



Artificial Intelligence

Laboratory activity

Name: Enoiu Diana Cristina

Group: 30432

Email: edianagte@yahoo.com

Teaching Assistant: Adrian Groza
Adrian.Groza@cs.utcluj.ro



Contents

1	A1: Search - Other algorithms	3
1.0.1	Algorithms.	3
1.0.2	Comparisons.	9
2	A2: Logics	10
2.0.1	Sudoku Generator	10
2.0.2	Complete board generation	10
2.0.3	Shuffling	12
2.0.4	Removing numbers one by one	12
2.0.5	Final board	14
3	A3: Planning	16
3.0.1	Boxup Puzzle Game	16
3.0.2	Boxup Level 1	18
3.0.3	Boxup Level 2	19
3.0.4	Boxup Level 3	19
A	Your original code	21
A.0.1	Assignment 1	21
A.0.2	Assignment 2	24
A.0.3	Assignment 3	31

Chapter 1

A1: Search - Other algorithms

Introduction. In the previous laboratories we tried to make Pacman find the shortest-path to the food dot by implementing various algorithms. Some algorithms were uninformed search such as DFS, BFS, UCS and some others used more information in regards to the problem, that is heuristics, like A* algorithm.


The objective of this assignment is to find new algorithms besides DFS,BFS,UCS,A* so that Pacman successfully finds the food dot and we will also try to analyze how this algorithms behave on different types of layouts. Some algorithms do not work on every maze.

In the end, we will place all the results in a table, to compare the algorithms between them.

1.0.1 Algorithms.

1. *Mouse search*

One of the first algorithms that we implemented in the laboratories hours was random search. Mouse search attempts to be a better version of random search because instead of choosing a random move every time Pacman moves, it will choose a random move every time it will reach a junction, otherwise it will keep it's direction if possible. Another improvement is removing the option to go back, since there will be no reason to turn back since it is on a path. This will remove the unnecessary going back and forth that happens in random search.

Overall, mouse search is better than random search, but it is still not a good algorithm since it still takes a lot of time to find the food dot and is almost impossible to find the shortest path, unless we are very, very lucky .

For this algorithm I have used the following methods:

- (a) This method is used to remove the successor that sends the Pacman backwards from the list of successors.

```
def getSuccessorsWithoutReverse(successors, curr_dir):
    rev_dir = Actions.reverseDirection(curr_dir)
    reverseSuccessor = None
    for successor in successors:
        nextAction = successor[1]
        if rev_dir == nextAction: # i am trying to not go in the
opposite direction
            reverseSuccessor = successor
    if reverseSuccessor is not None:
        successors.remove(reverseSuccessor)
    return successors
```

- (b) This is the method with the main algorithm. Basically, if we are at a junction we choose a random move, if we are on a passageway we go forward, else we can only go back. We always maintain the current state of the Pacman in a variable named "curr_dir", so we can keep going in a specific direction.

```
def mouseSearch(problem):
    from game import Directions
    current_state = problem.getStartState()
    solution = []
    curr_dir = Directions.EAST
    while not (problem.isGoalState(current_state)):
        successors = problem.getSuccessors(current_state)
        if len(successors) >= 2: # i am at a junction
            successorsWithoutReverse = getSuccessorsWithoutReverse(
                successors, curr_dir) # i dont want to go back
            nextState = choice(successorsWithoutReverse) # i will
            choose a random move
        elif len(successors) == 2: # i am at a passageway
            keepDirectionSuccessor = getSuccessorsWithoutReverse(
                successors, curr_dir)
            nextState = keepDirectionSuccessor[0] # since reverse is
            removed, there is only one option left
        else: # dead end
            nextState = successors[0] # the only option left is going
            back
        current_state = nextState[0]
        curr_dir = nextState[1]
        solution.append(curr_dir)
    return solution
```

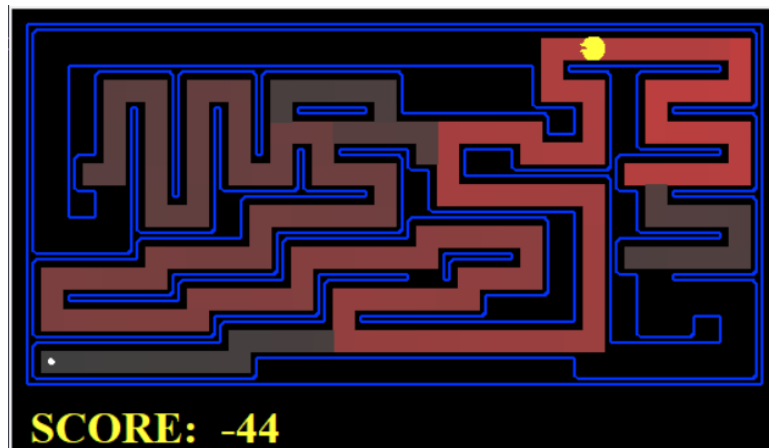


Figure 1.1: How mouse search finds a path on medium maze

2. Wall flower search

Imagine that Pacman is blind and the only way he could possibly find the food dot is by using his hands. Because he cannot see forward, he will simply place his hand on the wall and it will just go along the wall, without ever taking his hand of the wall, hoping that he eventually finds the food dot. This method will work only if the maze is simply connected, that is, all the walls are connected together. Also, the food dot should be placed near the outer walls (like an exit).

The efficiency of this algorithm is not great, but it is better than the mouse search.

To apply this algorithm we must have a desired action, that is left if the left hand is placed on wall. If the wall turns left, then Pacman must turn left with it, if the wall goes right, Pacman will go right as well. In case of dead end, we have to imagine that as Pacman is keeping the hand on the wall he will find his way out by simply rotating with the wall.

For this algorithm I have used the following methods:

- (a) This method is used to obtain the desired action that Pacman has to make, depending on his current direction in the maze and which hand is keeping on the wall.

```
def desiredAction(action, hand):
    if hand == "left":
        if action == Directions.WEST:
            return Directions.SOUTH
        elif action == Directions.SOUTH:
            return Directions.EAST
        elif action == Directions.EAST:
            return Directions.NORTH
        else:
            return Directions.WEST
    else:
        if action == Directions.WEST:
            return Directions.NORTH
        elif action == Directions.NORTH:
            return Directions.EAST
        elif action == Directions.EAST:
            return Directions.SOUTH
        else:
            return Directions.WEST
```

- (b) This function is used to return the successor that is accordingly to the desired action, if there is one.

```
def getDesiredSuccessor(successors, curr_dir, hand):
    desired_dir = desiredAction(curr_dir, hand)
    desiredSuccessor = None
    for successor in successors:
        nextAction = successor[1]
        if desired_dir == nextAction: # i am looking for the successor
            with the desired action
            desiredSuccessor = successor
    return desiredSuccessor
```

- (c) This function will try to return the successor that keeps my direction, since we cannot perform the desired action. There are 2 cases: the wall goes straight, then we go straight or the wall turns in the opposite direction than the one we want, so we do the same.

```
def getKeepDirectionSuccessor(successors, curr_dir): # assumes
    there are more than or 2 successors,
    # and the desired action is not possible
    keepDirSuccessor = None
    reverseSuccessor = None
    rev_dir = Actions.reverseDirection(curr_dir)
    for successor in successors:
        nextAction = successor[1]
        if curr_dir == nextAction: # successor that keeps my direction
            keepDirSuccessor = successor
```

```

        if rev_dir == nextAction: # i am trying to not go in the
opposite direction
            reverseSuccessor = successor
        if reverseSuccessor is not None: # since there is more than one
successor i dont need this
            successors.remove(reverseSuccessor)
        if keepDirSuccessor is not None:
            return keepDirSuccessor
        return successors[0] # in case there is no keep dir successor
there is only one left successor

