CS 440 Assignment 2

R section 3 credits

Group members:
Chendi Lin, Zhuoyue Wang

1. Overview

In this assignment, we try to use backtracking searching and several algorithms to solve constraint satisfaction problems and games.

In the first part, we implemented three different CSP formulations for comparison. Dumb formulation is the one with randomized variable order and randomized value assignment order with no forward checking. Smart formulation has some forward checking and worked well enough to solve all 3-credit puzzles. The smarter formulation, which is called SmartEc in our implementation, has more clever pruning strategies and more comprehensive forward checking, which can solve all 4-credit puzzles and some extra puzzles. Their performance and efficiency comparisons are also discussed in that section.

In the second part, we implements a simple two-player zero-sum game called Breakthrough, and use minimax search and alpha-beta search to simulate two players' actions. Besides these, we also create two original evaluation heuristic functions, "Defensive" one and "Offensive" one to help agents find best-fit actions. To beat the these "dumb" functions, we build two new heuristic functions which considers both players' performance and defeat the original functions successfully. In the 4-credit problem, we extend the game rule and change the board size from the square into a long rectangle. Then we apply the same evaluation functions to this game and find some different results compared with those in the 3-credit problem. We will discuss the difference in the following sections.

2. Work distribution

Chendi Lin: Flow Free problems, report

Zhuoyue Wang: Breakthrough game problems, report

3. Part 1: CSP - Flow Free

In this project, we formulate the problem as a CSP, in which the variables are the colors, and the value domain is all the empty grid cells. Basically, we want to find out, for each source pair, which cells can be assigned to this color. The constraints include: 1. The pipes do not intersect with each other, 2. All cells have to be assigned a color, 3.

Each non-source cell can have only two neighbors with the same color, 4. Each source cell can have only one neighbor with the same color.

Since all the work are coded in **Python**, run on **Mac Air**, it might take significantly longer than C++ code. But with appropriate forward checking and pruning, the consumed time is acceptable.

3.1. Dumb Solution

For the dumb solution, since no ordering and no forward checking is allowed, we randomize the order of colors and the order of neighbors to be added in the frontier. For 5*5 puzzle, it solves the problem pretty quickly.

```
done
98
0.006290912628173828
['B', 'R', 'R', 'R', 'O']
['B', 'R', 'Y', 'Y', 'O']
['B', 'R', 'Y', 'O', 'O']
['B', 'R', 'O', 'O', 'G']
['B', 'B', 'G', 'G', 'G']
```

98 nodes were expanded, and 0.00629 seconds was taken. Since the orders are randomized, the number of expanded nodes and execution time varied every time.

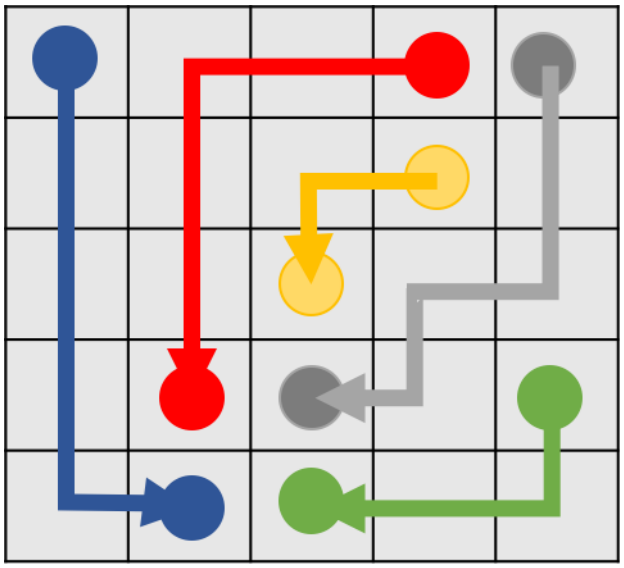

Figure 1. Solution of 5*5 Puzzle

It can also solve 7*7 puzzle

```
done
47904
5.2850501537323
['G', 'G', 'G', 'O', 'O', 'O', 'O']
['G', 'B', 'G', 'G', 'G', 'Y', 'O']
['G', 'B', 'B', 'B', 'R', 'Y', 'O']
['G', 'Y', 'Y', 'Y', 'R', 'Y', 'O']
['G', 'Y', 'R', 'R', 'R', 'Y', 'O']
['G', 'Y', 'R', 'Y', 'Y', 'Y', 'O']
['G', 'Y', 'Y', 'Y', 'O', 'O', 'O']
```

47904 nodes were expanded, and 5.2850 seconds was taken. Since the orders are randomized, the number of expanded nodes and execution time varied every time.

