Shruti chanumolu(shrutic2)
Anna Shewell(shewell2)

3 credit

Assignment 2: Constraint Satisfaction Problems and Games

Part 1: CSP - Flow Free

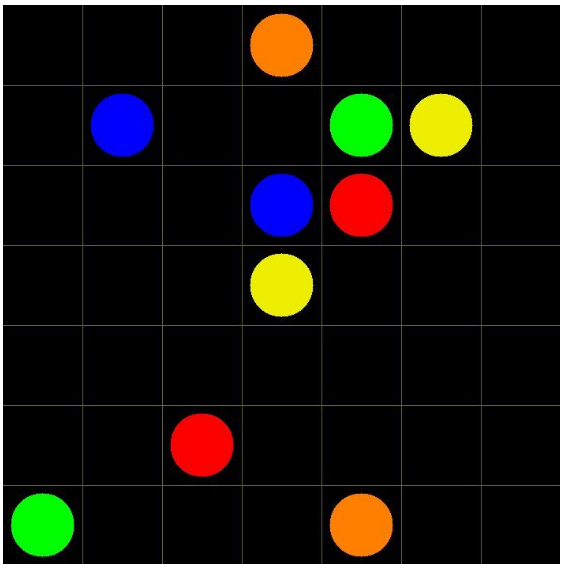
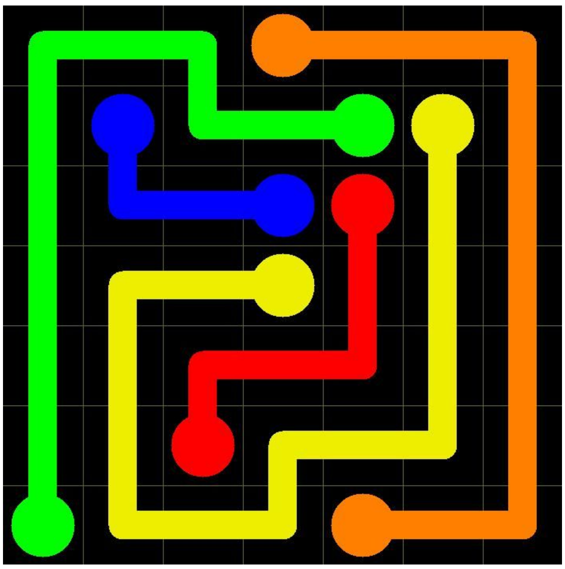
To formulate Flow Free as a CSP, we must define the variables, domain, and constraints for each problem. In general, we define:

Variables: All the non-source cells on the grid
Domain: All the colors
Constraints:

1. Sources must have exactly 1 neighbor with the same color.
2. Non-source cells must have exactly 2 neighbors with the same color.
3. All cells must be filled.
4. Every color must form a single path.

For each grid below, we've listed the specific variables and domain that pertain to that puzzle. Since the constraints are the same for every grid, we will not repeat them.

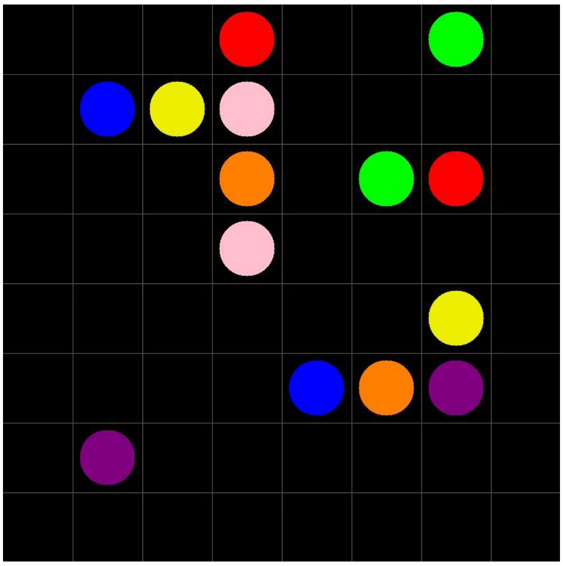
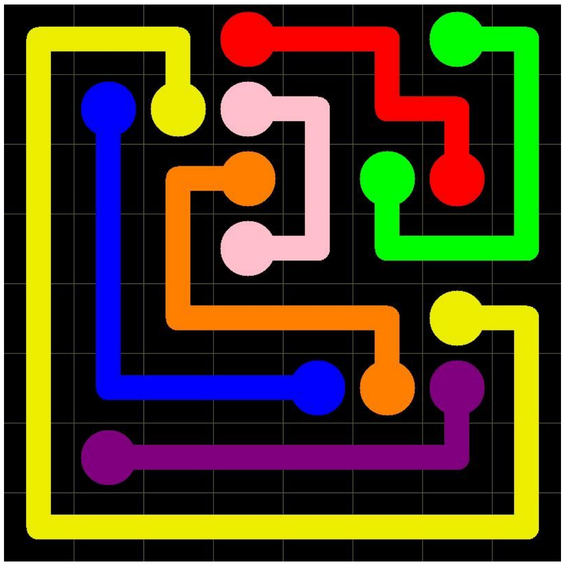
7x7