```

- (d) The final method that implements the wall flower algorithm. We can choose which hand we want to use. The algorithm can be improved, if the pacman reaches the starting point, it can try to switch hands so that he can travel on the rest of unexplored maze in search of the food dot (if the food dot is not placed near an outer wall, because if it is it will be found with certainty at the first travel with left hand).

```

def wallFlowerSearch(problem):
    current_state = problem.getStartState()
    solution = []
    curr_dir = Directions.NORTH
    hand = "left"
    while not (problem.isGoalState(current_state)):
        successors = problem.getSuccessors(current_state)
        if len(successors) >= 2: # keep going forward
            desiredSuccessor = getDesiredSuccessor(successors, curr_dir
, hand)
            if desiredSuccessor is not None:
                nextState = desiredSuccessor
            else:
                nextState = getKeepDirectionSuccessor(successors,
curr_dir)
        else: # dead end
            nextState = successors[0]
            current_state = nextState[0]
            curr_dir = nextState[1]
            solution.append(curr_dir)
    return solution

```

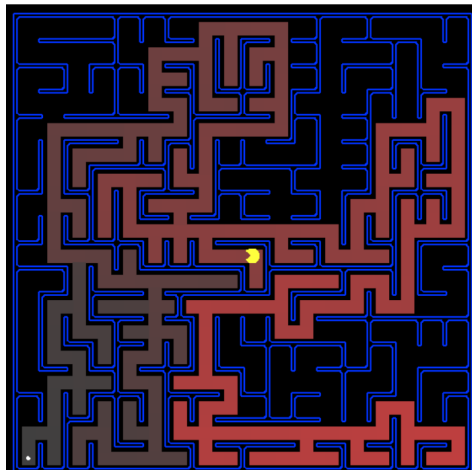


Figure 1.2: How wall flower search finds a path on big maze

From the predefined layouts, only big maze satisfies the requirements for this algorithm to work. I have also made a different layout to see better what this algorithm does.

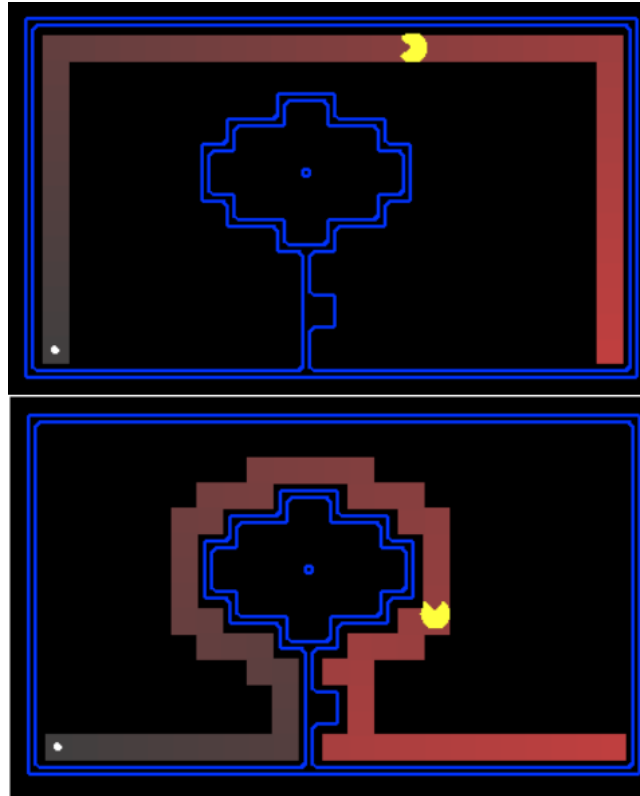


Figure 1.3: How wall flower search finds the target on a custom layout depending on which hand stays on the wall (and also an attempt to make a flower)

3. Bidirectional search

A better algorithm and a more serious one would be the bidirectional search. The idea of the bidirectional search is to start searching from the start to the goal but also from the goal to the start in hope that the two searches will meet somewhere in the middle. As AIMA notes, *the motivation is that $b^{d/2} + b^{d/2}$ is much less than b^d , or as in the figure, the area of the two small circles is less than the area of one big circle centered on the start and reaching to the goal.*

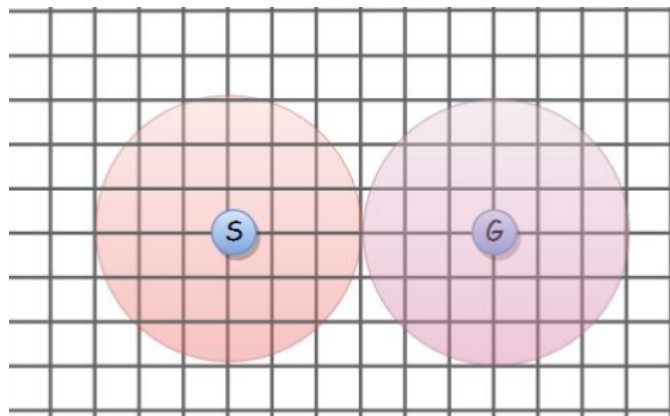


Figure 1.4: Bidirectional search visualization

I decided to implement this algorithm to work also with two greedy searches, not just BFS. The greedy search is basically the same algorithm as A* but instead of taking into

consideration the cost from start to goal, it only uses the heuristic when choosing the next state from priority queue. Thus $f(n) = h(n)$ and not $f(n) = g(n) + h(n)$ as in case of A*.

```
def greedySearch(problem, heuristic=nullHeuristic):
    frontier = util.PriorityQueue()
    exploredStates = []
    initial_state = problem.getStartState()
    start_heuristic = heuristic(initial_state, problem)
    if problem.isGoalState(initial_state):
        return []
    frontier.push((initial_state, [], 0), start_heuristic)
    while not frontier.isEmpty():
        currentState, currentActions, currentCost = frontier.pop()
        if currentState not in exploredStates:
            exploredStates.append(currentState)
            if problem.isGoalState(currentState):
                return currentActions
            for nextState, nextAction, nextCost in problem.getSuccessors(
                currentState):
                newActions = currentActions + [nextAction]
                getHeuristics = heuristic(nextState, problem)
                frontier.push((nextState, newActions, nextCost),
                    getHeuristics)
```

However, if I do not specify a heuristic for the search, it will do just a BFS search instead of Greedy. The way this algorithm works is that it has two priority queues for both searches, two lists to visualize the nodes in the queues and two sets for the explored nodes. I will perform the searches from both start and goal states, simultaneously. If one node is in the other queue or, vice-versa, it means that the two searches found a common path and the algorithm is over. Otherwise, one of the searches will eventually reach the end node that it was looking for, and a common path will not be found. Once the path is found, the job is not done, since we still need to reverse the path that comes from the goal to the meeting point and we should also reverse the actions(directions) of the states in that path, as in the following method:

```
def getOppositeDirections(directions):
    oppositeDirections = []
    for direction in directions:
        if direction == Directions.NORTH:
            oppositeDirections.append(Directions.SOUTH)
        elif direction == Directions.SOUTH:
            oppositeDirections.append(Directions.NORTH)
        elif direction == Directions.WEST:
            oppositeDirections.append(Directions.EAST)
        else:
            oppositeDirections.append(Directions.WEST)
    return oppositeDirections
```

The full code of bidirectional search can be found in Appendix, here is a pseudo-code that gives us a better idea what bidirectional algorithm does.

When to use bidirectional approach?