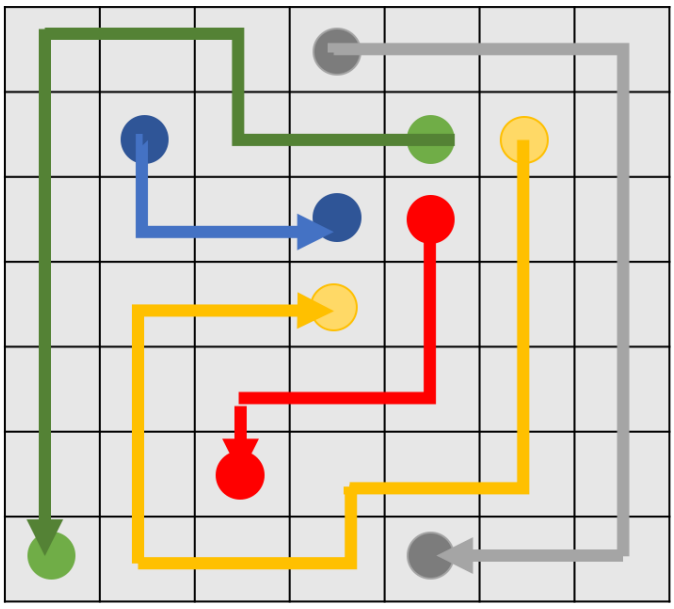

Figure 2. Solution of 7*7 Puzzle

8*8 Puzzle can also be solved using the dump implementation, but with relatively long time.

```
done
201879
26.556051015853882
['Y', 'Y', 'Y', 'R', 'R', 'R', 'G', 'G']
['Y', 'B', 'Y', 'P', 'P', 'R', 'R', 'G']
['Y', 'B', 'O', 'O', 'P', 'G', 'R', 'G']
['Y', 'B', 'O', 'P', 'P', 'G', 'G', 'G']
['Y', 'B', 'O', 'O', 'O', 'O', 'Y', 'Y']
['Y', 'B', 'B', 'B', 'B', 'O', 'Q', 'Y']
['Y', 'Q', 'Q', 'Q', 'Q', 'Q', 'Q', 'Y']
['Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y']
```

201879 nodes were expanded, and 26.556 seconds was taken. Since the orders are randomized, the number of expanded nodes and execution time varied every time.

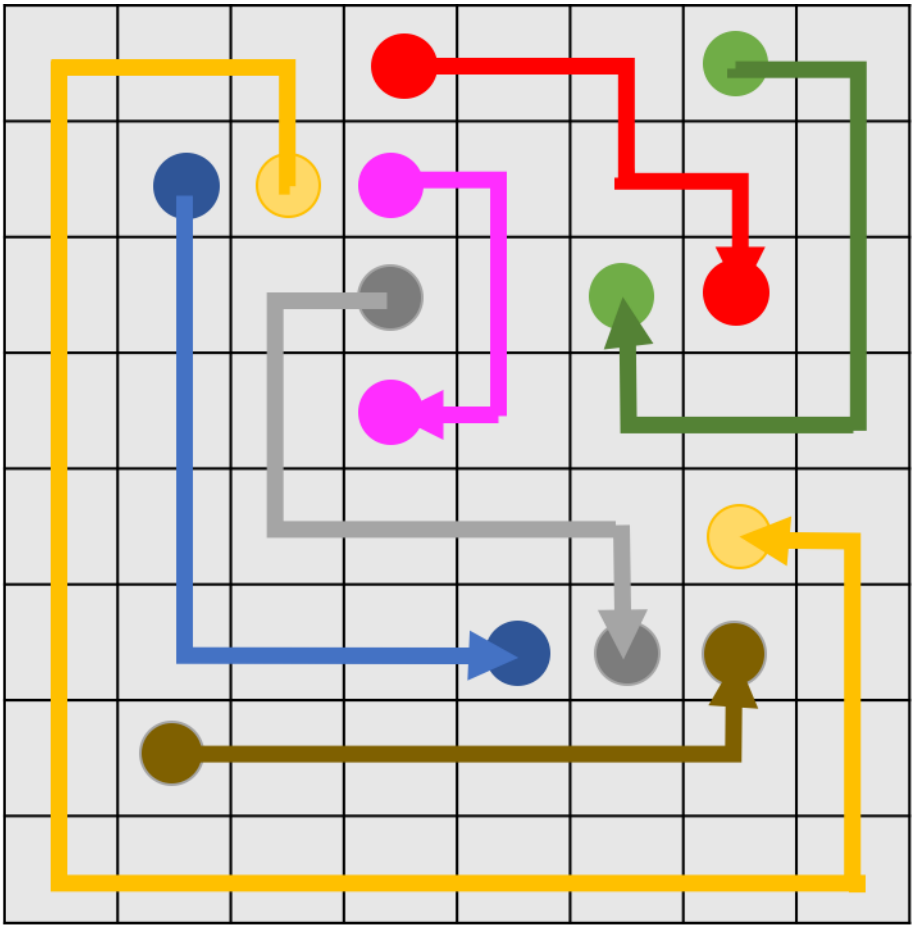

Figure 3. Solution of 8*8 Puzzle

It took too long to solve a 9*9 puzzle, so it is not discussed here.

3.2. Smart Solution

To solve larger puzzle, we mainly made two changes: order of variables, and a simple forward checking at each step.

We order the variable in this way: when reading the puzzle, whenever we encounter a source node for the first time, we put it in a list called “sourceA”, so the other source node in the same pair will be in the list called “sourceB”. Then we choose which color we want to use to fill in the cells with the order in “sourceA”. The order works well because the source cells on the relatively front positions of ”sourceA” tend to be closer to the corner or closer to the wall, while the cells at the end of “sourceA” are usually near center. So when we start assigning colors to the neighbors of the source node, the empty cells, which are the values in our formulation, will give less constraints if they are located next to the wall. It is basically a Least Constraining Assignment Heuristic, but in a preliminary way.

What’s more important is the forward checking implemented here. At each step, we check if there is still a path connecting the unpaired source node, by BFS. It prevents a huge amount of invalid assignments at early stage. We will discuss the efficiency that it brought in the next subsection.

