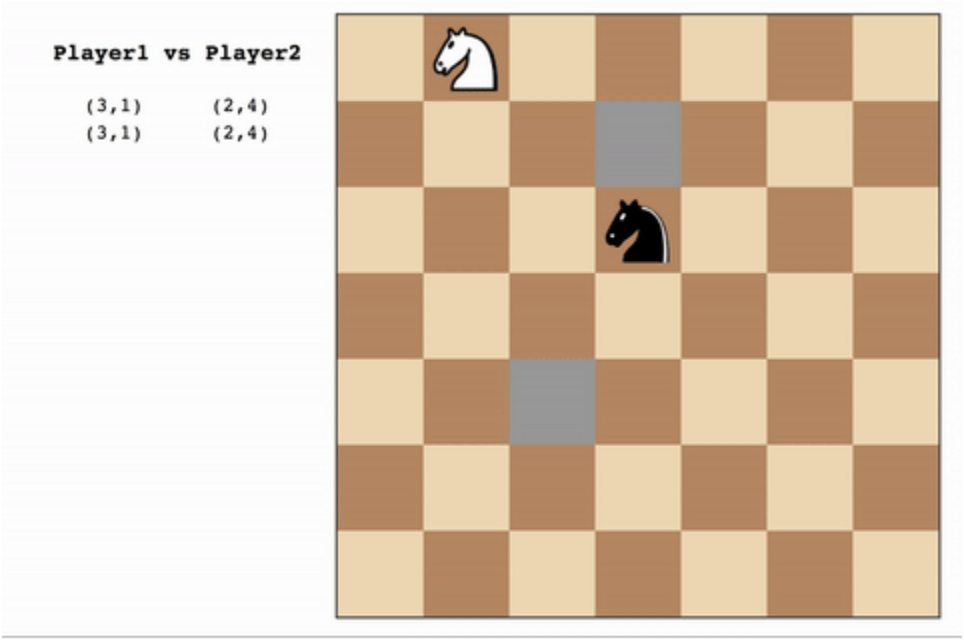
Adversarial Game Playing Agent Report

Ebrahim Jakoet Oct 30, 2019

Introduction

In this project, we use adversarial search techniques to build an agent to play knights Isolation. In this Isolation Game, tokens move in L-shapes like Knights on chess game.



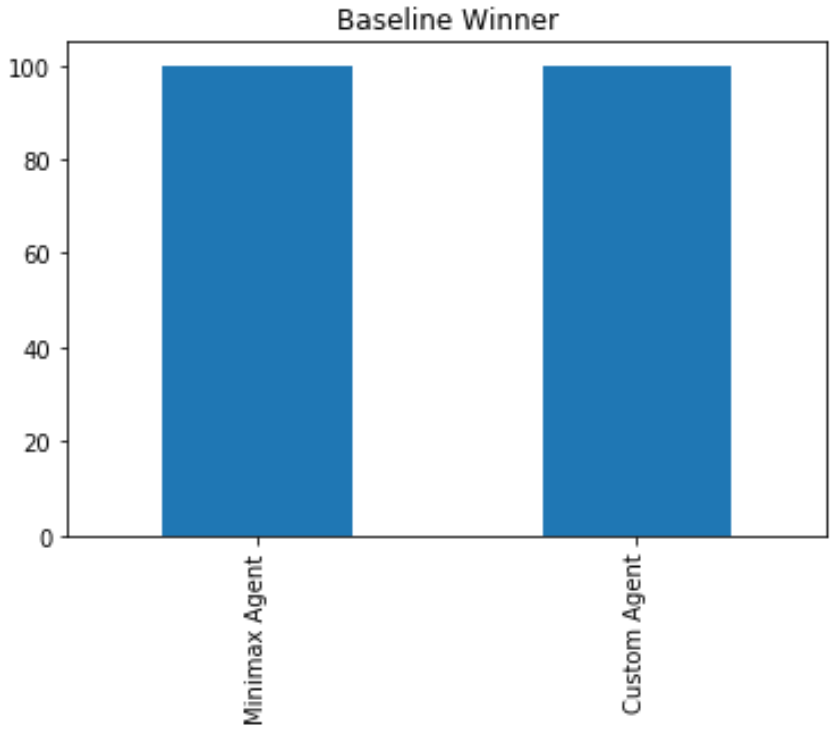
NOTE: The project is played on a 9x11 grid instead of a 7x7 grid.