We can consider bidirectional approach when:

```

1 startq = Queue for BFS from start node;
2 endq = Queue for BFS from end node;
3 visited1= List with visited nodes for BFS from start node;
4 visited2= List with visited nodes for BFS from goal node;
5 while startq is not empty and endq is not empty do
6     perform next iteration of BFS for startq (also send node to visited1 list);
7     if we have encountered the intersection node then
8         compute the full path and return;
9     else
10        continue the searching algorithm;
11    perform next iteration of BFS for endq (also send node to visited2 list);
12    if we have encountered the intersection node then
13        compute the full path and return;
14    else
15        continue the searching algorithm;


```

- (a) Both initial and goal states are unique and completely defined.
- (b) The branching factor is the same in both directions.

Performance measures

- (a) Completeness: Bidirectional search is complete if BFS is used in both searches.
- (b) Optimality: It is optimal if BFS is used for search and paths have uniform cost.
- (c) Time and Space Complexity: Time and space complexity is $O(b^{d/2})$.

1.0.2 Comparisons.

Results	Mouse	Wall Flower	Bidirect.(BFS)	Bidirect.(Greedy)	BFS	A*	Greedy
Score	<-1000	-88	300	300	300	300	300
Cost	1628 	598	210	210	210	210	210
E.N.	1628	598	596	532	620	549	466

This results were performed on Big Maze, using the Manhattan heuristic, where it is the case. We can conclude that, in this situation, Greedy performed the best with only 466 nodes expanded. We managed to obtain a better result than A* with the bidirectional search using two Greedy searches. The bidirectional with BFS also beat the BFS alone, at a small difference. The wall flower algorithm manages to have less expanded nodes than BFS, but the path that is found is far from optimal (as expected since Pacman is blind).

Chapter 2

A2: Logics

2.0.1 Sudoku Generator

Introduction. For the Logics assignment we have decided to implement a Sudoku board generator that uses Mace4 Model Searcher. The main idea is to generate with Mace4 a fully completed Sudoku board, that will be further shuffled and then we will erase numbers one by one, in order to generate a board that is valid for a Sudoku game. After each deletion of a number we will check if the new board still has a unique solution. This checking will be performed with a Sudoku solver that will be run with Mace4 as well.

The main algorithm to implement this Sudoku generator is described here:

```
1 Generate a complete, valid board;
2 Shuffle rows and columns randomly to make it look like a different board;
3 ListSudokuCells = unordered, shuffled list with the 81 position cells;
4 while ListSudokuCells not empty do
5   take next position from list, and remove that number from the board;
6   if we still have a valid sudoku board with one solution then
7     continue with next number from list;
8   else
9     undo the removal and continue with the next number from list;
```

2.0.2 Complete board generation

We will use Mace4 to generate a complete solved Sudoku board. To do that, we should specify the Sudoku rules, but we will give no hints regarding where the numbers are placed, thus starting from an empty board. After we run `sudoku_generator.in` with Mace4, we will obtain:

f	0	1	2	3	4	5	6	7	8	same_interval	0	1	2	3	4	5	6	7	8
0	0	1	2	3	4	5	6	7	8	0	1	1	1	0	0	0	0	0	0
1	3	4	5	6	7	8	0	1	2	1	1	1	1	0	0	0	0	0	0
2	6	7	8	0	1	2	3	4	5	2	1	1	1	0	0	0	0	0	0
3	1	2	0	5	6	3	7	8	4	3	0	0	0	1	1	1	0	0	0
4	7	6	4	8	0	1	2	5	3	4	0	0	0	1	1	1	0	0	0
5	5	8	3	4	2	7	1	0	6	5	0	0	0	1	1	1	0	0	0
6	2	0	6	1	5	4	8	3	7	6	0	0	0	0	0	0	1	1	1
7	4	3	1	7	8	6	5	2	0	7	0	0	0	0	0	0	1	1	1
8	8	5	7	2	3	0	4	6	1	8	0	0	0	0	0	0	1	1	1

The 'f' function represents the Sudoku board. Since Mace4 arithmetic domain starts from 0, but Sudoku has numbers form 1 to 9, we will replace the 0's with 9, when generating the final board.

In `sudoku_generator.in` file we have the following rules:

1. All numbers in a row are different.

```
all x all y1 all y2 (f(x, y1) = f(x, y2) -> y1 = y2).
```

2. All numbers in a column are different.

```
all x1 all x2 all y (f(x1, y) = f(x2, y) -> x1 = x2).
```

3. All numbers in a square are different.

For this we had to define a relationship called `same_interval`.

```
all x same_interval(x,x).
all x all y (same_interval(x,y) -> same_interval(y,x)).
all x all y all z (same_interval(x,y) & same_interval(y,z) ->
same_interval(x,z)).

same_interval(0,1).
same_interval(1,2).
same_interval(3,4).
same_interval(4,5).
same_interval(6,7).
same_interval(7,8).
-same_interval(0,3).
-same_interval(3,6).
-same_interval(0,6).
```

And then specify the rule.

```
all x1 all y1 all x2 all y2
(
    same_interval(x1,x2) &
    same_interval(y1,y2) &
    f(x1, y1) = f(x2, y2)
->
    x1 = x2 &
    y1 = y2
).
```

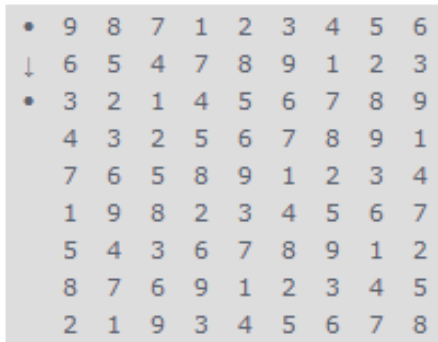
We have to generate just one model using the command `mace4 -m 1 -f sudoku_generator.in`. The output of Mace4 will be send to a file called `mace4_complete_sudoku.out`, that will be further use by `generator.py` to interpret the result and generate the final sudoku board. The `generator.py` file will be responsible for calling Mace4 to generate models, then parsing the files that Mace4 outputs, and then send a valid Sudoku board for a Sudoku game implemented in `sudoku.py`.

To add more randomness when creating the complete Sudoku board, we can also give to Mace4 just a single random number in a random place. This will force Mace4 to generate different completed boards, and not just the one we have shown previously.

2.0.3 Shuffling


Since Mace4 will always output the same board when generating a complete board, starting from an empty one, we have to shuffle between rows and columns to give the impression of a randomly created board. This operation will be handled by `generator.py`.

In order to perform the shuffling we have to ensure that the newly created Sudoku board still complies to the Sudoku rules. Thus, we are only allowed to shuffle between rows and columns from the same interval i.e. $\{1, 2, 3\}$ or $\{4, 5, 6\}$ or $\{7, 8, 9\}$.



•	9	8	7	1	2	3	4	5	6
↓	6	5	4	7	8	9	1	2	3
•	3	2	1	4	5	6	7	8	9
	4	3	2	5	6	7	8	9	1
	7	6	5	8	9	1	2	3	4
	1	9	8	2	3	4	5	6	7
	5	4	3	6	7	8	9	1	2
	8	7	6	9	1	2	3	4	5
	2	1	9	3	4	5	6	7	8

(a) Shuffle rows



•	↔	•							
3	2	1	4	5	6	7	8	9	
6	5	4	7	8	9	1	2	3	
9	8	7	1	2	3	4	5	6	
4	3	2	5	6	7	8	9	1	
7	6	5	8	9	1	2	3	4	
1	9	8	2	3	4	5	6	7	
5	4	3	6	7	8	9	1	2	
8	7	6	9	1	2	3	4	5	
2	1	9	3	4	5	6	7	8	

(b) Shuffle columns

To achieve a pseudo-random solution every time we generate a new puzzle, we randomly shuffle groups of rows then transpose the board. We repeat this process a random number of times, and the output result is a solvable Sudoku board that is always different from the previous one.

```
def __shuffle(self, old_board):
    board = numpy.array(old_board)
    group = 3 # we can only safely shuffle between group of 3 consecutive rows
              /columns
    number_of_shuffles = random.randrange(1, 10, 1)
    for i in range(number_of_shuffles):
        numpy.random.shuffle(board.reshape(board.shape[0] // group, -1, board.
shape[1])) # shuffles rows
        board = numpy.transpose(board) # transpose the board so that we can
shuffle the columns
    return board.tolist()
```

Listing 2.1: board shuffler

2.0.4 Removing numbers one by one

The chosen algorithm starts with a completed board, then gradually removes numbers until it finishes. We used two variations of this algorithm. Either of them works and provides a functioning solution, and each of them have their advantages and disadvantages.