3.2.1 3-Credit

With the smart formulation, 7*7, 8*8, and 9*9 puzzles can be solved in a fairly short amount of time. For 7*7 puzzle

```
done
2030
1.2097079753875732
['G', 'G', 'G', '0', '0', '0', '0']
['G', 'B', 'G', 'G', 'G', 'Y', '0']
['G', 'B', 'B', 'B', 'R', 'Y', '0']
['G', 'Y', 'Y', 'Y', 'R', 'Y', '0']
['G', 'Y', 'R', 'R', 'R', 'Y', '0']
['G', 'Y', 'R', 'Y', 'Y', 'Y', '0']
['G', 'Y', 'Y', 'Y', '0', '0', '0']
```

Only 2030 nodes were expanded, around one-tenth of the dumb case, and 1.2097 second was taken, way faster than 5 seconds in the dumb case.

For 8*8 puzzle

```
done
2621
1.0458309650421143
['Y', 'Y', 'Y', 'R', 'R', 'R', 'G', 'G']
['Y', 'B', 'Y', 'P', 'P', 'R', 'R', 'G']
['Y', 'B', 'O', 'O', 'P', 'G', 'R', 'G']
['Y', 'B', 'O', 'P', 'P', 'G', 'G', 'G']
['Y', 'B', 'O', 'O', 'O', 'O', 'Y', 'Y']
['Y', 'B', 'B', 'B', 'B', 'O', 'Q', 'Y']
['Y', 'Q', 'Q', 'Q', 'Q', 'Q', 'Q', 'Y']
['Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y']
```

Only 2621 nodes were expanded, around 1/420 of the dumb case, and 1.0458 second was taken, around 1/100 of the dumb case.

9*9 can also be solved fairly quickly this way.

```
done
22432
25.579658031463623
['D', 'B', 'B', 'B', 'O', 'K', 'K', 'K', 'K']
['D', 'B', 'O', 'O', 'O', 'R', 'R', 'R', 'K']
['D', 'B', 'R', 'Q', 'Q', 'Q', 'Q', 'R', 'K']
['D', 'B', 'R', 'R', 'R', 'R', 'R', 'R', 'K']
['G', 'G', 'K', 'K', 'K', 'K', 'K', 'K', 'K']
['G', 'K', 'K', 'P', 'P', 'P', 'P', 'P', 'G']
['G', 'K', 'Y', 'Y', 'Y', 'Y', 'Y', 'P', 'G']
['G', 'K', 'K', 'K', 'K', 'K', 'K', 'P', 'G']
['G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G']
```

Only 22432 nodes were expanded, and 25.5797 seconds was taken. It is interesting to notice that, even when the forward checking is very effective in this special test, a lot of BFS needs to be done to prune the unnecessary braches, for a larger puzzle, so the time taken is longer.

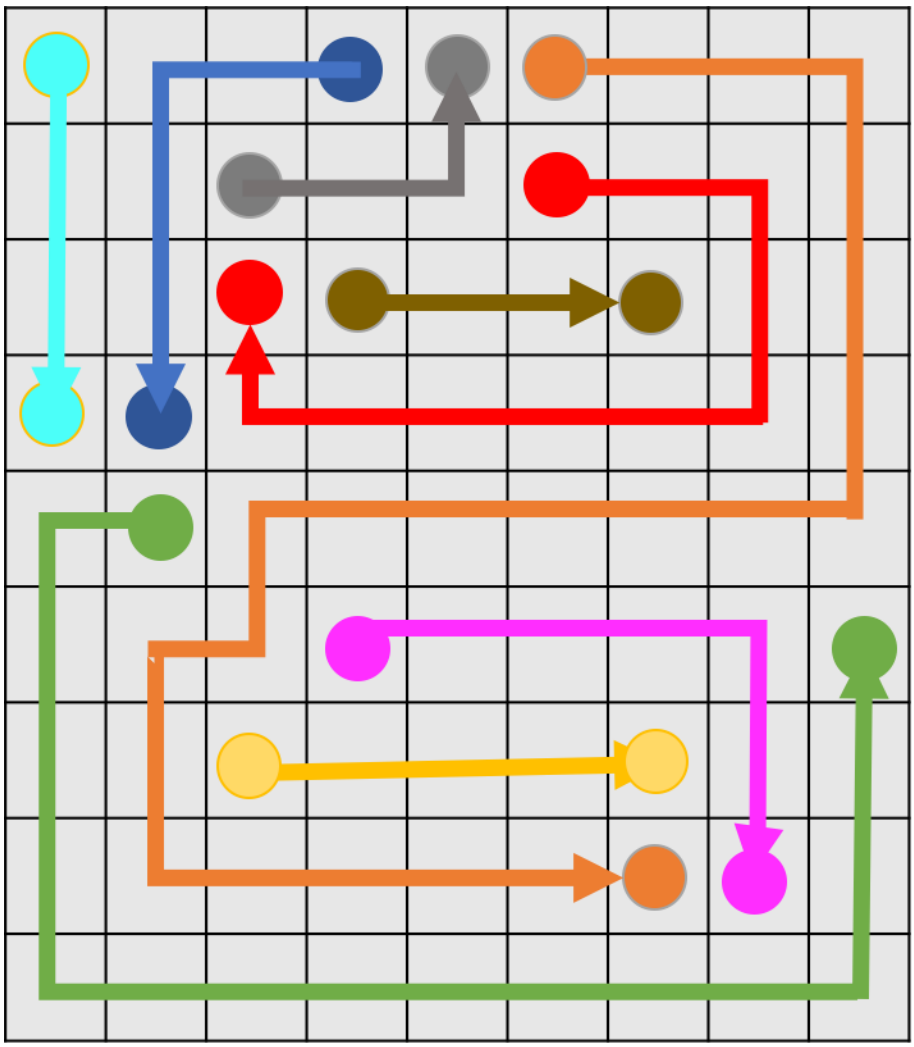

Figure 4. Solution of 9*9 Puzzle

3.2.2 4-Credit (Bonus)

Smart formulation is not good enough to solve the first 10*10 nor the larger puzzles. We have to use some more advanced method to prune.

