COMP 560 Artificial Intelligence: Constraint Satisfaction Problems & Games (Assignment 2)

Ben Hu,* Brian Luong,† and Stephen Yan‡
(Dated: October 14, 2015)

We discuss several different algorithm to solve Constraint Satisfaction Problems (CSP) and Games. First, we discuss the Graduation Dinner problem, a basic CSP. Secondly, we explore a more complex CSP called Hide and Seek. We then move on to the problem of Games where we discuss several adversarial search algorithms for the Candy Game. Throughout the paper, we discuss our approaches and algorithm implementations in Java to solve the problems and their subproblems

I. CONSTRAINT SATISFACTION PROBLEMS

A. Graduation Dinner

The problem is restated here: "You are helping to figure out the seating at your graduation dinner table. Your mom has already worked out seating on your side of the table, but wants you to decide where to put the relatives on the other side of the table. The relatives are your cousin Jasmine, your cousin Jason, your aunt Trudy, her husband Randy, and your aunt Misty. There are 5 seats, numbered 1-5 with 1 being the seat closest to the kitchen. You have the following constraints:"

a. No two relatives can sit in the same seat.
b. Your cousins can't be seated next to each other because they will start a food fight.
c. Jason as the eldest cousin gets the privilege of being seated closer to the kitchen than his sister Jasmine so he can get to the food first.
d. Your aunt Misty shouldn't be seated next to Jason, Trudy or Randy because she invariably instigates an argument about politics and disrupts dinner.
e. Additionally, Misty and Trudy have to be seated with at least 2 seats between them as a buffer.

We can formulate this as a binary CSP where the variables are the relatives: `Jasmine, Jason, Trudy, Randy,` and `Misty`. The domains for each of these are integers {`1,2,3,4,5`}, which represent seat locations, with `1` being closest to the kitchen and `5` being the farthest. The binary constraints inferenced from the scenario above are as follows: `P(x)` denotes the position of relative `x`.

1. *`Alldiff`*`(Jasmine, Jason, Trudy, Randy, Misty)`
2. *`notAdjacent`*`(Jasmine, Jason)`
3. `Jason < Jasmine`
4. *`notAdjacent`*`(Misty, Jason),`
   `notAdjacent``(Misty, Trudy),`
   `notAdjacent``(Misty, Randy)`
5. `|Misty - Trudy| >= 2`

We run arc consistency to reduce the size of our domain for each variable. The domains for each variable are now:

1. `Jasmine: [4]`
2. `Jason: [1, 2, 3]`
3. `Trudy: [1, 2]`
4. `Randy: [1, 2, 3]`
5. `Misty: [5]`

Using Minimum Remaining Value (MRV) to assign variables to the domains, we will either assign a variable to Jasmine or Misty, because they have the smallest domains.

Let's say we assume Randy is seated in the middle and run arc consistency again. The domains for each variable are now:

1. `Jasmine: [4]`
2. `Jason: [1, 2]`
3. `Trudy: [1, 2]`
4. `Randy: [3]`
5. `Misty: [5]`

After this, the solutions to this CSP are apparent.
`[Jason, Trudy, Randy, Jasmine, Misty]`
or `[Trudy, Jason, Randy, Jasmine, Misty]`

B. Hide and Seek

This problem is formulated where there are `N` friends all playing a game of hide and seek on a `N x N` grid. Each column contains one friend. There are also some trees within this grid. Friends cannot stand on trees or see past them. Friends have the ability to see in 8 directions: up, down, right, left, and the diagonals. We want to position all the friends in the grid such that no friend can see another friend. This problem is similar to the N-Queens problem, except there are "barrier" that limit the range of sight.

Formally, we can define the CSP such that the variables are the grid locations with the domain being the state at

* bxhu@live.unc.edu
† bluong@email.unc.edu
‡ sjyan@cs.unc.edu

that grid location (friend, tree, empty). Any friend on the grid must not be able to view another friend in any of the eight directions.

To solve this problem, we decided to use various local search algorithms.[1]

In these algorithms, the heuristic we use is the sum of all friends each friend can see. The optimal solution is then the value 0, when every friend is unable to see any friend.

```
private int calculateHeuristic(Grid grid) {
   int heuristic = 0;
   for (int columnIndex = 0; columnIndex <
       grid.getNumFriends(); columnIndex++) {
      int friendIndex =
          grid.getColumnToFriendMap().
      get(columnIndex);
      heuristic += findConflicts(friendIndex,
          columnIndex, grid);
   }
   return heuristic;
}

private int findConflicts(int x, int y, Grid
    grid) {
   int conflicts = 0;
   conflicts += findConflictUpRight(x, y, grid);
   conflicts += findConflictUpLeft(x, y, grid);
   conflicts += findConflictDownLeft(x, y, grid);
   conflicts += findConflictDownRight(x, y,
       grid);
   conflicts += findConflictsRight(x, y, grid);
   conflicts += findConflictsLeft(x, y, grid);
   return conflicts;
}
```

Steepest Ascent Hill Climbing

This version of local search is essentially what it sounds like. At each state we generate all possible successor states and our next move with the best state. It is essentially a type of greedy algorithm.

For our first version of this algorithm, we continuously loop through every column and update the grid to an improved state until there are no further improvements. In the hill analogy, we will only choose the steepest option if it exists.

```
if (heuristic < bestHeuristic) {
  nextBestIndex = i;
  bestHeuristic = heuristic;
}
```

In our second version, we modify the algorithm so that we allow for successor states with equal value to that of the current state. This is different from the first version in that the first version will terminate when there are no more better options available. This modification will allow changing of states even when there may not be a better one. Using the hill analogy, this version will allow movement on plateaus. This algorithm will come at a higher cost of moving sideways, but with the hope of finding a "breakthrough" into a better state. However, it is easy to see that an infinite loop will occur whenever an infinite plateau is reached. To solve this problem, a limit on the number of consecutive sideways movements is implemented.

```
if (heuristic <= bestHeuristic) {
   nextBestIndex = i;
   bestHeuristic = heuristic;
}
```

The third version uses a more randomized approach. Instead of incrementing the column and checking successor states, this algorithm randomly chooses a column. This variation may or may not result in faster or better results. We believe that this version is more inconsistent; it may find some solutions much faster, but sometimes much slower.

```
int randomPivotColumn = (int) (Math.random() *
    grid.getNumFriends());
```

Stochastic Hill Climbing

In this version of local search, instead of greedily choosing the best successor state (steepest), the next state is chosen at random out of the possible improved successor states (uphill moves).

```
if (numImprovements > 0) {
   int randomIndex = (int) (Math.random() *
       numImprovements;
   grid[randomIndex][column].setState(FRIEND);
   grid[randomIndex][column].setState(EMPTY);
   grid.updateColumnToFriendMap(column,
       possibleImprovements.get(randomIndex));
   return 0;
}
```

All of the algorithms mentioned above are incomplete - there is a possibility to get stuck in a local minima and never reach the optimal solution. The only way to resolve this issue is to allow for movement to less optimal successor states (downhill). This way, local minima (maxima) can be avoided.

We ran 100 trials for each algorithm for a variety of input grids which vary by size and number of trees.

We can observe that the optimal solution was not always found in most of these trials. This was not surprising given the greedy nature of hill climbing searches.