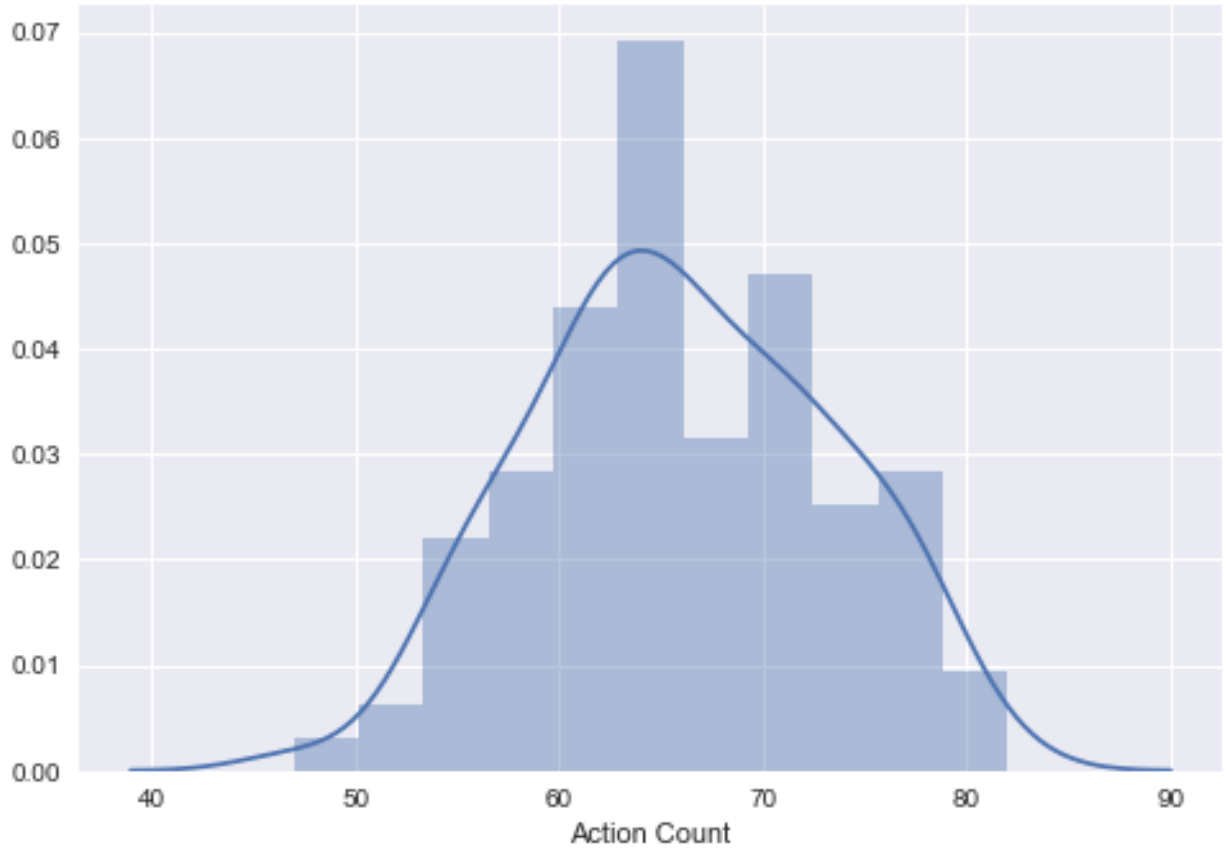
I have selected Option 3, using the Advanced Search Techniques. The Baseline technique uses the MiniMax search algorithm and the advanced search algorithm it is compared to, is the Monte Carlo Tree Search algorithm.

Baseline Results

In the Baseline technique we used the MiniMax algorithm against a MiniMax opponent for a total of 100 games. We used the fair game option for a fair comparison between the agents. The Baseline MiniMax Custom player had a 50% success rate against the opponent MiniMax algorithm. This result is expected because both algorithms were the same in this case.