In the “smarter” formulation, much more forward checking and clever pruning methods are utilized.

1) After one pair of source nodes are connected, we check if the new created pipe create a dead end for other empty cells. In another words, since each non-source cell needs exactly two neighbors with the same color, if the new pipe surround an empty cell and leave only one or none of its neighbors empty, we consider it as invalid and cut the branch.
2) At each step we check if the newly-created path violates the constraint that each non-source node have to have exactly two neighbors with the same color. If so, we cut the branch.
3) As before, at each step, we check if there is still a path connecting the unpaired source node, by BFS. If not, cut the branch.
4) At each step, we check that if there are isolated empty cell blocks created by the existing paths. This one is used to guarantee that every cell will take a color, and if empty cell block is created, obviously it would be an invalid assignment, so we cut the branch. This is also done by BFS. The logic behind it is that, on the puzzle, we search that if there exists an empty cell, we mark it and all the empty cells can be reached by a BFS from this cell with. If this block is connected to at least a pair of source nodes with the same color, this block is valid, and we search the next empty cell without the mark. If this block cannot reach at least one pair of source node, cut the branch. This way is super effective, especially in the second test of 10*10 puzzle.

Using the smarter formulation to solve the first 10*10 puzzle, we have

```
done
391
2.1945579051971436
['R', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G']
['R', 'R', 'R', 'R', 'O', 'O', 'O', 'O', 'O', 'G']
['Y', 'Y', 'P', 'R', 'Q', 'Q', 'Q', 'Q', 'Q', 'G']
['Y', 'P', 'P', 'R', 'R', 'R', 'R', 'R', 'R', 'G']
['Y', 'P', 'G', 'G', 'B', 'B', 'B', 'B', 'R', 'G']
['Y', 'P', 'P', 'G', 'B', 'R', 'R', 'B', 'R', 'G']
['Y', 'Y', 'P', 'G', 'B', 'R', 'B', 'B', 'R', 'G']
['P', 'Y', 'P', 'G', 'B', 'R', 'R', 'R', 'R', 'G']
['P', 'Y', 'P', 'G', 'B', 'B', 'B', 'B', 'B', 'G']
['P', 'P', 'P', 'G', 'G', 'G', 'G', 'G', 'G', 'G']
```