From these trials, we can conclude that the second version of Steepest Ascent Hill Climbing Search is the best algorithm. It has the highest rate of finding the optimal solution for every grid variation. However, this algorithm

[1] The starting state of each iteration of the following algorithms includes the trees set in place and each friend places in a randomly initialized location.

on average has a higher cost for finding solutions. This makes sense because there could be a large number of sideways movements. Another notable result was that the number of steps it took for version three before finding a non-optimal solution was significantly higher than any other cost of any algorithm.

Version one of the Steepest Ascent Hill Climbing Search yields the least cost for all solutions found, although performing second best (next to version two). In regards to finding the optimal solution, Stochastic and version one and three of Steepest Ascent performed roughly equally over all the trials.

We also found that as the number of trees decreased for a certain grid size, the probability of finding an optimal solution decreased and cost increased. This observation was not unexpected - the decrease in trees increased the number possible states, and thus allowed a greater variability in choosing successive states.

There are also possible configurations where no optimal solution can ever be found; the algorithm always gets stuck in a local minima. This depends on the tree and initial friend placement. One such configuration is the 4x4 10 tree solution in the appendix.

II. GAMES

The game of interest is a board game where the spaces are assigned static values for the entirety of a game (but due to change for different games). There are two opponents, green candy and blue candy, whose goal is to obtain the maximum score by owning spaces on the board. Normally this would be a fairly simple game to evaluate, but we introduce the notion of *candy capture*. We allow candies that are placed adjacent to an opponent candy as well as its own color to overtake the opposing color, thus earning itself additional values on the board (much like in the game "Reversi").

We implemented MiniMax search and a revision on MiniMax, Alpha-Beta search to play the game. The main task apart from implementing the search was to design the evaluation function to fit the intelligence for the game.

The evaluation function will return a numerical "score" based on different evaluation criteria that will be used to determine the next best move. In this game, we will use the obvious criteria which will be the disparity between opponent and player candy values[2]. We will also use candy "stability" as a criterion for the evaluation function. Stability refers to the likelihood a candy will be captured in the future. A stable candy is a candy that is impossible to capture from that game state forward. An unstable candy is a candy that can be captured in an immediate move. A semi-stable candy is one that cannot be captured in the next move but can still be captured in some future state.

[2] Legal moves for a player are all spaces that are empty

```
private int evaluate() {
    int score = 0, myCandies = 0, oppCandies = 0;
    for(int row = 0; row < ROWS; row++) {
      for(int col = 0; col < COLS; col++) {
         if(cells[row][col].getColor() == myColor) {
             myCandies += cells[row][col].getValue();
         } else if(cells[row][col].getColor() ==
              oppColor) {
             oppCandies +=
                 cells[row][col].getValue();
         }
        }
    }
    score += (myCandies - oppCandies) / (myCandies
        + oppCandies);

    return score;
}
```

Implicitly, the above first draft rewards positively if the player's candies' sum total outnumber the opponent's candies' total value, and punishes if the opposite is true. This is weighed the most because it is the primary determinant in winning the game.

```
private int evaluate() {
    int score = 0, myStability = 0; oppStability =
        0;
    for(int row = 0; row < ROWS; row++) {
      for(int col = 0; col < COLS; col++) {
         if(cells[row][col].getColor() == myColor) {
            Cell[] neighbors =
                getNeighbors(cells[row][col]);
            boolean flankable = false;
            checkNeighbors:
            for(int i = 0; i < 4; i++) {
                if(neighbors[i].getColor() ==
                    Color.EMPTY) {
                  Cell[] neighborsNeighbors =
                      getNeighbors(neighbors[i]);
                 for(int j = 0; j < 3; j++) {
                    if(i + j != 3) {
                       if(neighborsNeighbors[j].
                       getColor()
                             == oppColor) {
                          break checkNeighbors;
                       }
                    }
                 }
              }
            }

            myStability += (flankable) ? -1 *
                cells[row][col].getValue()
                 : cells[row][col].getValue();
         } else if(cells[row][col].getColor() ==
              oppColor) {
            Cell[] neighbors =
                getNeighbors(cells[row][col]);
            boolean flankable = false;
```

```
            checkNeighbors:
            for(int i = 0; i < 4; i++) {
                if(neighbors[i].getColor() ==
                    Color.EMPTY) {
                  Cell[] neighborsNeighbors =
                      getNeighbors(neighbors[i]);
                  for(int j = 0; j < 3; j++) {
                     if(i + j != 3) {
                        if(neighborsNeighbors[j].
                        getColor()
                              == oppColor) {
                           break checkNeighbors;
                        }
                     }
                  }
               }
            }
            myStability += (flankable) ?
                cells[row][col].getValue()
                 : -1 * cells[row][col].getValue();
        }

    score += (myStability - oppStability) /
        (myStability + oppStability);

    return score;
}
```

We also check for total piece stability. We check each candy for a player to see if it is flankable in the next move. We check this by checking the neighbors of the piece's neighbors (less the neighbor that is the actual piece). We first check if a neighbor is empty (because the opponent would choose to place a candy there if it has potential to be flanked) and then check that neighbor's neighbors for any opponent candies. If there exists any opponent candies in those three spaces, then we say that the candy is unstable. The candy is stable if its neighbors are completely taken (because it could not be flanked for the remainder of the game). The candy is semi-stable if it is not immediately flankable but could be flanked at some point in the remainder of the game.

The result was that the stability criterion didn't have that much of an impact on the algorithm's decisions. We suspect that this is because our specifications for the neighbor's neighbors statistic wasn't refined enough. It is a greater influence on games like Reversi where multiple pieces can be flanked in a single move. However that is not the case for our game where only at most a single candy can be flanked at any given move.

We would have liked to include a criterion along the lines of mobility, however that too would have little impact on the resulting intelligence since mobility is defined as playable moves, which would simply be the number of empty spaces on the board.

We also found there to be a huge top left bias because we looped through iteratively starting at the top left. Since we only chose another move if it gave more utility and our only utility function was the points rewarded for a given move, unbalanced boards would not necessarily favor more depth on a search. For example, the game board Bit-O-Honey is weighted towards the bottom, which slowed the alpha beta search considerably because there was nothing to prune until the end. This also meant that boards that were more favorable towards not playing near the top-left would cause that player to lose.

The searches are also favorable at an odd depth. This is because of the horizon effect of the flipping strategy. With an even depth, the player of interest would not "see" the opponent being able to flip its piece. In effect, it would let itself play a bad move because it thinks it will take a piece, even though that would leave it vulnerable to being taken back.

III. FUTURE IMPROVEMENTS

Since Steepest Ascent and Stochastic Hill Climbing are both unable to solve the issue of local peaks, we could try implementing an algorithm that supports downhill moves such as the Simulating Annealing algorithm.

Since our implementation of MiniMax has a large top-left bias for games, we could implement random iterator that could sometimes start in the center of the grid or in other corners in the future. We could also use a quiescence search or a singular extension method to eliminate the horizon effect that is especially in our implementation of Alpha-Beta pruning.

We also found different patterns to be heavily favorable for the candy game. In the future, we could include a table-lookup method for a variety of winning patterns to make the searches more efficient.