The first method used (`simple_solve`) is one that makes a number of attempts to pick a random number from the grid, remove it, check if the board still has a unique solution, then put the number back if the board is not solvable. The advantages of this approach is that it does not have a fixed number of attempts and we can control how many numbers (approximately)

we want to remove from our board. This is a way to control the level of difficulty of the board (the more numbers are removed, the more complex the puzzle will be).

```
def __simple_solve(self):
    # generation algorithm
    to_remove = 81 # number of attempts to remove numbers from the board

    while to_remove > 0:
        # select a random row and a column
        row = random.randint(0, 8)
        col = random.randint(0, 8)

        # if that position has already been removed, search again
        while self.sudoku_board[row][col] == 0:
            row = random.randint(0, 8)
            col = random.randint(0, 8)

        # memorize the value to be removed, then remove it
        old_value = self.sudoku_board[row][col]
        self.sudoku_board[row][col] = 0

        self.__generate_sudoku_solver_file() # generate mace4 file for the
current board
        self.__call_mace4_to_verify_sudoku() # solve the current board
        if not self.__solution_is_unique():
            # the board does not have a unique solution, put the number back
and try again
            self.sudoku_board[row][col] = old_value
            to_remove -= 1
```

Listing 2.2: Simple Solve Method

The second method used is slightly more time efficient, because it does not attempt to find a random position at every step. Instead, it starts with a list of all 81 positions on a Sudoku board, shuffles them, and then traverses the list. At every step of the algorithm, it attempts to remove the number situated at the current position from the list. If, after removing the number, the Sudoku board no longer has a unique solution, it puts the number back on the board and then moves on with the next position. This algorithm stops when it reaches the end of the list. Here, too, we can select the difficulty of the level by also counting how many values we have removed from the board and stopping at a given threshold.

```
def complex_solve(self):
    positions = numpy.arange(0, 81, 1) # list of positions
    numpy.random.shuffle(positions) # shuffle the positions
    board_clone = self.__save_a_copy_board() # duplicate the original board

    values_removed = 0
    current_index = 0

    while values_removed < 64 and current_index < 81:
        row = positions[current_index] // 9
        col = positions[current_index] % 9

        # remove number from current index
        self.sudoku_board[row][col] = 0
```

```

        self.__generate_sudoku_solver_file() # generate mace4 file for the
current board
        self.__call_mace4_to_verify_sudoku() # solve the current board

    if not self.__solution_is_unique():
        # put the number back
        self.sudoku_board[row][col] = board_clone[row][col]
    else:
        # it's still solvable, move on
        values_removed += 1

    current_index += 1

```

Listing 2.3: Complex Solve Method

Both algorithms work in a very similar way. After removing a value, they call `generate_sudoku_solver_file()`, which edits the `sudoku_solver.in` file (see appendix) to add the current values present in the Sudoku board. Then, Mace4 is called to verify if the Sudoku puzzle with the given constraints has a unique solution using `call_mace4_to_verify_sudoku()`.

2.0.5 Final board

Our goal is to achieve a solvable Sudoku board that contains the fewest numbers. It is not always possible to reach the absolute minimum amount of numbers (17), but we can get close enough and this produces high difficulty Sudoku puzzles.

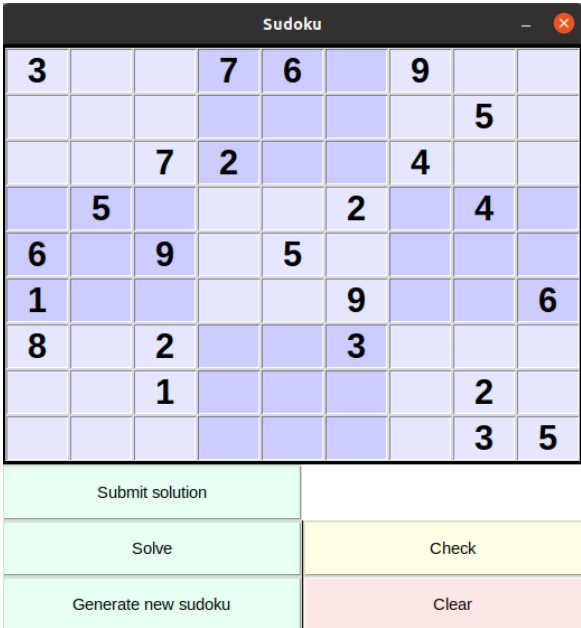
By repeatedly running the algorithm, it results that we typically find solvable Sudoku boards with 20-30 filled in numbers, which produces a challenging Sudoku game.

Below are some sample Sudoku games generated with our algorithm:

	5					8	3	
		6		3				
2					7			
				1		6		
					5			
	1	2	6	7		3		
5	6		1	2		7		
		1	7		4	2		3
				8			9	

	3			7	5			
			2					
2		8	9					
	6		5		4			
	7				2	5	8	
		2				4		
		3						
7	1						2	6
				3				1

In the end, we also had some fun creating a real Sudoku game, using python and Tkinter to make the GUI. The game is able to generate sudoku boards by using the GenerateSudokuBoard class found in `generator.py`. It is also able to detect when we have added a duplicated number, or to check if the introduced solution is correct.



Chapter 3

A3: Planning

3.0.1 Boxup Puzzle Game

Introduction. For the Planning assignment I have decided to implement a way in which I can solve some puzzles in the Boxup Puzzle Game found at: [Boxup Puzzle Game Link](#). The main idea of the game is to drag the red block inside the blue block, using the small black block and also any additional blocks that are black and can move.

To solve this game we need the following predicates in our domain:

```
(:predicates
  (is-tile ?t)
  (is-block ?b)
  (is-miss-left ?l)
  (is-miss-right ?l)
  (is-miss-up ?l)
  (is-miss-down ?l)
  (tile-at ?t ?r ?c)
  (is-blank ?r ?c)
  (next-row ?r1 ?r2)
  (next-column ?c1 ?c2)
  (block-at ?b ?r ?c)
  (block-missing ?b ?x)
  (is-small-block ?b)
  (is-big-block ?b)
)
```

The `is-miss-left`, `is-miss-right`, `is-miss-up` and `is-miss-down` are to distinguish the missing parts. The `is-blank` is to identify if a position in the grid has a block on it or not (except from the small black block that we can move). The `block-missing` is to identify if a block has the missing part left, right, up or down.