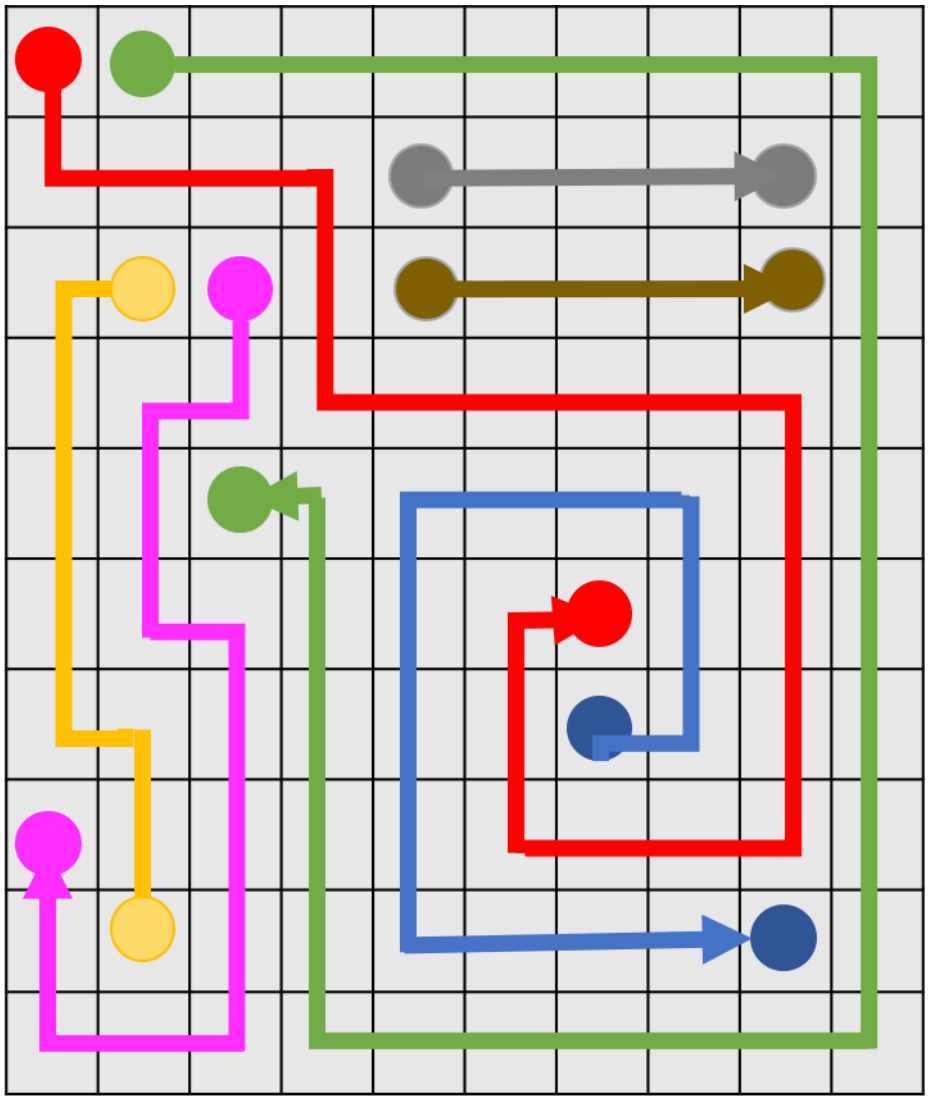

Figure 5. Solution of 10*10 Puzzle Case 1

Amazingly, only 391 nodes were expanded, around 1/1086 of the smart case, which is very impressive, and 2.194seconds was taken, around 1/150 of the smart case. It worked so well partially because of the great pruning strategy, and also because of the nice distribution of source nodes in “10101.txt”. While in the second test, it got more complicated. Smart formulation took forever, may be never, to solve the problem. However, using the smarter formulation, we have

```
done
14358
202.2185640335083
['T', 'T', 'T', 'P', 'P', 'P', 'P', 'P', 'P', 'P']
['T', 'B', 'T', 'P', 'F', 'F', 'F', 'F', 'F', 'P']
['T', 'B', 'T', 'P', 'F', 'B', 'T', 'V', 'F', 'P']
['T', 'B', 'B', 'B', 'B', 'B', 'T', 'V', 'F', 'P']
['T', 'T', 'T', 'T', 'T', 'T', 'T', 'V', 'F', 'P']
['F', 'N', 'N', 'N', 'N', 'N', 'N', 'V', 'F', 'F']
['F', 'N', 'S', 'S', 'S', 'S', 'N', 'V', 'V', 'F']
['F', 'N', 'S', 'N', 'H', 'S', 'N', 'H', 'V', 'F']
['F', 'N', 'N', 'N', 'H', 'H', 'H', 'H', 'V', 'F']
['F', 'F', 'F', 'F', 'F', 'F', 'F', 'F', 'F', 'F']
```

Only 14358 nodes were expanded, and 202.22 seconds was taken. It is a pretty good result considering it is run on Python.

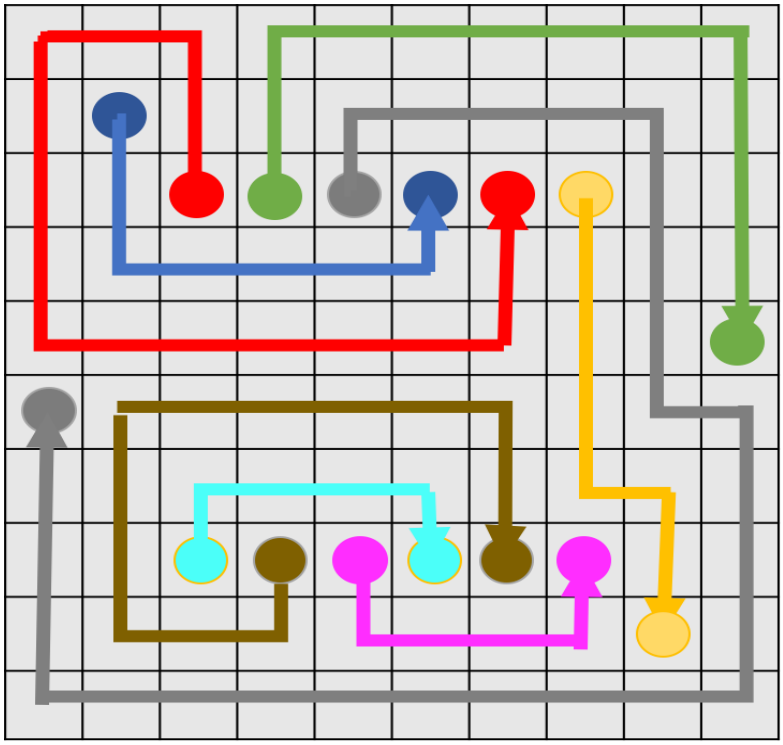

Figure 6. Solution of 10*10 Puzzle case 2

3.3. Bonus

12*12 puzzle can also be solved this way.

237869 nodes were expanded, and 8258 seconds was taken.