IV. INDIVIDUAL CONTRIBUTIONS

f. Hu, B. Analysis of Graduation Dinner, Minimax implementation

g. Luong, B. Contributing to Graduation Dinner, Implementing Hide & Seek algorithms and analysis

h. Yan, S. Minimax search implementation and evaluation function, analysis thereof, Report styling

Appendix

```
CURRENT FOREST -------> DEFAULT FOREST

FTF   F
T TFF
TF    T
   T   T

  TTTF
T    TTF
 TT
------------------------------------------------------------
An optimal solution from using the SAHC algorithm

FT F
T T  F
T     TF
  FT   T
      F
  TTT
TF   TT
 TT F

Solution Indices

1 1
1 4
2 6
3 8
4 3
5 7
7 2
8 5

Out of 100 runs, SAHC found the optimal solution 85 times.
Average Steps to an optimal solution 6
Average Steps for non-optimal solution 12
------------------------------------------------------------
An optimal solution from using the SAHCV2 algorithm

 T
T T    F
T   F T
  FT  FT

 FTTT
T  F TT
FTT  F

Solution Indices

2 8
3 5
4 3
4 7
6 2
7 4
8 1
8 6

Out of 100 runs, SAHCV2 found the optimal solution 98 times.
Average Steps to an optimal solution 9
Average Steps for non-optimal solution 21
------------------------------------------------------------
An optimal solution from using the SAHCV3 algorithm

FT    F
T T F
TF    T
   T F T
  F
  TTT
T    TTF
 TTF

Solution Indices

1 1
1 7
2 5
3 2
4 6
5 3
7 8
8 4

Out of 100 runs, SAHCV3 found the optimal solution 82 times.
Average Steps to an optimal solution 11
Average Steps for non-optimal solution 107
------------------------------------------------------------
An optimal solution from using the SHC algorithm

 T  F
TFT    F
T     T
  FT   T
     F
  TTT
T  F TT
FTT   F

Solution Indices

1 5
2 2
2 8
4 3
5 6
7 4
8 1
8 7

Out of 100 runs, SHC found the optimal solution 77 times.
Average Steps to an optimal solution 7
Average Steps for non-optimal solution 13
------------------------------------------------------------
CURRENT FOREST -------> 8x8 WITH 10 TREES

 T
TFT
F   F T
  FT  FT

   F
T    TTF
 T   F
------------------------------------------------------------
An optimal solution from using the SAHC algorithm

FT  F
T T
 F    TF
   T F T
  F
      F
T    TT
 T F

Solution Indices

1 1
1 5
3 2
3 8
4 6
5 3
6 7
8 4

Out of 100 runs, SAHC found the optimal solution 67 times.
Average Steps to an optimal solution 8
Average Steps for non-optimal solution 14
------------------------------------------------------------
An optimal solution from using the SAHCV2 algorithm

FT    F
T TF
      TF
   TF  T
 F

T F  TT
 T   F

Solution Indices

1 1
```

```
1 7
2 4
3 8
4 5
5 2
7 3
8 6

Out of 100 runs, SAHCV2 found the optimal solution 91 times.
Average Steps to an optimal solution 14
Average Steps for non-optimal solution 18
------------------------------------------------------------
An optimal solution from using the SAHCV3 algorithm

FTF
T T  F
      TF
 F T   T
    F
      F
T    TT
 T F

Solution Indices

1 1
1 3
2 6
3 8
4 2
5 5
6 7
8 4

Out of 100 runs, SAHCV3 found the optimal solution 58 times.
Average Steps to an optimal solution 16
Average Steps for non-optimal solution 109
------------------------------------------------------------
An optimal solution from using the SHC algorithm

 T F
TFT   F
      T
   T F T
  F
F
T    TTF
 T  F

Solution Indices

1 4
2 2
2 7
4 6
5 3
6 1
7 8
8 5

Out of 100 runs, SHC found the optimal solution 59 times.
Average Steps to an optimal solution 9
Average Steps for non-optimal solution 14
------------------------------------------------------------
CURRENT FOREST -------> 8x8 WITH 5 TREES

FF   F
  F
   FF

      F
  T T
     TT
  T    F
------------------------------------------------------------
An optimal solution from using the SAHC algorithm

     F

    F
 F
   F
  T T F
  F  TT
F T    F

Solution Indices

1 6
3 5
4 2
5 4
6 7
7 3
8 1
8 8

Out of 100 runs, SAHC found the optimal solution 46 times.
Average Steps to an optimal solution 8
Average Steps for non-optimal solution 13
------------------------------------------------------------
An optimal solution from using the SAHCV2 algorithm

   F
      F
  F
     F
 F
  T T
    FTTF
F T

Solution Indices

1 4
2 7
3 3
4 6
5 2
7 5
7 8
8 1

Out of 100 runs, SAHCV2 found the optimal solution 87 times.
Average Steps to an optimal solution 13
Average Steps for non-optimal solution 21
------------------------------------------------------------
An optimal solution from using the SAHCV3 algorithm

    F
      F
 F

  F
F T TF
   F TTF
  T

Solution Indices

1 5
2 7
3 2
5 3
6 1
6 6
7 4
7 8

Out of 100 runs, SAHCV3 found the optimal solution 35 times.
Average Steps to an optimal solution 18
Average Steps for non-optimal solution 110
------------------------------------------------------------
An optimal solution from using the SHC algorithm

  F
      F
     F
   F
 F
  T T F
     TT
F T F

Solution Indices
```

```
1 3
2 8
3 6
4 4
5 2
6 7
8 1
8 5

Out of 100 runs, SHC found the optimal solution 49 times.
Average Steps to an optimal solution 9
Average Steps for non-optimal solution 15
------------------------------------------------------------
CURRENT FOREST -------> 8x8 WITH 2 TREES

   F


  F


     F F
    TF
 F    T
F     F
------------------------------------------------------------
An optimal solution from using the SAHC algorithm

  F
     F
       F
F
    F
   T  F
 F    T
   F

Solution Indices

1 3
2 6
3 8
4 1
5 5
6 7
7 2
8 4

Out of 100 runs, SAHC found the optimal solution 30 times.
Average Steps to an optimal solution 8
Average Steps for non-optimal solution 14
------------------------------------------------------------
An optimal solution from using the SAHCV2 algorithm

     F


 F
   F
F
  FT  F
    F T
       F

Solution Indices

1 6
3 2
4 4
5 1
6 3
6 7
7 5
8 8

Out of 100 runs, SAHCV2 found the optimal solution 83 times.