As for actions, we are going to extend the moves that we used for `sliding-tiles` example from the laboratory. The basic move is moving from a blank position to another, but we also have other moves like moving inside a block, outside a block or from block to block. When moving inside blocks, we have to consider the missing piece so we don't go through walls. Here are the actions down movement:

```
(:action move-tile-down
  :parameters (?tile ?old-row ?new-row ?col ?block)
  :precondition (and (is-tile ?tile)
    (is-block ?block)
    (next-row ?old-row ?new-row)
    (tile-at ?tile ?old-row ?col))
)
```



```

        (is-blank ?new-row ?col)
        (is-blank ?old-row ?col)))
:effect (and (not (tile-at ?tile ?old-row ?col))
             (tile-at ?tile ?new-row ?col)))

(:action move-tile-down-inside-block
:parameters (?tile ?old-row ?new-row ?col ?block ?up)
:precondition (and (is-tile ?tile)
                  (is-block ?block)
                  (is-miss-up ?up)
                  (next-row ?old-row ?new-row)
                  (tile-at ?tile ?old-row ?col)
                  (is-blank ?old-row ?col)
                  (not(is-blank ?new-row ?col))
                  (block-at ?block ?new-row ?col)
                  (block-missing ?block ?up)))
:effect (and (not (tile-at ?tile ?old-row ?col))
             (tile-at ?tile ?new-row ?col)))

(:action move-tile-down-outside-block
:parameters (?tile ?old-row ?new-row ?col ?block ?down)
:precondition (and (is-tile ?tile)
                  (is-block ?block)
                  (is-miss-down ?down)
                  (next-row ?old-row ?new-row)
                  (tile-at ?tile ?old-row ?col)
                  (is-blank ?new-row ?col)
                  (block-at ?block ?old-row ?col)
                  (block-missing ?block ?down)))
:effect (and (not (tile-at ?tile ?old-row ?col))
             (tile-at ?tile ?new-row ?col)))

(:action move-tile-down-from-block-to-block
:parameters (?tile ?old-row ?new-row ?col ?block1 ?block2 ?up ?down)
:precondition (and (is-tile ?tile)
                  (is-block ?block1)
                  (is-block ?block2)
                  (is-miss-up ?up)
                  (is-miss-down ?down)
                  (next-row ?new-row ?old-row)
                  (tile-at ?tile ?old-row ?col)
                  (block-at ?block1 ?old-row ?col)
                  (block-missing ?block1 ?down)
                  (block-at ?block2 ?new-row ?col)
                  (block-missing ?block2 ?up)))
:effect (and (not (tile-at ?tile ?old-row ?col))
             (tile-at ?tile ?new-row ?col)))

```

There is also another types of moves where we move with blocks. We can move with one block, or we can move with two blocks if we manage to position a small block inside a big block. When moving with blocks we have to update their position and also to update the blank squares. (blank squares are squares where there is no block). We can not move a big block inside a big block no matter the missing piece so we have to check that as well. Here is an example of moving with blocks down.

```

(:action move-tile-down-with-block
:parameters (?tile ?old-row ?new-row ?col ?block1 ?block2 ?down ?up)
:precondition (and (is-tile ?tile)
                  (is-block ?block1)
                  (is-block ?block2)

```

```

(is-miss-down ?down)
(is-miss-up ?up)
(next-row ?old-row ?new-row)
  (tile-at ?tile ?old-row ?col)
  (block-at ?block1 ?old-row ?col)
  (not (block-missing ?block1 ?down))
  (or (is-blank ?new-row ?col)
      (and (block-at ?block2 ?new-row ?col)
            (is-small-block ?block1)
            (is-big-block ?block2)
            (block-missing ?block2 ?up) )))
:effect (and (not (tile-at ?tile ?old-row ?col))
             (tile-at ?tile ?new-row ?col)
             (not (block-at ?block1 ?old-row ?col))
             (block-at ?block1 ?new-row ?col)
             (not(is-blank ?new-row ?col))
             (is-blank ?old-row ?col)))

(:action move-tile-down-with-two-block
:parameters (?tile ?old-row ?new-row ?col ?block1 ?block2 ?up ?down)
:precondition (and (is-tile ?tile)
                  (is-block ?block1)
                  (is-block ?block2)
                  (is-miss-down ?down)
                  (next-row ?old-row ?new-row)
                    (tile-at ?tile ?old-row ?col)
                    (block-at ?block1 ?old-row ?col)
                    (block-at ?block2 ?old-row ?col)
                    (is-small-block ?block1)
                    (is-big-block ?block2)
                    (not (block-missing ?block2 ?down))
                    (is-blank ?new-row ?col) )
:effect (and (not (tile-at ?tile ?old-row ?col))
             (tile-at ?tile ?new-row ?col)
             (not (block-at ?block1 ?old-row ?col))
             (block-at ?block1 ?new-row ?col)
             (not (block-at ?block2 ?old-row ?col))
             (block-at ?block2 ?new-row ?col)
             (not(is-blank ?new-row ?col))
             (is-blank ?old-row ?col)))

```

3.0.2 Boxup Level 1

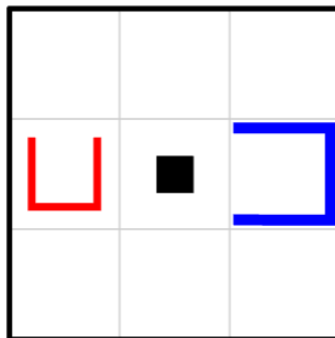


Figure 3.1: Boxup level 1

For this level we found the following solution:

```
(move-tile-up tile row2 row1 col2)
(move-tile-left tile row1 col2 col1)
(move-tile-down-inside-block tile row1 row2 col1 red-block miss-up)
(move-tile-right-with-block tile row2 col1 col2 red-block blue-block miss-right
 miss-left)
(move-tile-right-with-block tile row2 col2 col3 red-block blue-block miss-right
 miss-left)
; cost = 5 (unit cost)
```

3.0.3 Boxup Level 2

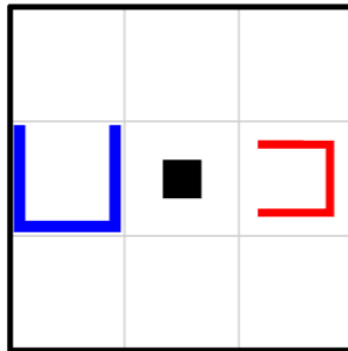


Figure 3.2: Boxup level 2

For this level we found the following solution:

```
(move-tile-right-inside-block tile row2 col2 col3 red-block miss-left)
(move-tile-up-with-block tile row2 row1 col3 red-block blue-block miss-up miss-
 down)
(move-tile-left-outside-block tile row1 col3 col2 red-block miss-left)
(move-tile-left tile row1 col2 col1)
(move-tile-down-inside-block tile row1 row2 col1 blue-block miss-up)
(move-tile-right-with-block tile row2 col1 col2 blue-block blue-block miss-right
 miss-left)
(move-tile-right-with-block tile row2 col2 col3 blue-block blue-block miss-right
 miss-left)
(move-tile-down-with-block tile row2 row3 col3 blue-block blue-block miss-down
 miss-up)
(move-tile-up-outside-block tile row3 row2 col3 blue-block miss-up)
(move-tile-left tile row2 col3 col2)
(move-tile-up tile row2 row1 col2)
(move-tile-right-inside-block tile row1 col2 col3 red-block miss-left)
(move-tile-down-with-block tile row1 row2 col3 red-block blue-block miss-down
 miss-up)
(move-tile-down-with-block tile row2 row3 col3 red-block blue-block miss-down
 miss-up)
; cost = 14 (unit cost)
```

3.0.4 Boxup Level 3

For this level we found the following solution:

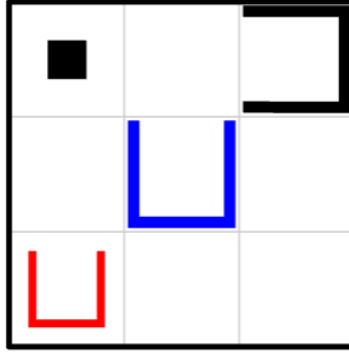


Figure 3.3: Boxup level 3

```

(move-tile-right tile row1 col1 col2)
(move-tile-down-inside-block tile row1 row2 col2 blue-block miss-up)
(move-tile-left-with-block tile row2 col2 col1 blue-block black-block miss-left
  miss-right)
(move-tile-up-outside-block tile row2 row1 col1 blue-block miss-up)
(move-tile-right tile row1 col1 col2)
(move-tile-right-inside-block tile row1 col2 col3 black-block miss-left)
(move-tile-down-with-block tile row1 row2 col3 black-block black-block miss-down
  miss-up)
(move-tile-down-with-block tile row2 row3 col3 black-block black-block miss-down
  miss-up)
(move-tile-left-outside-block tile row3 col3 col2 black-block miss-left)
(move-tile-up tile row3 row2 col2)
(move-tile-up tile row2 row1 col2)
(move-tile-left tile row1 col2 col1)
(move-tile-down-inside-block tile row1 row2 col1 blue-block miss-up)
(move-tile-right-with-block tile row2 col1 col2 blue-block black-block miss-
  right miss-left)
(move-tile-up-outside-block tile row2 row1 col2 blue-block miss-up)
(move-tile-left tile row1 col2 col1)
(move-tile-down tile row1 row2 col1)
(move-tile-down-inside-block tile row2 row3 col1 red-block miss-up)
(move-tile-right-with-block tile row3 col1 col2 red-block black-block miss-right
  miss-left)
(move-tile-right-with-block tile row3 col2 col3 red-block black-block miss-right
  miss-left)
(move-tile-up-with-two-block tile row3 row2 col3 red-block black-block miss-up)
(move-tile-up-with-two-block tile row2 row1 col3 red-block black-block miss-up)
(move-tile-left-with-block tile row1 col3 col2 red-block black-block miss-left
  miss-right)
(move-tile-down-with-block tile row1 row2 col2 red-block blue-block miss-down
  miss-up)
; cost = 24 (unit cost)

```

Appendix A

Your original code

A.0.1 Assignment 1

1. Mouse Search

```
def getSuccessorsWithoutReverse(successors, curr_dir):
    rev_dir = Actions.reverseDirection(curr_dir)
    reverseSuccessor = None
    for successor in successors:
        nextAction = successor[1]
        if rev_dir == nextAction: # i am trying to not go in the opposite
            direction
            reverseSuccessor = successor
    if reverseSuccessor is not None:
        successors.remove(reverseSuccessor)
    return successors

def mouseSearch(problem):
    from game import Directions
    current_state = problem.getStartState()
    solution = []
    curr_dir = Directions.EAST
    while not (problem.isGoalState(current_state)):
        successors = problem.getSuccessors(current_state)
        if len(successors) >= 2: # i am at a junction
            successorsWithoutReverse = getSuccessorsWithoutReverse(
                successors, curr_dir) # i dont want to go back
            nextState = choice(successorsWithoutReverse) # i will choose a
                random move
        elif len(successors) == 2: # i am at a passageway
            keepDirectionSuccessor = getSuccessorsWithoutReverse(successors
                , curr_dir)
            nextState = keepDirectionSuccessor[0] # since reverse is
                removed, there is only one option left
        else: # dead end
            nextState = successors[0] # the only option left is going back
            current_state = nextState[0]
            curr_dir = nextState[1]
            solution.append(curr_dir)
    return solution
```

2. Wall Flower Search

```
def desiredAction(action, hand):
```

```

if hand == "left":
    if action == Directions.WEST:
        return Directions.SOUTH
    elif action == Directions.SOUTH:
        return Directions.EAST
    elif action == Directions.EAST:
        return Directions.NORTH
    else:
        return Directions.WEST
else:
    if action == Directions.WEST:
        return Directions.NORTH
    elif action == Directions.NORTH:
        return Directions.EAST
    elif action == Directions.EAST:
        return Directions.SOUTH
    else:
        return Directions.WEST

def getDesiredSuccessor(successors, curr_dir, hand):
    desired_dir = desiredAction(curr_dir, hand)
    desiredSuccessor = None
    for successor in successors:
        nextAction = successor[1]
        if desired_dir == nextAction: # i am looking for the successor
            with the desired action
                desiredSuccessor = successor
    return desiredSuccessor

def getKeepDirectionSuccessor(successors, curr_dir): # assumes there are
    more than or 2 successors,
    # and the desired action is not possible
    keepDirSuccessor = None
    reverseSuccessor = None
    rev_dir = Actions.reverseDirection(curr_dir)
    for successor in successors:
        nextAction = successor[1]
        if curr_dir == nextAction: # successor that keeps my direction
            keepDirSuccessor = successor
        if rev_dir == nextAction: # i am trying to not go in the opposite
            direction
                reverseSuccessor = successor
    if reverseSuccessor is not None: # since there is more than one
        successor i dont need this
            successors.remove(reverseSuccessor)
    if keepDirSuccessor is not None:
        return keepDirSuccessor
    return successors[0] # in case there is no keep dir successor there is
    only one left successor

def wallFlowerSearch(problem):
    current_state = problem.getStartState()
    solution = []
    curr_dir = Directions.NORTH
    hand = "left"
    while not (problem.isGoalState(current_state)):
        successors = problem.getSuccessors(current_state)

```

```

        if len(successors) >= 2: # keep going forward
            desiredSuccessor = getDesiredSuccessor(successors, curr_dir,
hand)
            if desiredSuccessor is not None:
                nextState = desiredSuccessor
            else:
                nextState = getKeepDirectionSuccessor(successors, curr_dir)
        else: # dead end
            nextState = successors[0]
        current_state = nextState[0]
        curr_dir = nextState[1]
        solution.append(curr_dir)
    return solution

```

3. Bidirectional Search

```

def getOppositeDirections(directions):
    oppositeDirections = []
    for direction in directions:
        if direction == Directions.NORTH:
            oppositeDirections.append(Directions.SOUTH)
        elif direction == Directions.SOUTH:
            oppositeDirections.append(Directions.NORTH)
        elif direction == Directions.WEST:
            oppositeDirections.append(Directions.EAST)
        else:
            oppositeDirections.append(Directions.WEST)
    return oppositeDirections

def bidirectionalSearch(problem, heuristic=nullHeuristic):
    startFrontier = util.PriorityQueue()
    goalFrontier = util.PriorityQueue()
    allStatesInStartFrontier = set() # to visualize the nodes in the
frontier
    allStatesInGoalFrontier = set()
    exploredStatesByStart = []
    exploredStatesByGoal = []
    initial_state = problem.getStartState()
    end_state = problem.goal
    start_heuristic = heuristic(initial_state, problem)
    goal_heuristic = heuristic(end_state, problem)
    startFrontier.push((initial_state, []), start_heuristic)
    goalFrontier.push((end_state, []), goal_heuristic)

    while startFrontier.isEmpty() is not True and goalFrontier.isEmpty() is
not True:
        if startFrontier.isEmpty() is not True: # this is the search that
starts from the start
            currentState, currentActions = startFrontier.pop()
            if currentState not in exploredStatesByStart:
                exploredStatesByStart.append(currentState)
                if problem.isGoalState(currentState):
                    print "did not meet"
                    return currentActions
            if currentState in allStatesInGoalFrontier: # this means
we found a common path
                while not goalFrontier.isEmpty():