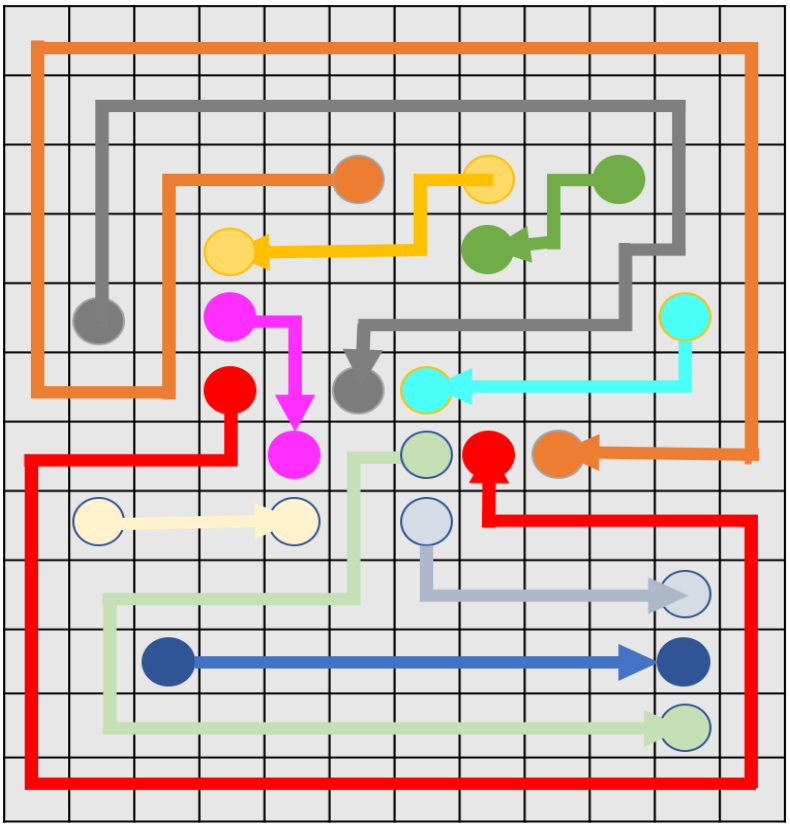

Figure 7. Solution of 12*12 Puzzle

4. Part 2: Game of Breakthrough

In this section, we tried to implement a simple two-player zero-sum game called Breakthrough, and used minimax search and alpha-beta search to simulate two players' actions. Besides these, we also created two evaluation functions, "Defensive" one and "Offensive" one to help agents find best-fit actions.

These are our evaluation functions:

Defensive Heuristic 1:
6*(my remaining pieces) + random()
Offensive Heuristic 1:
6*(30-opponent's remaining pieces) +random()
Defensive Heuristic 2:

6*(my remaining piece+(60-total sum of distances from my each piece to the opponent's base)) - 2*(opponent's remaining piece+(60-total sum of distances from opponent's each piece to my base))+random()

Offensive Heuristic 2:

2*(my remaining pieces+(60-total sum of distances from my each piece to the opponent's base)) - 6*(opponent's remaining pieces+(60-total sum of distances from opponent's each piece to my base))+random()

Different from Offensive and Defensive Heuristic 1, our Offensive and Defensive Heuristic 2 not only check the remaining pieces in one side, but also calculate the difference between opponent's remaining pieces and my remaining pieces. Moreover, we consider the total sum of the distance from each piece on one side to the other's base. We combine them as the component for defensive part and offensive part, as the function above shown.

For Defensive Heuristic 2, we multiply larger coefficient on one defensive component, the sum of my remaining piece and difference between a constant and the total distance from my each piece to the opponent's base, because we want give priority to consider keeping my remaining piece.

At the same time, we notice that it is also important to consider the closeness of victory according to our piece's performance. In our calculation, when our pieces have less distance to the opponent's base, it likely means it almost reach the victory. Therefore, we believe it is desirable to set (60-total sum of distances from opponent's each piece to my base) into the defensive component.

On the other hand, we also considered the opponent's performance in order to defeat the Offensive Heuristic 1. When the opponent has more remaining pieces and become closer to our base, we tend to become more conservative, and vice versa in the case that the opponent has less pieces and far from our base.

Offensive Heuristic 2 follow the same strategy as the Defensive Heuristic 2 function to beat Defensive Heuristic 1. To save some space, we will leave it here and discuss more about the result and performance in the next section.

In general, Heuristic 2 functions are not that complicated but they worked pretty well unexpectedly. The reason behind is that, when we only count our remaining pieces or only count the opponent's remaining pieces, it is possible that we choose the move that keep most of our pieces but also keep the opponent's pieces alive, or we may choose the move that eliminate most of the opponent's pieces but also kill most of our pieces. Either move is undesirable. Taking the difference of our remaining pieces and considering both sides' performance into account is a more reasonable way. What's more, we also need to approximate the completion process of the game in order to adjust our degree of defense or offense. To realize this goal, we check the distance

between pieces to the base in both sides and apply them into each of component. We believe this implementation matches human's chess logic closer.

2.1 Minimax and alpha-beta agents

```
[1, 1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1, 1]
[0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0]
[2, 2, 2, 2, 2, 2, 2, 2]
[2, 2, 2, 2, 2, 2, 2, 2]
```

1 —> Black
2 —> White
0 —> empty

1. Minimax (Offensive Heuristic 1) vs Alpha-beta (Offensive Heuristic 1)

Winner:
White: Alpha-beta (Offensive Heuristic 1)

White:
Total steps: 25
Total expand game tree nodes: 13895961
Average expanded nodes per move: 555838.4
Average time to make a move: 13.8
Number of opponent captured: 11

```
[0, 0, 1, 1, 1, 2, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0]
[2, 2, 2, 2, 2, 2, 2, 2]
```

Black:
Total steps: 25
Total expand game tree nodes: 349520
Average expanded nodes per move: 13980.8
Average time to make a move: 0.17
Number of opponent captured: 7

2. Alpha-beta (Offensive Heuristic 2) vs Alpha-beta (Defensive Heuristic 1)

Winner:
Black: Alpha-beta (Offensive Heuristic 2)