Average Steps to an optimal solution 23
Average Steps for non-optimal solution 30
------------------------------------------------------------
An optimal solution from using the SAHCV3 algorithm

      F
    F
  F
F
     F
   T   F
 F    T
   F

Solution Indices

1 7
2 5
3 3
4 1
5 6
6 8
7 2
8 4

Out of 100 runs, SAHCV3 found the optimal solution 35 times.
Average Steps to an optimal solution 16
Average Steps for non-optimal solution 111
------------------------------------------------------------
An optimal solution from using the SHC algorithm

    F
 F
   F
      F
  F
   T   F
     FT
F

Solution Indices

1 5
2 2
3 4
4 7
5 3
6 8
7 6
8 1

Out of 100 runs, SHC found the optimal solution 29 times.
Average Steps to an optimal solution 13
Average Steps for non-optimal solution 15
------------------------------------------------------------
CURRENT FOREST -------> 6x6 WITH 15 TREES

FT  TF
 TTTT
 FT T
  TT
T  FFT
T FT
------------------------------------------------------------
An optimal solution from using the SAHC algorithm

FT FT
 TTTT
 FT TF
  TT
T F  T
T  TF

Solution Indices

1 1
1 4
3 2
3 6
5 3
6 5

Out of 100 runs, SAHC found the optimal solution 88 times.
Average Steps to an optimal solution 3
Average Steps for non-optimal solution 8
------------------------------------------------------------
An optimal solution from using the SAHCV2 algorithm

 TF T
FTTTT
  TFT
 FTT F
T    T
```

```
T  TF

Solution Indices

1 3
2 1
3 4
4 2
4 6
6 5

Out of 100 runs, SAHCV2 found the optimal solution 96 times.
Average Steps to an optimal solution 5
Average Steps for non-optimal solution 11
------------------------------------------------------------
An optimal solution from using the SAHCV3 algorithm

 T FTF
FTTTT
  T T
 FTT
T   FT
T FT

Solution Indices

1 4
1 6
2 1
4 2
5 5
6 3

Out of 100 runs, SAHCV3 found the optimal solution 80 times.
Average Steps to an optimal solution 5
Average Steps for non-optimal solution 102
------------------------------------------------------------
An optimal solution from using the SHC algorithm

FT FT
 TTTT
 FT T
  TT F
T    T
T FTF

Solution Indices

1 1
1 4
3 2
4 6
6 3
6 5

Out of 100 runs, SHC found the optimal solution 82 times.
Average Steps to an optimal solution 3
Average Steps for non-optimal solution 7
------------------------------------------------------------
CURRENT FOREST -------> 6x6 WITH 10 TREES

T   FT
 T   T
F
 TF TT
 F F
    F
------------------------------------------------------------
An optimal solution from using the SAHC algorithm

TF   T
 T   T
    F
FTF TT
     F
   F

Solution Indices

1 2
3 5
4 1
4 3
5 6
6 4

Out of 100 runs, SAHC found the optimal solution 27 times.
Average Steps to an optimal solution 4
Average Steps for non-optimal solution 9
------------------------------------------------------------
An optimal solution from using the SAHCV2 algorithm

TF   T
 T F T
     F
FTF TT
    F


Solution Indices

1 2
2 4
3 6
4 1
4 3
5 5

Out of 100 runs, SAHCV2 found the optimal solution 81 times.
Average Steps to an optimal solution 16
Average Steps for non-optimal solution 29
------------------------------------------------------------
An optimal solution from using the SAHCV3 algorithm

T    T
 T F T
     F
FTF TT
    F
 F

Solution Indices

2 4
3 6
4 1
4 3
5 5
6 2

Out of 100 runs, SAHCV3 found the optimal solution 21 times.
Average Steps to an optimal solution 7
Average Steps for non-optimal solution 105
------------------------------------------------------------
An optimal solution from using the SHC algorithm

T   FT
FTF  T
     F
 T FTT
 F

Solution Indices

1 5
2 1
2 3
3 6
4 4
5 2

Out of 100 runs, SHC found the optimal solution 13 times.
Average Steps to an optimal solution 7
Average Steps for non-optimal solution 9
------------------------------------------------------------
CURRENT FOREST -------> 6x6 WITH 5 TREES

FF
T
  F  F
   TTT
    FT
   F
------------------------------------------------------------
An optimal solution from using the SAHC algorithm
```

```
 F
T  F
     F
F  TTT
  F  T
    F

Solution Indices

1 2
2 4
3 6
4 1
5 3
6 5

Out of 100 runs, SAHC found the optimal solution 21 times.
Average Steps to an optimal solution 6
Average Steps for non-optimal solution 9
------------------------------------------------------------
An optimal solution from using the SAHCV2 algorithm

 F
T   F
  F
F  TTT
   F T
     F

Solution Indices

1 2
2 5
3 3
4 1
5 4
6 6

Out of 100 runs, SAHCV2 found the optimal solution 78 times.
Average Steps to an optimal solution 19
Average Steps for non-optimal solution 31
------------------------------------------------------------
An optimal solution from using the SAHCV3 algorithm

   F
T    F
F
  FTTT
    FT
 F

Solution Indices

1 4
2 6
3 1
4 3
5 5
6 2

Out of 100 runs, SAHCV3 found the optimal solution 20 times.
Average Steps to an optimal solution 11
Average Steps for non-optimal solution 106
------------------------------------------------------------
An optimal solution from using the SHC algorithm

F
T    F
   F
 F TTT
    FT
  F

Solution Indices

1 1
2 6
3 4
4 2
5 5
6 3

Out of 100 runs, SHC found the optimal solution 27 times.
Average Steps to an optimal solution 6
Average Steps for non-optimal solution 10
------------------------------------------------------------
CURRENT FOREST -------> 6x6 WITH 2 TREES

    T
F F

 F  F
   FT
     F
------------------------------------------------------------
An optimal solution from using the SAHC algorithm

F   T
  F
    F
 F
   FT
     F

Solution Indices

1 1
2 3
3 5
4 2
5 4
6 6

Out of 100 runs, SAHC found the optimal solution 33 times.
Average Steps to an optimal solution 6
Average Steps for non-optimal solution 9
------------------------------------------------------------
An optimal solution from using the SAHCV2 algorithm