Variables: (0,0), (0,1), (0,2), (0,3), (0,4), (0,5), (1,0), (1,2), (1,3), (1,4), (1,5), (1,6), (2,0), (2,1), (2,2), (2,3), (2,4), (2,6), (3,1), (3,4), (3,5), (3,6), (4,0), (4,3), (4,4), (4,5), (5,0), (5,2), (5,3), (5,4), (5,5), (5,6), (6,0), (6,1), (6,2), (6,3), (6,4), (6,5), (6,6)

Domain: O (orange), B (blue), G (green), R (red), Y (yellow)

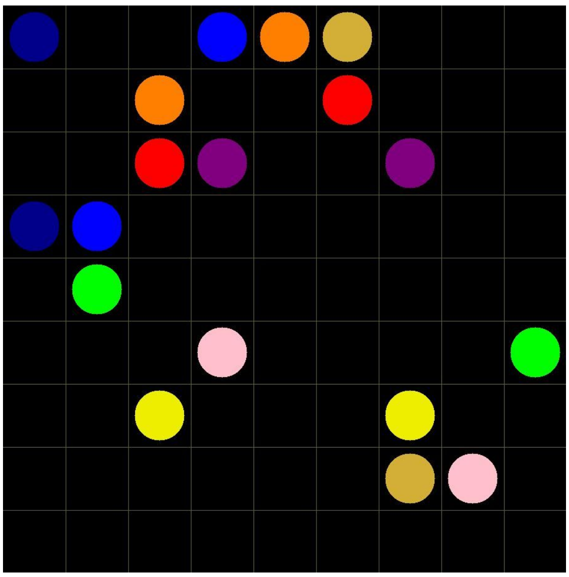
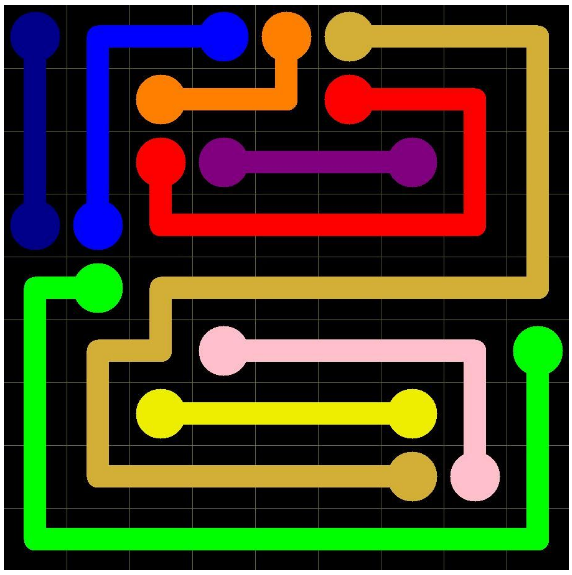
8x8

Variables: (0,0), (0,1), (0,2), (0,3), (0,4), (0,5), (0,6), (0,7), (1,0), (1,2), (1,3), (1,4), (1,5), (1,7), (2,0), (2,1), (2,2), (2,3), (2,4), (2,6), (2,7), (3,4), (3,5), (3,6), (3,7), (4,0), (4,1), (4,2), (4,3), (4,4), (4,6), (4,7), (5,0), (5,1), (5,3), (5,4), (5,6), (5,7), (6,1), (6,3), (6,6), (6,7), (7,0), (7,1), (7,2), (7,3), (7,4), (7,5), (7,6), (7,7)