The charts below show the win counts and number of actions distribution for Baseline algorithm.


Baseline Winner
100
80
60
40
20
0
Minimax Agent
Custom Agent


0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
40
50
60
70
80
90
Action Count

The Monte Carlo Tree Search Algorithm

The Monte Carlo Tree Search (MCTS) was implemented based on the pseudocode from the notes. The opponent player used the MiniMax Algorithm as before.

Algorithm 2 The UCT algorithm.

function UCTSEARCH(s_0)
  create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow$ TREEPOLICY(v_0)
    $\Delta \leftarrow$ DEFAULTPOLICY($s(v_l)$)
    BACKUP(v_l, Δ)
  return a(BESTCHILD($v_0, 0$))

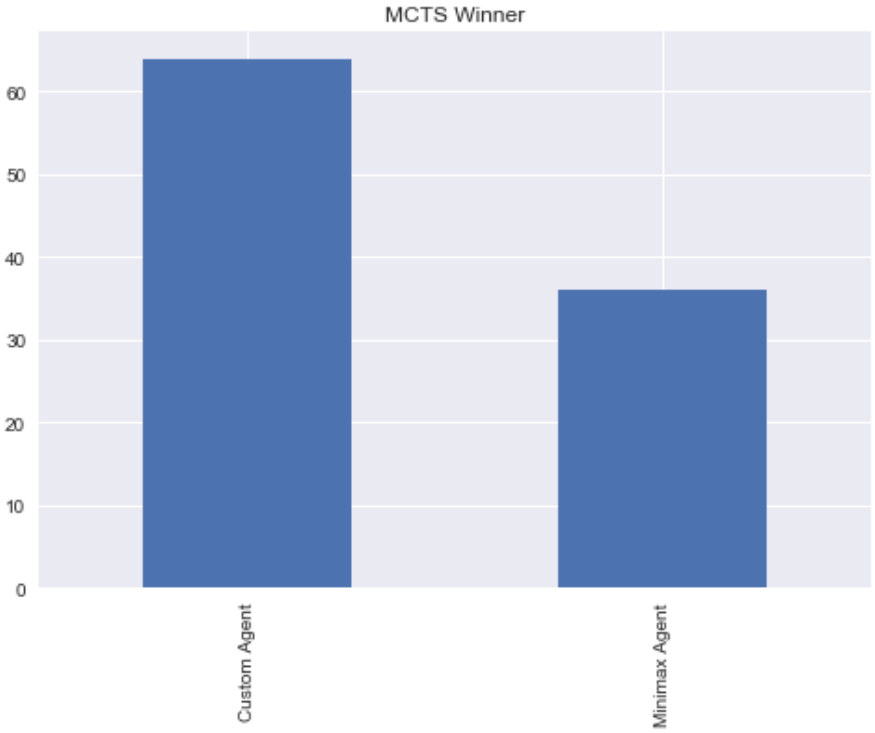
function TREEPOLICY(v)
  while v is nonterminal **do**
    if v not fully expanded **then**
      return EXPAND(v)
    else
      $v \leftarrow$ BESTCHILD(v, Cp)
  return v