   FT
F
  F
    F
 F  T
     F

Solution Indices

1 4
2 1
3 3
4 5
5 2
6 6

Out of 100 runs, SAHCV2 found the optimal solution 95 times.
Average Steps to an optimal solution 15
Average Steps for non-optimal solution 49
------------------------------------------------------------
An optimal solution from using the SAHCV3 algorithm

   FTF
 F
    F

F   T
  F

Solution Indices

1 4
1 6
2 2
3 5
5 1
6 3

Out of 100 runs, SAHCV3 found the optimal solution 38 times.
Average Steps to an optimal solution 12
Average Steps for non-optimal solution 106
------------------------------------------------------------
An optimal solution from using the SHC algorithm

F   TF
  F
```

```
    F
 F
   FT


Solution Indices

1 1
1 6
2 3
3 5
4 2
5 4

Out of 100 runs, SHC found the optimal solution 40 times.
Average Steps to an optimal solution 6
Average Steps for non-optimal solution 10
------------------------------------------------------------
CURRENT FOREST -------> 4x4 WITH 10 TREES

 TTT
 TFT
TTTF
FFTT
------------------------------------------------------------
Out of 100 runs, SAHC found the optimal solution 0 times.
Average Steps to an optimal solution 0
Average Steps for non-optimal solution 3
------------------------------------------------------------
Out of 100 runs, SAHCV2 found the optimal solution 0 times.
Average Steps to an optimal solution 0
Average Steps for non-optimal solution 37
------------------------------------------------------------
Out of 100 runs, SAHCV3 found the optimal solution 0 times.
Average Steps to an optimal solution 0
Average Steps for non-optimal solution 100
------------------------------------------------------------
Out of 100 runs, SHC found the optimal solution 0 times.
Average Steps to an optimal solution 0
Average Steps for non-optimal solution 3
------------------------------------------------------------
CURRENT FOREST -------> 4x4 WITH 5 TREES

 F
T
FT F
TTF
------------------------------------------------------------
An optimal solution from using the SAHC algorithm

 F
T  F
FT
TTF

Solution Indices

1 2
2 4
3 1
4 3

Out of 100 runs, SAHC found the optimal solution 32 times.
Average Steps to an optimal solution 2
Average Steps for non-optimal solution 5
------------------------------------------------------------
An optimal solution from using the SAHCV2 algorithm

 F
T  F
FT
TTF

Solution Indices

1 2
2 4
3 1
4 3

Out of 100 runs, SAHCV2 found the optimal solution 100 times.
Average Steps to an optimal solution 5
Average Steps for non-optimal solution 0
------------------------------------------------------------
An optimal solution from using the SAHCV3 algorithm

 F
T  F
FT
TTF

Solution Indices

1 2
2 4
3 1
4 3

Out of 100 runs, SAHCV3 found the optimal solution 25 times.
Average Steps to an optimal solution 3
Average Steps for non-optimal solution 102
------------------------------------------------------------
An optimal solution from using the SHC algorithm

 F
T  F
FT
TTF

Solution Indices

1 2
2 4
3 1
4 3

Out of 100 runs, SHC found the optimal solution 30 times.
Average Steps to an optimal solution 3
Average Steps for non-optimal solution 5
------------------------------------------------------------
CURRENT FOREST -------> 4x4 WITH 2 TREES

 FF

F TF
  T
------------------------------------------------------------
An optimal solution from using the SAHC algorithm

  F
F
  TF
 FT

Solution Indices

1 3
2 1
3 4
4 2

Out of 100 runs, SAHC found the optimal solution 51 times.
Average Steps to an optimal solution 2
Average Steps for non-optimal solution 5
------------------------------------------------------------
An optimal solution from using the SAHCV2 algorithm

  F
F
  T
 FTF

Solution Indices

1 3
2 1
4 2
4 4

Out of 100 runs, SAHCV2 found the optimal solution 100 times.
Average Steps to an optimal solution 3
Average Steps for non-optimal solution 0
------------------------------------------------------------
An optimal solution from using the SAHCV3 algorithm

F
```

```
  F
  T
 FTF

Solution Indices

1 1
2 3
4 2
4 4

Out of 100 runs, SAHCV3 found the optimal solution 47 times.
Average Steps to an optimal solution 5
Average Steps for non-optimal solution 101
------------------------------------------------------------
An optimal solution from using the SHC algorithm

F
  F
  T
 FTF

Solution Indices

1 1
2 3
4 2
4 4

Out of 100 runs, SHC found the optimal solution 62 times.
Average Steps to an optimal solution 3
Average Steps for non-optimal solution 5
------------------------------------------------------------
```

```
AlmondJoy
Minimax vs. Minimax
Blue Moves: A1
Green Moves: B1
Blue Moves: C1
Green Moves: D1
Blue Moves: E1
Green Moves: F1
Blue Moves: B2
Green Moves: A2
Blue Moves: D2
Green Moves: C2
Blue Moves: F2
Green Moves: E2
Blue Moves: A3
Green Moves: B3
Blue Moves: C3
Green Moves: D3
Blue Moves: E3
Green Moves: F3
Blue Moves: B4
Green Moves: A4
Blue Moves: D4
Green Moves: C4
Blue Moves: F4
Green Moves: E4
Blue Moves: A5
Green Moves: B5
Blue Moves: C5
Green Moves: D5
Blue Moves: E5
Green Moves: F5
Blue Moves: B6
Green Moves: A6
Blue Moves: D6
Green Moves: C6
Blue Moves: F6
Green Moves: E6
B G B G B G
G B G B G B
B G B G B G
G B G B G B
B G B G B G
G B G B G B
Tie Game
Blue score: 36 Green score: 36
Blue Average Move Time: 604ms Green Average Move Time: 604ms
Blue Total Nodes: 5604698 Green Total Nodes: 4856839
Blue Average Nodes: 311372.1111111111 Green Average Nodes: 269824.3888888889
--------------------------
AlmondJoy
AlphaBeta vs. AlphaBeta
Blue Moves: A1
Green Moves: B1
Blue Moves: C1
Green Moves: D1
Blue Moves: E1
Green Moves: F1
Blue Moves: B2
Green Moves: A2
Blue Moves: D2
Green Moves: C2
Blue Moves: F2
Green Moves: E2
Blue Moves: A3
Green Moves: B3
Blue Moves: C3
Green Moves: D3