Domain: O (orange), B (blue), G (green), R (red), Y (yellow), P (pink), Q (purple)

9x9

Variables: (0,1), (0,2), (0,4), (0,5), (0,6), (0,7), (0,8), (1,0), (1,1), (1,2), (1,5), (1,6), (1,7), (1,8), (2,0), (2,3), (2,4), (2,5), (2,7), (2,8), (3,1), (3,3), (3,4), (3,6), (3,7), (3,8), (4,1), (4,2), (4,3), (4,4), (4,5), (4,6), (4,7), (4,8), (5,2), (5,3), (5,4), (5,5,)(5,6), (5,7), (5,8), (6,0), (6,1), (6,3), (6,4), (6,5), (6,8), (7,0), (7,1), (7,2), (7,3), (7,4), (7,5), (7,6), (7,8), (8,0), (8,1), (8,2), (8,3), (8,4), (8,6), (8,7), (8,8)

Domain: O (orange), B (blue), G (green), R (red), Y (yellow), P (pink), Q (purple), D (dark blue), K (gold)

"Smart" Implementation:

For our "smart" search algorithm, we chose to implement a combination of MRV, MCV, forward checking, and some additional heuristics specific to the flow free problem (most notably, we chose to favor values that would assign a variable a color matching one of its neighbors) in order to improve the efficiency of our algorithm. The addition of the color-matching-neighbor heuristic brought our number of attempted assignments down from 19 to 15 for the 5x5 puzzle which is the minimum number possible. Of all the heuristics we implemented though, the combination of MRV and forward checking seemed to make the greatest difference in terms of speed and number of cells assigned.

As can be seen from the chart below, we likely have a few remaining bugs in our code. In the very least, there are still ways we could improve our algorithm to reduce the number of assignments and execution time for puzzles larger than 5x5.

		5x5	7x7	8x8	9x9
Smart	# of Assignments	15	1673	3126	6475
	Execution Time	~0.0988 sec	~80 sec	~370 sec	~1099 sec
*Dumb	# of Assignments	91	???	???	???
	Execution Time	~0.0737 sec	>10 min	>10min	>10min

* Note, we cut off our "dumb" implementation for all three of the larger puzzles.

Files Related To Part 1:

dumb.py - contains our "dumb" implementation
smart.py - contains our "smart" implementation
beautify.py - contains the code we wrote to generate the graphics for each grid

Report 2.1:**Minimax and alpha-beta agent**

Minimax():

Inputs:current state, player1, player2, depth of game tree , evaluation function

Outputs:returns the best move that maximises utility

Purpose: It returns the best possible move that leads to the outcome with the best utility (calculated by evaluation functions), assuming that the opponent plays to minimize utility.

Alphabeta():

Inputs:current state, player1, player2, depth of game tree , evaluation function

Outputs:returns the best move that maximises utility

Purpose: computes the correct minimax decision without looking at every node in the game tree using pruning and thus prunes away branches that cannot possibly influence the final decision.

Ordering of moves to improve Pruning: In order effectiveness of alpha–beta pruning we wrote a simple ordering function **def move_ordering(moves,mat,player1,player 2)** which takes the state of board and available moves for a player as parameter, and arranges the moves according to captures first, then threats, then forward moves.

Defensive Heuristic 2:Number of workers of the player in the state+ 3*(empty_squares_2blocks ahead_in all possible forward movement directions)+2*(empty_squares_1blocks ahead_in all possible forward movement directions)-3*no. of workers surrounded by opponents workers. If worker of player is not surrounded by opponent players then it should be given higher preference to move, this decreases the chance of the worker being attacked. Therefore we should choose states where more number of workers are free to move without being attacked by opponent atleast for 2 moves. In addition, We should choose a state so as to minimise the no. of workers surrounded by opponents to maintain the player's workers and thus ensuring strong defence.

Offensive Heuristic 2:In a given state of board we count workers of a player who aren't surrounded by their opponents at least 2 blocks up and 2 blocks on either diagonal and added to the score of offensive Heuristic 1. This way these workers are more free to move forward and attack opponents. Moreover they are also protected from being captured by opponent workers.