```

```

        current_state, current_actions = goalFrontier.pop()
        if current_state == currentState:
            current_actions.reverse() # we have to reverse
the states that come from the search
            # that was performed from goal to start
            solution = currentActions +
getOppositeDirections(current_actions) # this states specify
            # how the pacman should move from the goal to
the meeting point sa we have to
            # also reverse the actions(directions) inside
the states

            return solution
        for (nextState, nextAction, nextCost) in problem.
getSuccessors(currentState):
            # if common path not met, continue searching
            getHeuristics = heuristic(nextState, problem)
            # this is greedy searching but it can be bfs if no
heuristic is given
            startFrontier.push((nextState, currentActions + [
nextAction]), getHeuristics)
            allStatesInStartFrontier.add(nextState)

        if goalFrontier.isEmpty() is not True: # this is the search that
starts from the goal
            currentState, currentActions = goalFrontier.pop()
            if currentState not in exploredStatesByGoal:
                exploredStatesByGoal.append(currentState)
            if currentState == initial_state:
                print "did not meet"
                currentActions.reverse()
                return getOppositeDirections(currentActions)
            if currentState in allStatesInStartFrontier:
                while not startFrontier.isEmpty():
                    current_state, current_actions = startFrontier.pop
()

                    if current_state == currentState:
                        currentActions.reverse()
                        solution = current_actions +
getOppositeDirections(currentActions)
                        return solution
            for (nextState, nextAction, nextCost) in problem.
getSuccessors(currentState):
                getHeuristics = heuristic(nextState, problem)
                goalFrontier.push((nextState, currentActions + [
nextAction]), getHeuristics)
                allStatesInGoalFrontier.add(nextState)

```

A.0.2 Assignment 2

1. Mace4 Sudoku Generator (without seed)

```

assign(domain_size, 9).      % for a 9x9 puzzle
assign(max_seconds, 2).      % time limit
set(arithmetic).

```

```

formulas(sudoku_rules).

```

```

% At most one of each in each row.

```



```

all x all y1 all y2 (f(x, y1) = f(x, y2) -> y1 = y2).

% At most one of each in each column.

all x1 all x2 all y (f(x1, y) = f(x2, y) -> x1 = x2).

all x same_interval(x,x).
all x all y (same_interval(x,y) -> same_interval(y,x)).
all x all y all z (same_interval(x,y) & same_interval(y,z) -> same_interval
(x,z)).

same_interval(0,1).
same_interval(1,2).

same_interval(3,4).
same_interval(4,5).

same_interval(6,7).
same_interval(7,8).

-same_interval(0,3).
-same_interval(3,6).
-same_interval(0,6).

all x1 all y1 all x2 all y2
(
    same_interval(x1,x2) &
    same_interval(y1,y2) &
    f(x1, y1) = f(x2, y2)
->
    x1 = x2 &
    y1 = y2
).

end_of_list.

```

2. Mace4 Sudoku Generator (with seed)

```

assign(domain_size, 9).          % for a 9x9 puzzle
assign(max_seconds, 2).          % time limit
set(arithmetic).

formulas(sudoku_rules).

% At most one of each in each row.

all x all y1 all y2 (f(x, y1) = f(x, y2) -> y1 = y2).

% At most one of each in each column.

all x1 all x2 all y (f(x1, y) = f(x2, y) -> x1 = x2).

all x same_interval(x,x).
all x all y (same_interval(x,y) -> same_interval(y,x)).
all x all y all z (same_interval(x,y) & same_interval(y,z) -> same_interval
(x,z)).

same_interval(0,1).
same_interval(1,2).

same_interval(3,4).

```

```

same_interval(4,5).

same_interval(6,7).
same_interval(7,8).

-same_interval(0,3).
-same_interval(3,6).
-same_interval(0,6).

all x1 all y1 all x2 all y2
(
    same_interval(x1,x2) &
    same_interval(y1,y2) &
    f(x1, y1) = f(x2, y2)
->
    x1 = x2 &
    y1 = y2
).

end_of_list.

formulas(sample_puzzle).
f(3,0) = 7.
end_of_list.

```

3. Mace4 Sudoku Solver

```

assign(domain_size, 9).      % for a 9x9 puzzle
assign(max_seconds, 2).      % time limit
set(arithmetic).

formulas(sudoku_rules).

% At most one of each in each row.

all x all y1 all y2 (f(x, y1) = f(x, y2) -> y1 = y2).

% At most one of each in each column.

all x1 all x2 all y (f(x1, y) = f(x2, y) -> x1 = x2).

all x same_interval(x,x).
all x all y (same_interval(x,y) -> same_interval(y,x)).
all x all y all z (same_interval(x,y) & same_interval(y,z) -> same_interval
(x,z)).

same_interval(0,1).
same_interval(1,2).

same_interval(3,4).
same_interval(4,5).

same_interval(6,7).
same_interval(7,8).

-same_interval(0,3).
-same_interval(3,6).
-same_interval(0,6).

all x1 all y1 all x2 all y2
(

```

```

        same_interval(x1,x2) &
        same_interval(y1,y2) &
        f(x1, y1) = f(x2, y2)
->
    x1 = x2 &
    y1 = y2
).

end_of_list.

```

```

formulas(sample_puzzle).
f(0,5) = 2.
f(0,8) = 0.
f(1,0) = 5.
f(1,1) = 4.
f(1,4) = 8.
f(1,6) = 2.
f(2,0) = 1.
f(2,2) = 3.
f(2,4) = 7.
f(2,6) = 5.
f(3,0) = 7.
f(3,2) = 4.
f(4,3) = 0.
f(4,8) = 7.
f(5,2) = 5.
f(5,4) = 3.
f(5,7) = 6.
f(6,5) = 5.
f(6,7) = 0.
f(6,8) = 1.
f(7,3) = 6.
f(7,4) = 0.
f(7,5) = 3.
f(8,6) = 8.
f(8,8) = 5.
end_of_list.

```

4. Python Code

```

import re
import os
import random
import numpy

class GenerateSudokuBoard(object):
    """
    Will generate a board to play sudoku
    """

    def __init__(self):
        self.testing_board = [
            [0, 0, 3, 0, 2, 0, 6, 0, 0],
            [9, 0, 0, 3, 0, 5, 0, 0, 1],
            [0, 0, 1, 8, 0, 6, 4, 0, 0],
            [0, 0, 8, 1, 0, 2, 9, 0, 0],
            [7, 0, 0, 0, 0, 0, 0, 0, 8],
            [0, 0, 6, 7, 0, 8, 2, 0, 0],
            [0, 0, 2, 6, 0, 9, 5, 0, 0],
            [8, 0, 0, 2, 0, 3, 0, 0, 9],

```

```

        [0, 0, 5, 0, 1, 0, 3, 0, 0]
    ]
    self.sudoku_board = []

    def generate_board_with_mace4(self):
        self.__create_sudoku_generator_file()
        self.__call_mace4_to_generate_full_board()
        self.sudoku_board = self.__parse_full_board_from_mace4_file() #
    get a full board from mace4
        self.__print_sudoku()
        self.sudoku_board = self.__shuffle(self.sudoku_board)
        self.__print_sudoku()
        self.__complex_solve()
        self.__print_sudoku()

        self.__generate_sudoku_solver_file() # generate mace4 file for the
    current board
        self.__call_mace4_to_verify_sudoku()
        if self.__solution_is_unique():
            print("unique")
        else:
            print("not unique")
        return self.sudoku_board

    def __complex_solve(self):
        positions = numpy.arange(0, 81, 1) # list of positions
        numpy.random.shuffle(positions) # shuffle the positions
        board_clone = self.__save_a_copy_board() # duplicate the original
    board

        values_removed = 0
        current_index = 0

        while values_removed < 64 and current_index < 81:
            row = positions[current_index] // 9
            col = positions[current_index] % 9

            # remove number from current index
            self.sudoku_board[row][col] = 0

            self.__generate_sudoku_solver_file() # generate mace4 file for
    the current board
            self.__call_mace4_to_verify_sudoku() # solve the current board

            if not self.__solution_is_unique():
                # put the number back
                self.sudoku_board[row][col] = board_clone[row][col]
            else:
                # it's still solvable, move on
                values_removed += 1

            current_index += 1

    def __save_a_copy_board(self):
        copy_board = []
        for a_list in self.sudoku_board:
            copy_board.append(list.copy(a_list))
        return copy_board

    def __shuffle(self, old_board):