Blue Moves: E3
Green Moves: F3
Blue Moves: B4
Green Moves: A4
Blue Moves: D4
Green Moves: C4
Blue Moves: F4
Green Moves: E4
Blue Moves: A5
Green Moves: B5
Blue Moves: C5
Green Moves: D5
Blue Moves: E5
Green Moves: F5
Blue Moves: B6
Green Moves: A6
Blue Moves: D6
Green Moves: C6
Blue Moves: F6
Green Moves: E6
B G B G B G
G B G B G B
B G B G B G
G B G B G B
B G B G B G
G B G B G B
Tie Game
Blue score: 36 Green score: 36
Blue Average Move Time: 48ms Green Average Move Time: 48ms
Blue Total Nodes: 268523 Green Total Nodes: 234638
Blue Average Nodes: 14917.944444444445 Green Average Nodes: 13035.444444444445
--------------------------
AlmondJoy
AlphaBeta vs. Minimax
Blue Moves: A1
Green Moves: B1
Blue Moves: C1
Green Moves: D1
Blue Moves: E1
Green Moves: F1
Blue Moves: B2
Green Moves: A2
Blue Moves: D2
Green Moves: C2
Blue Moves: F2
Green Moves: E2
Blue Moves: A3
Green Moves: B3
Blue Moves: C3
Green Moves: D3
Blue Moves: E3
Green Moves: F3
Blue Moves: B4
Green Moves: A4
Blue Moves: D4
Green Moves: C4
Blue Moves: F4
Green Moves: E4
Blue Moves: A5
Green Moves: B5
Blue Moves: C5
Green Moves: D5
Blue Moves: E5
Green Moves: F5
Blue Moves: B6
Green Moves: A6
Blue Moves: D6
Green Moves: C6
Blue Moves: F6
Green Moves: E6
B G B G B G
G B G B G B
B G B G B G
G B G B G B
B G B G B G
G B G B G B
Tie Game
Blue score: 36 Green score: 36
Blue Average Move Time: 49ms Green Average Move Time: 49ms
Blue Total Nodes: 268523 Green Total Nodes: 4856839
Blue Average Nodes: 14917.944444444445 Green Average Nodes: 269824.3888888889
--------------------------
AlmondJoy
Minimax vs. AlphaBeta
Blue Moves: A1
Green Moves: B1
Blue Moves: C1
Green Moves: D1
Blue Moves: E1
Green Moves: F1
Blue Moves: B2
Green Moves: A2
Blue Moves: D2
Green Moves: C2
Blue Moves: F2
Green Moves: E2
Blue Moves: A3
Green Moves: B3
```

```
Blue Moves: C3
Green Moves: D3
Blue Moves: E3
Green Moves: F3
Blue Moves: B4
Green Moves: A4
Blue Moves: D4
Green Moves: C4
Blue Moves: F4
Green Moves: E4
Blue Moves: A5
Green Moves: B5
Blue Moves: C5
Green Moves: D5
Blue Moves: E5
Green Moves: F5
Blue Moves: B6
Green Moves: A6
Blue Moves: D6
Green Moves: C6
Blue Moves: F6
Green Moves: E6
B G B G B G
G B G B G B
B G B G B G
G B G B G B
B G B G B G
G B G B G B
Tie Game
Blue score: 36 Green score: 36
Blue Average Move Time: 590ms Green Average Move Time: 590ms
Blue Total Nodes: 5604698 Green Total Nodes: 234638
Blue Average Nodes: 311372.1111111111 Green Average Nodes: 13035.444444444445
--------------------------
Ayds
Minimax vs. Minimax
Blue Moves: A1
Green Moves: E1
Blue Moves: C1
Green Moves: D1
Blue Moves: B1
Green Moves: D2
Blue Moves: C2
Green Moves: E2
Blue Moves: B2
Green Moves: F2
Blue Moves: A3
Green Moves: C3
Blue Moves: B3
Green Moves: D3
Blue Moves: B4
Green Moves: C4
Blue Moves: A4
Green Moves: E3
Blue Moves: C5
Green Moves: F6
Blue Moves: A5
Green Moves: D4
Blue Moves: B6
Green Moves: D5
Blue Moves: B5
Green Moves: F4
Blue Moves: D6
Green Moves: E5
Blue Moves: F1
Green Moves: A2
Blue Moves: F3
Green Moves: E4
Blue Moves: F5
Green Moves: E6
Blue Moves: C6
Green Moves: A6
B B B G G B
G B B G G G
B B G G G B
B B G G G G
B B B G G B
G B B B G G
Tie Game
Blue score: 1800 Green score: 1800
Blue Average Move Time: 596ms Green Average Move Time: 596ms
Blue Total Nodes: 5604698 Green Total Nodes: 4856839
Blue Average Nodes: 311372.1111111111 Green Average Nodes: 269824.3888888889
--------------------------
Ayds
AlphaBeta vs. AlphaBeta
Blue Moves: A1
Green Moves: E1
Blue Moves: C1
Green Moves: D1
Blue Moves: B1
Green Moves: D2
Blue Moves: C2
Green Moves: E2
Blue Moves: B2
Green Moves: F2
Blue Moves: A3
Green Moves: C3
Blue Moves: B3
Green Moves: D3
Blue Moves: B4
Green Moves: C4
Blue Moves: A4
Green Moves: E3
Blue Moves: C5
Green Moves: F6
Blue Moves: A5
Green Moves: D4
Blue Moves: B6
Green Moves: D5
Blue Moves: B5
Green Moves: F4
Blue Moves: D6
Green Moves: E5
Blue Moves: F1
Green Moves: A2
Blue Moves: F3
Green Moves: E4
Blue Moves: F5
Green Moves: E6
Blue Moves: C6
Green Moves: A6
B B B G G B
G B B G G G
B B G G G B
B B G G G G
B B B G G B
G B B B G G
Tie Game
Blue score: 1800 Green score: 1800
Blue Average Move Time: 61ms Green Average Move Time: 61ms
Blue Total Nodes: 393018 Green Total Nodes: 373151
Blue Average Nodes: 21834.333333333332 Green Average Nodes: 20730.61111111111
--------------------------
Ayds
AlphaBeta vs. Minimax
Blue Moves: A1
Green Moves: E1
Blue Moves: C1
Green Moves: D1
Blue Moves: B1
Green Moves: D2
Blue Moves: C2
Green Moves: E2
Blue Moves: B2
Green Moves: F2
Blue Moves: A3
Green Moves: C3
Blue Moves: B3
Green Moves: D3
Blue Moves: B4
Green Moves: C4
Blue Moves: A4
Green Moves: E3
Blue Moves: C5
Green Moves: F6
Blue Moves: A5
Green Moves: D4
Blue Moves: B6
Green Moves: D5
Blue Moves: B5
Green Moves: F4
Blue Moves: D6
Green Moves: E5
Blue Moves: F1
Green Moves: A2
```

```
Blue Moves: F3
Green Moves: E4
Blue Moves: F5
Green Moves: E6
Blue Moves: C6
Green Moves: A6
B B B G G B
G B B G G G
B B G G G B
B B G G G G
B B B G G B
G B B B G G
Tie Game
Blue score: 1800 Green score: 1800
Blue Average Move Time: 61ms Green Average Move Time: 61ms
Blue Total Nodes: 393018 Green Total Nodes: 4856839
Blue Average Nodes: 21834.333333333332 Green Average Nodes: 269824.3888888889
--------------------------
Ayds
Minimax vs. AlphaBeta
Blue Moves: A1
Green Moves: E1
Blue Moves: C1
Green Moves: D1
Blue Moves: B1
Green Moves: D2
Blue Moves: C2
Green Moves: E2
Blue Moves: B2
Green Moves: F2
Blue Moves: A3
Green Moves: C3
Blue Moves: B3
Green Moves: D3
Blue Moves: B4
Green Moves: C4
Blue Moves: A4
Green Moves: E3
Blue Moves: C5
Green Moves: F6
Blue Moves: A5
Green Moves: D4
Blue Moves: B6
Green Moves: D5
Blue Moves: B5
Green Moves: F4
Blue Moves: D6
Green Moves: E5
Blue Moves: F1
Green Moves: A2
Blue Moves: F3
Green Moves: E4