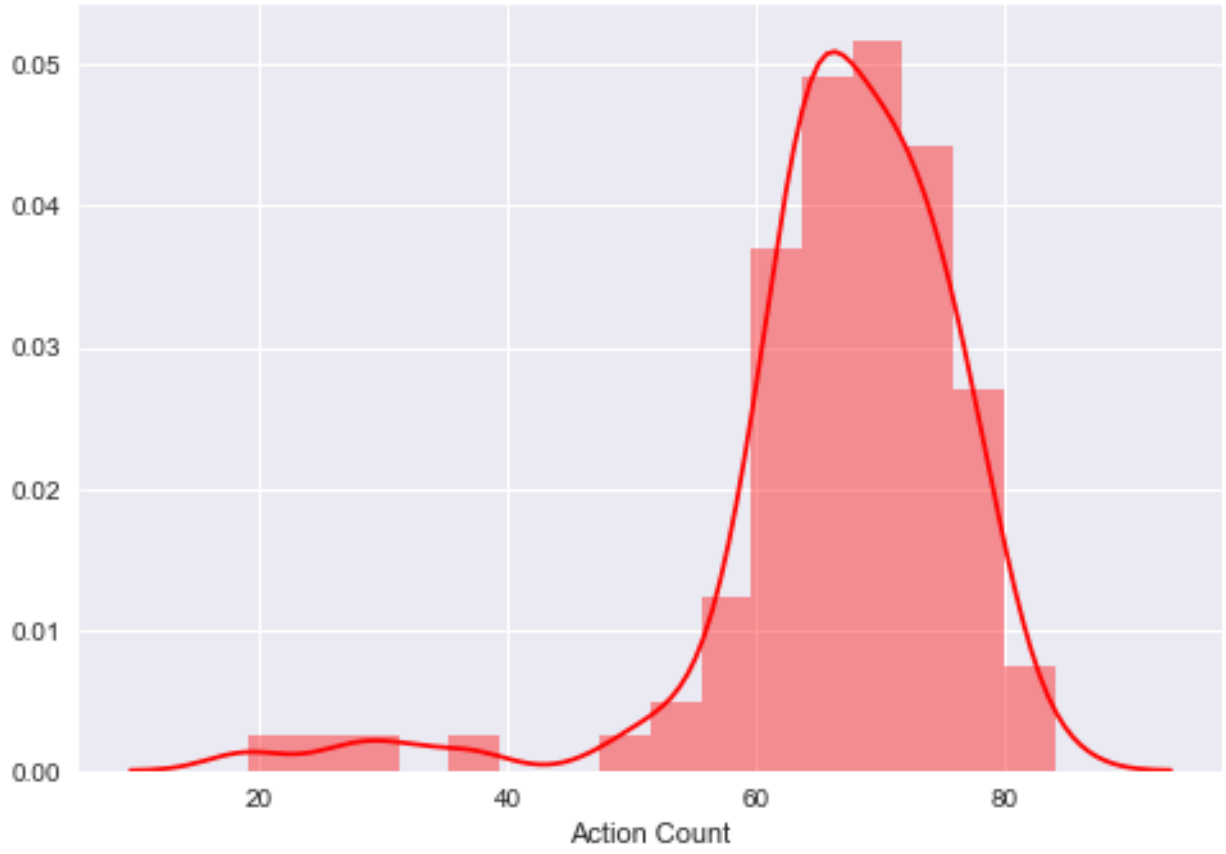
function EXPAND(v)
  choose $a \in$ untried actions from $A(s(v))$
  add a new child v' to v
    with $s(v') = f(s(v), a)$
    and $a(v') = a$
  return v'

function BESTCHILD(v, c)
  return $\underset{v' \in \text{children of } v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$

function DEFAULTPOLICY(s)
  while s is non-terminal **do**
    choose $a \in A(s)$ uniformly at random
    $s \leftarrow f(s, a)$
  return reward for state s

The charts below show the win counts and number of actions distribution for MCTS algorithm. The MCTS algorithm had a 64% win ratio against the MiniMax algorithm. This rate can be improved if more iterations were allowed, but it comes at the cost of increased processing time.


MCTS Winner
60
50
40
30
20
10
0
Custom Agent
Minimax Agent


0.05
0.04
0.03
0.02
0.01
0.00
20
40
60
80
Action Count

Discussion on MCTS vs Baseline

The MCTS algorithm had a 14% improvement over the MiniMax algorithm and this can be improved further by allowing more iterations. What's interesting about looking at the actions taken, the Baseline MiniMax algorithm seems to be more normally distributed, but the MCTS algorithm was able to win the game more often with fewer actions. This shows that it has a more efficient exploration of the tree compared to the MiniMax algorithm. The strength of the MCTS algorithm is that it simulates the remaining game based on a default policy. Using the Upper Confidence Bound for Trees (UCT)as a sampling strategy we can avoid doing an exhaustive search of the search tree. This allows us to sample more promising actions more often than other actions, thus producing a more efficient search result.