White:
Total steps: 25
Total expand game tree nodes: 3675522
Average expanded nodes per move: 2162071.7
Average time to make a move: 15.0
Number of opponent captured: 3

```
[1, 1, 1, 1, 1, 1, 1, 1]
[0, 0, 0, 0, 0, 0, 1, 1]
[0, 0, 0, 2, 0, 0, 0, 0]
[0, 0, 0, 0, 2, 0, 2, 2]
[2, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 1, 0, 2, 2, 2]
[0, 0, 0, 0, 0, 2, 0, 0]
[2, 1, 2, 2, 2, 0, 0, 0]
```

Black:
Total steps: 26
Total expand game tree nodes: 4041409
Average expanded nodes per move: 224522.5
Average time to make a move: 87.4
Number of opponent captured: 3

3. Alpha-beta (Defensive Heuristic 2) vs Alpha-beta (Offensive Heuristic 1)

Winner:
Black: Alpha-beta (Defensive Heuristic 2)

White:
Total steps: 14
Total expand game tree nodes:6854829
Average expanded nodes per move: 489630.6
Average time to make a move: 14.7
Number of opponent captured: 1

```
[1, 1, 1, 1, 1, 1, 1, 1]
[0, 0, 0, 0, 1, 0, 1, 1]
[0, 0, 0, 1, 1, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0]
[0, 2, 0, 0, 0, 2, 0, 2]
[0, 0, 0, 2, 2, 2, 0, 0]
[1, 0, 0, 2, 2, 2, 2, 2]
```

Black:

Total steps: 15
Total expand game tree nodes: 914810
Average expanded nodes per move: 60987.3
Average time to make a move: 4.0
Number of opponent captured: 5

4. Alpha-beta (Offensive Heuristic 2) vs Alpha-beta (Offensive Heuristic 1)

Winner:
Black: Alpha-beta (Offensive Heuristic 2)

White:
Total steps: 35
Total expand game tree nodes: 14955900
Average expanded nodes per move: 427311.4
Average time to make a move: 12.3
Number of opponent captured: 9

```
[0, 1, 1, 0, 1, 0, 1, 1]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 2, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 2, 0, 2, 0, 0, 0]
[1, 0, 2, 0, 0, 0, 2, 0]
```

Black:
Total steps: 36
Total expand game tree nodes: 4410448
Average expanded nodes per move: 122512.4
Average time to make a move: 5.7
Number of opponent captured: 11

5. Alpha-beta (Defensive Heuristic 2) vs Alpha-beta (Defensive Heuristic 1)

Winner:
Black: Alpha-beta (Defensive Heuristic 2)

White:
Total steps: 30
Total expand game tree nodes: 12209874
Average expanded nodes per move: 406995.8
Average time to make a move: 19.4
Number of opponent captured: 1

```
[0, 1, 1, 0, 0, 0, 1, 1]
[1, 0, 0, 0, 1, 1, 1, 0]
[2, 0, 2, 0, 0, 0, 0, 1]
[1, 0, 2, 0, 0, 2, 0, 0]
[1, 0, 0, 1, 2, 0, 2, 2]
[2, 1, 1, 2, 0, 2, 0, 2]
[0, 0, 0, 0, 0, 0, 2, 0]
[0, 2, 2, 0, 1, 2, 0, 0]
```

Black:
Total steps: 30
Total expand game tree nodes: 7444547
Average expanded nodes per move: 240146.7
Average time to make a move: 13.8
Number of opponent captured: 1

6. Alpha-beta (Offensive Heuristic 2) vs Alpha-beta (Defensive Heuristic 2)

Winner:
Black: Alpha-beta (Offensive Heuristic 2)

White:
Total steps: 25
Total expand game tree nodes: 13849007

```
[1, 1, 1, 1, 1, 1, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 1, 1, 0, 1, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[2, 0, 2, 2, 0, 2, 0, 0]
[0, 2, 2, 0, 0, 1, 0, 2]
[2, 2, 0, 0, 0, 2, 0, 2]
[0, 0, 0, 1, 0, 0, 0, 2]
```

Average expanded nodes per move: 553960.2
Average time to make a move: 22.9
Number of opponent captured: 4

Black:
Total steps: 26
Total expand game tree nodes: 23022326
Average expanded nodes per move: 885474.1
Average time to make a move: 32.2
Number of opponent captured: 4

According to the results, we can see that firstly using Alpha-beta search can defeat minimax search based on the same heuristic function. Secondly, our Heuristic 2 functions work pretty well and defeat the correspond Heuristic 1 functions as expected. The results above (Alpha-beta (Offensive Heuristic 2) vs Alpha-beta (Defensive Heuristic 1) and Alpha-beta (Defensive Heuristic 2) vs Alpha-beta (Offensive Heuristic 1) show that even the black player (Heuristic 2 functions) doesn't move any piece which is in the base and win in the end.

However, when these Heuristic 2 functions compete with their same type's Heuristic 1 functions, each player's total steps increase significantly and both have nearly the same number of eaten piece. It means that Heuristic 2 functions only have slight advantage when they occur the same style of evaluation function with them.

Moreover, when we run the last game Alpha-beta (Offensive Heuristic 2) vs Alpha-beta (Defensive Heuristic 2) for several times, we find Offensive Heuristic 2 has 80% chance to win (4/5). We believe it is because:

1. it expanded more nodes so it took more cases into considerations.
2. The winning strategy in this game is not to keep more pieces, but to kill all of opponent's pieces or to reach the other end. In this case, an offensive strategy may bring more advantages in this game.