Offensive Heuristic2=Offensive Heuristic1+ 3*(empty_squares_2blocks ahead_in all possible forward movement directions)+2*(empty_squares_1blocks ahead_in all possible forward movement directions)

Noise of evaluation function on final outcome: On running each of the below matchup at least 5 times I observed that there was a difference in outcomes. Noise of evaluation function created by random does affect the final outcome

Minimax (Offensive Heuristic 1) (Player1) vs Alpha-beta (Offensive Heuristic 1)(Player2)

The final state of the board (who owns each square) and the winning player:

```
player1:1 and player2:-1
Final state of the board:
[[ 1.  0. -1.  0.  0.  1.  1.  1.]
 [ 0.  0.  0.  0.  0.  0.  1.  1.]
 [ 0.  0.  0.  0.  0.  1.  1.  0.]
 [ 0.  0.  0.  0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0. -1. -1. -1. -1. -1.]
 [-1. -1. -1. -1. -1. -1. -1. -1.]]

Winner: player2
The number of opponent workers captured by  player1: 2
The number of opponent workers captured by  player2: 7
Total number of moves required till the win: 30
Total number of game tree nodes expanded by  player1: 172365
Total number of game tree nodes expanded by  player2: 197957
Average number of nodes expanded per move: 12344
Average amount of time to make a move: 0.5543892 seconds
```

Implemented the above match 50 times and Alphabeta search Agent won 93% games.

Alpha-beta (Defensive Heuristic 2)(Player2) vs Alpha-beta (Offensive Heuristic 1)(Player1)

```
player1:1 and player2:-1
Final state of the board:
[[ 0. -1.  0.  0.  0.  0.  0.  0.]
 [ 0.  1.  0.  1.  0.  1.  0.  1.]
 [ 0.  0.  0.  0.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  0.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]
 [-1. -1. -1. -1. -1. -1. -1. -1.]]

Winner: player2
The number of opponent workers captured by  player1: 7
The number of opponent workers captured by  player2: 3
Total number of moves required till the win: 54
Total number of game tree nodes expanded by  player1: 387567
Total number of game tree nodes expanded by  player2: 371240
Average number of nodes expanded per move: 14051
Average amount of time to make a move: 1.39560422222 seconds
```

Implemented the above match 15 times and Alphabeta search(Defensive Heuristic 2) Agent won about 76% games. It takes more time per move compared to the offensive Huerestics

Alpha-beta (Offensive Heuristic 2)(Player1) vs Alpha-beta (Defensive Heuristic 1)(Player2)

```
player1:1 and player2:-1
Final state of the board:
[[ 0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  1.  1.]
 [ 0.  0.  1.  1.  1.  1.  1.  1.]
 [ 1.  0.  0.  0.  0.  1.  0.  0.]
 [ 0.  1.  1.  0.  1.  0.  0.  0.]
 [ 0.  0.  0.  0.  1.  0.  0.  0.]
 [ 0.  1.  1.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]]

Winner: player1
The number of opponent workers captured by  player1: 16
The number of opponent workers captured by  player2: 0
Total number of moves required till the win: 81
Total number of game tree nodes expanded by  player1: 464533
Total number of game tree nodes expanded by  player2: 375282
Average number of nodes expanded per move: 10368
Average amount of time to make a move: 0.814747419753 seconds
```

Implemented the above match 50 times and Alphabeta search(Offensive Heuristic 2) Agent won 87.3% games.
Also the number of opponent workers captured by this player were always higher than the defensive agent

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

```
player1:1 and player2:-1
Final state of the board:
[[ 0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  1.  1.  1.]
 [ 0.  0.  0.  0.  0.  0.  1.  1.]
 [ 1.  1.  1.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  1.  1.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]]

Winner: player1
The number of opponent workers captured by  player1: 16
The number of opponent workers captured by  player2: 6
Total number of moves required till the win: 93
Total number of game tree nodes expanded by  player1: 452683
Total number of game tree nodes expanded by  player2: 402925
Average number of nodes expanded per move: 9200
Average amount of time to make a move: 0.885310311828 seconds
```