Blue Moves: F5
Green Moves: E6
Blue Moves: C6
Green Moves: A6
B B B G G B
G B B G G G
B B G G G B
B B G G G G
B B B G G B
G B B B G G
Tie Game
Blue score: 1800 Green score: 1800
Blue Average Move Time: 595ms Green Average Move Time: 595ms
Blue Total Nodes: 5604698 Green Total Nodes: 373151
Blue Average Nodes: 311372.1111111111 Green Average Nodes: 20730.61111111111
--------------------------
Bit-O-Honey
Minimax vs. Minimax
Blue Moves: A6
Green Moves: B6
Blue Moves: C6
Green Moves: D6
Blue Moves: E6
Green Moves: F6
Blue Moves: C4
Green Moves: A5
Blue Moves: B5
Green Moves: E5
Blue Moves: D5
Green Moves: C5
Blue Moves: F5
Green Moves: C3
Blue Moves: A4
Green Moves: B4
Blue Moves: D3
Green Moves: F4
Blue Moves: E4
Green Moves: D4
Blue Moves: B3
Green Moves: E3
Blue Moves: C2
Green Moves: D2
Blue Moves: A2
Green Moves: B2
Blue Moves: E2
Green Moves: A3
Blue Moves: F3
Green Moves: F2
Blue Moves: B1
Green Moves: A1
Blue Moves: D1
Green Moves: C1
Blue Moves: F1
Green Moves: E1
G B G B G B
B G B G B G
G B G B G B
B G B G B G
G B G B G B
B G B G B G
Tie Game
Blue score: 378 Green score: 378
Blue Average Move Time: 599ms Green Average Move Time: 599ms
Blue Total Nodes: 5604698 Green Total Nodes: 4856839
Blue Average Nodes: 311372.1111111111 Green Average Nodes: 269824.3888888889
--------------------------
Bit-O-Honey
AlphaBeta vs. AlphaBeta
Blue Moves: A6
Green Moves: B6
Blue Moves: C6
Green Moves: D6
Blue Moves: E6
Green Moves: F6
Blue Moves: C4
Green Moves: A5
Blue Moves: B5
Green Moves: E5
Blue Moves: D5
Green Moves: C5
Blue Moves: F5
Green Moves: C3
Blue Moves: A4
Green Moves: B4
Blue Moves: D3
Green Moves: F4
Blue Moves: E4
Green Moves: D4
Blue Moves: B3
Green Moves: E3
Blue Moves: C2
Green Moves: D2
Blue Moves: A2
Green Moves: B2
Blue Moves: E2
Green Moves: A3
Blue Moves: F3
Green Moves: F2
Blue Moves: B1
Green Moves: A1
Blue Moves: D1
Green Moves: C1
Blue Moves: F1
Green Moves: E1
G B G B G B
B G B G B G
G B G B G B
B G B G B G
G B G B G B
B G B G B G
Tie Game
Blue score: 378 Green score: 378
Blue Average Move Time: 479ms Green Average Move Time: 479ms
Blue Total Nodes: 4466695 Green Total Nodes: 3987999
```

```
Blue Average Nodes: 248149.72222222222 Green Average Nodes: 221555.5
--------------------------
Bit-O-Honey
AlphaBeta vs. Minimax
Blue Moves: A6
Green Moves: B6
Blue Moves: C6
Green Moves: D6
Blue Moves: E6
Green Moves: F6
Blue Moves: C4
Green Moves: A5
Blue Moves: B5
Green Moves: E5
Blue Moves: D5
Green Moves: C5
Blue Moves: F5
Green Moves: C3
Blue Moves: A4
Green Moves: B4
Blue Moves: D3
Green Moves: F4
Blue Moves: E4
Green Moves: D4
Blue Moves: B3
Green Moves: E3
Blue Moves: C2
Green Moves: D2
Blue Moves: A2
Green Moves: B2
Blue Moves: E2
Green Moves: A3
Blue Moves: F3
Green Moves: F2
Blue Moves: B1
Green Moves: A1
Blue Moves: D1
Green Moves: C1
Blue Moves: F1
Green Moves: E1
G B G B G B
B G B G B G
G B G B G B
B G B G B G
G B G B G B
B G B G B G
Tie Game
Blue score: 378 Green score: 378
Blue Average Move Time: 478ms Green Average Move Time: 478ms
Blue Total Nodes: 4466695 Green Total Nodes: 4856839
Blue Average Nodes: 248149.72222222222 Green Average Nodes: 269824.3888888889
--------------------------
Bit-O-Honey
Minimax vs. AlphaBeta
Blue Moves: A6
Green Moves: B6
Blue Moves: C6
Green Moves: D6
Blue Moves: E6
Green Moves: F6
Blue Moves: C4
Green Moves: A5
Blue Moves: B5
Green Moves: E5
Blue Moves: D5
Green Moves: C5
Blue Moves: F5
Green Moves: C3
Blue Moves: A4
Green Moves: B4
Blue Moves: D3
Green Moves: F4
Blue Moves: E4
Green Moves: D4
Blue Moves: B3
Green Moves: E3
Blue Moves: C2
Green Moves: D2
Blue Moves: A2
Green Moves: B2
Blue Moves: E2
Green Moves: A3
Blue Moves: F3
Green Moves: F2
Blue Moves: B1
Green Moves: A1
Blue Moves: D1
Green Moves: C1
Blue Moves: F1
Green Moves: E1
G B G B G B
B G B G B G
G B G B G B
B G B G B G
G B G B G B
B G B G B G
Tie Game
Blue score: 378 Green score: 378
Blue Average Move Time: 602ms Green Average Move Time: 602ms
Blue Total Nodes: 5604698 Green Total Nodes: 3987999
Blue Average Nodes: 311372.1111111111 Green Average Nodes: 221555.5
--------------------------
Mounds
Minimax vs. Minimax
Blue Moves: C2
Green Moves: D2
Blue Moves: B3
Green Moves: E3
Blue Moves: B4
Green Moves: D5
Blue Moves: C5
Green Moves: B2
Blue Moves: A1
Green Moves: A2
Blue Moves: A3
Green Moves: E2
Blue Moves: C1
Green Moves: B1
Blue Moves: D1
Green Moves: E1
Blue Moves: F1
Green Moves: F2
Blue Moves: A4
Green Moves: F3
Blue Moves: B5
Green Moves: E5
Blue Moves: D6
Green Moves: E6
Blue Moves: C6
Green Moves: E4
Blue Moves: A5
Green Moves: F4
Blue Moves: A6
Green Moves: F5
Blue Moves: C3
Green Moves: D3
Blue Moves: C4
Green Moves: D4
Blue Moves: B6
Green Moves: F6
G G G G G G
B G B G G G
B B B G G G
B B G G G G
B B B G G G
B B B B G G
Green Wins by: 14
Blue score: 58 Green score: 86
Blue Average Move Time: 607ms Green Average Move Time: 607ms
Blue Total Nodes: 5604698 Green Total Nodes: 4856839
Blue Average Nodes: 311372.1111111111 Green Average Nodes: 269824.3888888889
--------------------------
Mounds
AlphaBeta vs. AlphaBeta
Blue Moves: C2
Green Moves: D2
Blue Moves: B3
Green Moves: E3
Blue Moves: B4
Green Moves: D5
Blue Moves: C5
Green Moves: B2
Blue Moves: A1
Green Moves: A2
Blue Moves: A3
Green Moves: E2
```

```
Blue Moves: C1
Green Moves: B1
Blue Moves: D1
Green Moves: E1
Blue Moves: F1
Green Moves: F2
Blue Moves: A4
Green Moves: F3
Blue Moves: B5
Green Moves: E5
Blue Moves: D6
Green Moves: E6
Blue Moves: C6
Green Moves: E4
Blue Moves: A5
Green Moves: F4
Blue Moves: A6
Green Moves: F5
Blue Moves: C3
Green Moves: D3
Blue Moves: C4
Green Moves: D4
Blue Moves: B6
Green Moves: F6
G G G G G G
B G B G G G
B B B G G G
B B G G G G
B B B G G G
B B B B G G
Green Wins by: 14
Blue score: 58 Green score: 86
Blue Average Move Time: 142ms Green Average Move Time: 142ms