2.2 Extended rules (bonus)

```
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
[2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
```

1 —> Black
2 —> White
0 —> empty

In this part, to compare the difference between both parts, we use the same evaluation functions as the previous part. In addition, for convenience, we will only show the matches with different results from previous ones: Alpha-beta (Offensive Heuristic 2) vs Alpha-beta (Defensive Heuristic 2) and Alpha-beta (Defensive Heuristic 2) vs Alpha-beta (Defensive Heuristic 1), which have unstable result.

1.1 Alpha-beta (Offensive Heuristic 2) vs Alpha-beta (Defensive Heuristic 2)

Winner:
Black: Alpha-beta (Offensive Heuristic 2)

White:
Total steps: 28
Total expand game tree nodes: 4147067
Average expanded nodes per move: 148109.5
Average time to make a move: 12.1
Number of opponent captured: 13

```
[1, 2, 1, 2, 0, 0, 0, 1, 0, 0]
[2, 0, 2, 0, 0, 0, 0, 1, 1, 0]
[0, 0, 0, 0, 0, 0, 0, 2, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 2, 0, 0]
[0, 1, 1, 0, 0, 0, 1, 0, 0, 0]
```

Black:
Total steps: 29
Total expand game tree nodes: 9540034
Average expanded nodes per move: 328966.7
Average time to make a move: 19.4
Number of opponent captured: 13

1.2 Alpha-beta (Offensive Heuristic 2) vs Alpha-beta (Defensive Heuristic 2)

Winner:
White: Alpha-beta (Defensive Heuristic 2)

White:
Total steps: 33
Total expand game tree nodes: 5854073
Average expanded nodes per move: 177396.2
Average time to make a move: 13.0
Number of opponent captured: 15

```
[0, 1, 2, 2, 0, 0, 2, 0, 1, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 2, 0, 2, 0, 0, 0, 0, 2, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
```

Black:
Total steps: 33
Total expand game tree nodes: 12609801
Average expanded nodes per move: 382115.2
Average time to make a move: 20.0
Number of opponent captured: 15

2.1. Alpha-beta (Defensive Heuristic 2) vs Alpha-beta (Defensive Heuristic 1)

Winner:
Black: Alpha-beta (Defensive Heuristic 2)

White:
Total steps: 12
Total expand game tree nodes: 4677491
Average expanded nodes per move: 389790.9
Average time to make a move: 32.4
Number of opponent captured: 6

```
[1, 1, 1, 1, 1, 1, 1, 0, 1, 0]
[1, 1, 1, 1, 1, 1, 1, 0, 0, 0]
[2, 0, 0, 0, 0, 0, 2, 2, 0, 1]
[2, 2, 2, 2, 2, 2, 0, 0, 0, 0]
[2, 0, 2, 2, 2, 2, 1, 0, 1, 1]
```

Black:
Total steps: 13
Total expand game tree nodes: 3661140
Average expanded nodes per move: 281626.2
Average time to make a move: 22.4

Number of opponent captured: 6

2.2. Alpha-beta (Defensive Heuristic 2) vs Alpha-beta (Defensive Heuristic 1)

Winner:
White: Alpha-beta (Defensive Heuristic 1)

White:
Total steps: 21
Total expand game tree nodes: 5237119
Average expanded nodes per move: 249386.6
Average time to make a move: 18.7
Number of opponent captured: 5

```
[0, 0, 1, 0, 1, 2, 1, 2, 2, 1]
[1, 1, 1, 0, 0, 1, 0, 1, 1, 0]
[1, 0, 0, 1, 0, 0, 0, 0, 0, 0]
[2, 2, 0, 2, 0, 2, 0, 2, 1, 0]
[0, 0, 2, 2, 2, 0, 2, 1, 1, 0]
```

Black:
Total steps: 21
Total expand game tree nodes: 4820775
Average expanded nodes per move: 229560.7
Average time to make a move: 17.1
Number of opponent captured: 5

By tests, in Alpha-beta (Offensive Heuristic 2) vs Alpha-beta (Defensive Heuristic 2), we find that Defensive Heuristic 2 has slightly higher possibility to win (4/7). But we notice that each match has a fairly fierce competition: when Black/White player wins, the opponent also has already had two pieces in his base. Therefore, we believe in such extended rule, there is no best choice to win other one. But we want to mention that Defensive Heuristic 2 expands nearly half of expanded nodes as Offensive Heuristic 2 has. So we still conclude that Defensive Heuristic 2 should be a more desirable choice when the game has time limit.

On the other hand, we find that Alpha-beta (Defensive Heuristic 2) vs Alpha-beta (Defensive Heuristic 1) also have different results. Similar to the previous case, Defensive Heuristic 2 has slightly higher possibility to win (3/5). But we also notice that even though Defensive Heuristic 1 wins in some cases, Defensive Heuristic 2 still has

two pieces reach its base. Therefore, we conclude that Defensive Heuristic 2 still has its advantage.

By these results, we notice an interesting difference that Defensive Heuristic function is more desirable in general. We guess the reason is that in such condition with extended rule and the long rectangle board, it is more common to engage in a battle to eat each other's pieces and. Also, in such extended rule, it is easier to reach the base and win in the case that three pieces get to the opponent's base. Therefore, the player should give priority to consider his remaining piece and focus on its key pieces that are closer to the base to increase the chance and efficiency to get victory.