Alpha-beta (Offensive Heuristic 2)(Player 1) vs Alpha-beta (Offensive Heuristic 1)(Player2)

```
player1:1 and player2:-1
Final state of the board:
[[ 0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  1.  1.]
 [ 0.  0.  0.  0.  1.  0.  1.  1.]
 [ 1.  1.  1.  0.  1.  1.  0.  0.]
 [ 1.  0.  0.  0.  1.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]]

Winner: player1
The number of opponent workers captured by  player1: 16
The number of opponent workers captured by  player2: 4
Total number of moves required till the win: 95
Total number of game tree nodes expanded by  player1: 486634
Total number of game tree nodes expanded by  player2: 415423
Average number of nodes expanded per move: 9495
Average amount of time to make a move: 0.789284084211 seconds
```

Implemented the above match 20 times and Alphabeta search(Offensive Heuristic 2) Agent won about 85% games.

Also in the case where it played against defensive Heuristic 1 it always captured more workers compared to its opponents. However in this case there were cases when offensive heuristic two captured more opponents than offensive Heuristic 1 , though theses cases were very few.

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

```
player1:1 and player2:-1
Final state of the board:
[[ 0.  0. -1.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.  0.  1.  1.]
 [ 1.  1.  1.  1.  0.  1.  0.  1.]
 [ 0.  0.  0.  0.  1.  1.  0.  0.]
 [ 0.  0.  1.  0.  0.  0.  0. -1.]
 [ 0. -1.  0.  0.  0. -1.  0.  0.]
 [ 0.  0.  0. -1.  0.  0. -1. -1.]]

Winner: player2
The number of opponent workers captured by  player1: 9
The number of opponent workers captured by  player2: 4
Total number of moves required till the win: 78
Total number of game tree nodes expanded by  player1: 491898
Total number of game tree nodes expanded by  player2: 435252
Average number of nodes expanded per move: 11886
Average amount of time to make a move: 1.28354884615 seconds
```

Alpha Beta defensive bet Alpha beta offensive though offensive capture more opponents compared to defensive agent.

2.2(Bonus)
To implement this function i only changed the condition to check if the game is over or not to GameOver: (a) move 3 pieces to the enemy's home base; (b) capture n-2 of the enemy's pieces, where n is the total number of players the enemy has at the beginning of the game.

Alpha-beta (Offensive Heuristic 2)(Player1) vs Alpha-beta (Defensive Heuristic 1)(Player2)

```
player1:1 and player2:-1
Final state of the board:
[[ 0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  1.  1.  1.]
 [ 0.  1.  1.  1.  1.  1.  1.  1.]
 [ 0.  0.  0.  1.  1.  0.  0.  0.]
 [ 0.  1.  1.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  1.  1.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0. -1. -1.]]

Winner: player1
The number of opponent workers captured by  player1: 14
The number of opponent workers captured by  player2: 0
Total number of moves required till the win: 69
Total number of game tree nodes expanded by  player1: 456512
Total number of game tree nodes expanded by  player2: 370361
Average number of nodes expanded per move: 11983
Average amount of time to make a move: 0.943181217391 seconds
```

Alpha-beta (defensive Heuristic 2)(Player1) vs Alpha-beta (Offensive Heuristic 1)(Player2)

```
player1:1 and player2:-1
Final state of the board:
[[ 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.  1.  1.]
 [ 0.  1.  1.  1.  0.  1.  1.  1.]
 [ 0.  1.  0.  0.  0.  1.  0.  0.]
 [ 0.  1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]]

Winner: player1
The number of opponent workers captured by  player1: 14
The number of opponent workers captured by  player2: 5
Total number of moves required till the win: 103
Total number of game tree nodes expanded by  player1: 502439
Total number of game tree nodes expanded by  player2: 427588
Average number of nodes expanded per move: 9029
Average amount of time to make a move: 0.845540466019 seconds
```

For this implementation we changed the defensive 2 heuristic such that

4*(empty_squares_2blocks ahead_in all possible forward movement directions)+2*(empty_squares_1blocks ahead_in all possible forward movement directions)-4*no. of workers surrounded by opponents workers.
We gave equal weightage to minimising states having greater no. of workers surrounded by opponents, and maximising states allowing free movement for at least 2 blocks in any direction of forward motion

Workload Distribution
Anna =Part1
Shruti-part2