Blue Total Nodes: 1175331 Green Total Nodes: 721471
Blue Average Nodes: 65296.166666666664 Green Average Nodes: 40081.72222222222
--------------------------
Mounds
AlphaBeta vs. Minimax
Blue Moves: C2
Green Moves: D2
Blue Moves: B3
Green Moves: E3
Blue Moves: B4
Green Moves: D5
Blue Moves: C5
Green Moves: B2
Blue Moves: A1
Green Moves: A2
Blue Moves: A3
Green Moves: E2
Blue Moves: C1
Green Moves: B1
Blue Moves: D1
Green Moves: E1
Blue Moves: F1
Green Moves: F2
Blue Moves: A4
Green Moves: F3
Blue Moves: B5
Green Moves: E5
Blue Moves: D6
Green Moves: E6
Blue Moves: C6
Green Moves: E4
Blue Moves: A5
Green Moves: F4
Blue Moves: A6
Green Moves: F5
Blue Moves: C3
Green Moves: D3
Blue Moves: C4
Green Moves: D4
Blue Moves: B6
Green Moves: F6
G G G G G G
B G B G G G
B B B G G G
B B G G G G
B B B G G G
B B B B G G
Green Wins by: 14
Blue score: 58 Green score: 86
Blue Average Move Time: 143ms Green Average Move Time: 143ms
Blue Total Nodes: 1175331 Green Total Nodes: 4856839
Blue Average Nodes: 65296.166666666664 Green Average Nodes: 269824.3888888889
--------------------------
Mounds
Minimax vs. AlphaBeta
Blue Moves: C2
Green Moves: D2
Blue Moves: B3
Green Moves: E3
Blue Moves: B4
Green Moves: D5
Blue Moves: C5
Green Moves: B2
Blue Moves: A1
Green Moves: A2
Blue Moves: A3
Green Moves: E2
Blue Moves: C1
Green Moves: B1
Blue Moves: D1
Green Moves: E1
Blue Moves: F1
Green Moves: F2
Blue Moves: A4
Green Moves: F3
Blue Moves: B5
Green Moves: E5
Blue Moves: D6
Green Moves: E6
Blue Moves: C6
Green Moves: E4
Blue Moves: A5
Green Moves: F4
Blue Moves: A6
Green Moves: F5
Blue Moves: C3
Green Moves: D3
Blue Moves: C4
Green Moves: D4
Blue Moves: B6
Green Moves: F6
G G G G G G
B G B G G G
B B B G G G
B B G G G G
B B B G G G
B B B B G G
Green Wins by: 14
Blue score: 58 Green score: 86
Blue Average Move Time: 602ms Green Average Move Time: 602ms
Blue Total Nodes: 5604698 Green Total Nodes: 721471
Blue Average Nodes: 311372.1111111111 Green Average Nodes: 40081.72222222222
--------------------------
ReesesPieces
Minimax vs. Minimax
Blue Moves: A3
Green Moves: C6
Blue Moves: A5
Green Moves: D5
Blue Moves: B1
Green Moves: A6
Blue Moves: B3
Green Moves: E6
Blue Moves: C5
Green Moves: B5
Blue Moves: B4
Green Moves: B6
Blue Moves: A1
Green Moves: F6
Blue Moves: D1
Green Moves: A4
Blue Moves: F3
Green Moves: E5
Blue Moves: C2
Green Moves: C3
Blue Moves: C4
Green Moves: D4
Blue Moves: D3
Green Moves: E4
Blue Moves: F4
Green Moves: F5
Blue Moves: B2
Green Moves: A2
```

```
Blue Moves: E2
Green Moves: C1
Blue Moves: D6
Green Moves: E1
Blue Moves: F1
Green Moves: F2
Blue Moves: D2
Green Moves: E3
B B G B G B
G B B B B G
B B B B G B
G B G G B G
B G B G G G
G G G B G G
Blue wins by: 215
Blue score: 934 Green score: 719
Blue Average Move Time: 602ms Green Average Move Time: 602ms
Blue Total Nodes: 5604698 Green Total Nodes: 4856839
Blue Average Nodes: 311372.1111111111 Green Average Nodes: 269824.3888888889
--------------------------
ReesesPieces
AlphaBeta vs. AlphaBeta
Blue Moves: A3
Green Moves: C6
Blue Moves: A5
Green Moves: D5
Blue Moves: B1
Green Moves: A6
Blue Moves: B3
Green Moves: E6
Blue Moves: C5
Green Moves: B5
Blue Moves: B4
Green Moves: B6
Blue Moves: A1
Green Moves: F6
Blue Moves: D1
Green Moves: A4
Blue Moves: F3
Green Moves: E5
Blue Moves: C2
Green Moves: C3
Blue Moves: C4
Green Moves: D4
Blue Moves: D3
Green Moves: E4
Blue Moves: F4
Green Moves: F5
Blue Moves: B2
Green Moves: A2
Blue Moves: E2
Green Moves: C1
Blue Moves: D6
Green Moves: E1
Blue Moves: F1
Green Moves: F2
Blue Moves: D2
Green Moves: E3
B B G B G B
G B B B B G
B B B B G B
G B G G B G
B G B G G G
G G G B G G
Blue wins by: 215
Blue score: 934 Green score: 719
Blue Average Move Time: 112ms Green Average Move Time: 112ms
Blue Total Nodes: 854246 Green Total Nodes: 1023892
Blue Average Nodes: 47458.11111111111 Green Average Nodes: 56882.88888888889
--------------------------
ReesesPieces
AlphaBeta vs. Minimax
Blue Moves: A3
Green Moves: C6
Blue Moves: A5
Green Moves: D5
Blue Moves: B1
Green Moves: A6
Blue Moves: B3
Green Moves: E6
Blue Moves: C5
Green Moves: B5
Blue Moves: B4
Green Moves: B6
Blue Moves: A1
Green Moves: F6
Blue Moves: D1
Green Moves: A4
Blue Moves: F3
Green Moves: E5
Blue Moves: C2
Green Moves: C3
Blue Moves: C4
Green Moves: D4
Blue Moves: D3
Green Moves: E4
Blue Moves: F4
Green Moves: F5
Blue Moves: B2
Green Moves: A2
Blue Moves: E2
Green Moves: C1
Blue Moves: D6
Green Moves: E1
Blue Moves: F1
Green Moves: F2
Blue Moves: D2
Green Moves: E3
B B G B G B
G B B B B G
B B B B G B
G B G G B G
B G B G G G
G G G B G G
Blue wins by: 215
Blue score: 934 Green score: 719
Blue Average Move Time: 110ms Green Average Move Time: 110ms
Blue Total Nodes: 854246 Green Total Nodes: 4856839
Blue Average Nodes: 47458.11111111111 Green Average Nodes: 269824.3888888889
--------------------------
ReesesPieces
Minimax vs. AlphaBeta
Blue Moves: A3
Green Moves: C6
Blue Moves: A5
Green Moves: D5
Blue Moves: B1
Green Moves: A6
Blue Moves: B3
Green Moves: E6
Blue Moves: C5
Green Moves: B5
Blue Moves: B4
Green Moves: B6
Blue Moves: A1
Green Moves: F6
Blue Moves: D1
Green Moves: A4
Blue Moves: F3
Green Moves: E5
Blue Moves: C2
Green Moves: C3
Blue Moves: C4
Green Moves: D4
Blue Moves: D3
Green Moves: E4
Blue Moves: F4
Green Moves: F5
Blue Moves: B2
Green Moves: A2
Blue Moves: E2
Green Moves: C1
Blue Moves: D6
Green Moves: E1
Blue Moves: F1
Green Moves: F2
Blue Moves: D2
Green Moves: E3
B B G B G B
G B B B B G
B B B B G B
G B G G B G
B G B G G G
G G G B G G
Blue wins by: 215
Blue score: 934 Green score: 719
```

```
Blue Average Move Time: 598ms Green Average Move Time: 598ms
Blue Total Nodes: 5604698 Green Total Nodes: 1023892
Blue Average Nodes: 311372.1111111111 Green Average Nodes: 56882.88888888889
--------------------------
```