```

```

        board = numpy.array(old_board)
        group = 3 # we can only safely shuffle between group of 3
consecutive rows/columns
        number_of_shuffles = random.randrange(1, 10, 1)
        for i in range(number_of_shuffles):
            numpy.random.shuffle(board.reshape(board.shape[0] // group, -1,
board.shape[1])) # shuffles rows
            board = numpy.transpose(board) # transpose the board so that
we can shuffle the columns

        return board.tolist()

def __simple_solve(self):
    # generation algorithm
    to_remove = 81 # number of attempts to remove numbers from the
board

    while to_remove > 0:
        # select a random row and a column
        row = random.randint(0, 8)
        col = random.randint(0, 8)

        # if that position has already been removed, search again
        while self.sudoku_board[row][col] == 0:
            row = random.randint(0, 8)
            col = random.randint(0, 8)

        # memorize the value to be removed, then remove it
        old_value = self.sudoku_board[row][col]
        self.sudoku_board[row][col] = 0

        self.__generate_sudoku_solver_file() # generate mace4 file for
the current board
        self.__call_mace4_to_verify_sudoku() # solve the current board
        if not self.__solution_is_unique():
            # the board does not have a unique solution, put the number
back and try again
            self.sudoku_board[row][col] = old_value
            to_remove -= 1

def __print_sudoku(self):
    for r in range(9):
        for c in range(9):
            if c != 8:
                print(str(self.sudoku_board[r][c]), end=" ")
            else:
                print(str(self.sudoku_board[r][c]))
        print("\n")

def __call_mace4_to_generate_full_board(self):
    os.system("mace4 -m 1 -f sudoku_generator_with_1_value.in |
interformat > mace4_complete_sudoku.out")

def __parse_full_board_from_mace4_file(self):
    sudoku_board = []
    sudoku_file = "mace4_complete_sudoku.out"
    with open(sudoku_file, 'r') as inp:
        lines = inp.read()
        lines_list = lines.splitlines()
        sudoku_lines = []

```

```

        for (index, line) in enumerate(lines_list):
            if index >= 2 and index <= 10:
                sudoku_lines.append(line.strip())
        for line in sudoku_lines:
            parsed_line = re.split(',|\]|\\', line)
            sudoku_line = [ele for ele in parsed_line if ele.strip()]
            sudoku_board.append(self.__get_row_as_list(sudoku_line))
    return sudoku_board

def __get_row_as_list(self, line):
    row_list = []
    for element in line:
        number = int(element)
        if number == 0:
            number = 9 # transform zeros to 9
        row_list.append(number)
    return row_list

def __generate_sudoku_solver_file(self):
    sudoku_generator_file = "sudoku_generator.in"
    sudoku_solver_file = "sudoku_solver.in"
    with open(sudoku_solver_file, "w") as f2:
        with open(sudoku_generator_file, "r") as f1:
            for line in f1:
                f2.write(line)
            f2.write("formulas(sample_puzzle).\n")

        for r in range(9):
            for c in range(9):
                if self.sudoku_board[r][c] != 0:
                    if self.sudoku_board[r][c] == 9:
                        f2.write(f"f({r},{c}) = 0.\n")
                    else:
                        f2.write(f"f({r},{c}) = {self.sudoku_board[r][c]
}}.\n")

            f2.write("end_of_list.\n")

def __create_sudoku_generator_file(self):
    sudoku_generator_file = "sudoku_generator.in"
    sudoku_solver_file = "sudoku_generator_with_1_value.in"
    with open(sudoku_solver_file, "w") as f2:
        with open(sudoku_generator_file, "r") as f1:
            for line in f1:
                f2.write(line)
            f2.write("formulas(sample_puzzle).\n")

        random_row = random.randint(0, 8)
        random_column = random.randint(0, 8)
        random_nr = random.randint(0, 8)

        f2.write(f"f({random_row},{random_column}) = {random_nr}.\n")

        f2.write("end_of_list.\n")

def __call_mace4_to_verify_sudoku(self):
    os.system("mace4 -m 2 -f sudoku_solver.in | interpfomat >
mace4_verify_sudoku.out")

def __solution_is_unique(self):

```

```

no_of_solution = None
solutions_obtained_file = "mace4_verify_sudoku.out"
with open(solutions_obtained_file, "r") as inp:
    for line in inp.readlines():
        elements_parsed = self.__parse_interpretation_line(line)
        if elements_parsed[0] == "interpretation":
            no_of_solution = elements_parsed[3]
if (no_of_solution) == '1':
    return True
return False

def __parse_interpretation_line(self, line):
    line_parsed = re.split(',|=|\\(|\\[|\\]', line)
    remove_space = [x.strip(' ') for x in line_parsed]
    delete_empty = [ele for ele in remove_space if ele.strip()]
    return delete_empty

sudokuBoardGenerator = GenerateSudokuBoard()
sudokuBoardGenerator.generate_board_with_mace4()

```

A.0.3 Assignment 3

1. Domain

```

(define (domain sliding-tile)

  (:predicates
    (is-tile ?t)
    (is-block ?b)
    (is-miss-left ?l)
    (is-miss-right ?l)
    (is-miss-up ?l)
    (is-miss-down ?l)
    (tile-at ?t ?r ?c)
    (is-blank ?r ?c)
    (next-row ?r1 ?r2)
    (next-column ?c1 ?c2)
    (block-at ?b ?r ?c)
    (block-missing ?b ?x)
    (is-small-block ?b)
    (is-big-block ?b)
  )

  (:action move-tile-down
    :parameters (?tile ?old-row ?new-row ?col)
    :precondition (and (is-tile ?tile)
      (next-row ?old-row ?new-row)
      (tile-at ?tile ?old-row ?col)
      (is-blank ?new-row ?col)
      (is-blank ?old-row ?col))
    :effect (and (not (tile-at ?tile ?old-row ?col))
      (tile-at ?tile ?new-row ?col)))

  (:action move-tile-down-inside-block
    :parameters (?tile ?old-row ?new-row ?col ?block ?up)
    :precondition (and (is-tile ?tile)
      (is-block ?block)
      (is-miss-up ?up)

```

```

      (next-row ?old-row ?new-row)
      (tile-at ?tile ?old-row ?col)
      (is-blank ?old-row ?col)
      (not(is-blank ?new-row ?col))
      (block-at ?block ?new-row ?col)
      (block-missing ?block ?up))
:effect (and (not (tile-at ?tile ?old-row ?col))
             (tile-at ?tile ?new-row ?col)))

(:action move-tile-down-outside-block
:parameters (?tile ?old-row ?new-row ?col ?block ?down)
:precondition (and (is-tile ?tile)
                  (is-block ?block)
                  (is-miss-down ?down)
                  (next-row ?old-row ?new-row)
                  (tile-at ?tile ?old-row ?col)
                  (is-blank ?new-row ?col)
                  (block-at ?block ?old-row ?col)
                  (block-missing ?block ?down)))
:effect (and (not (tile-at ?tile ?old-row ?col))
             (tile-at ?tile ?new-row ?col)))

(:action move-tile-down-from-block-to-block
:parameters (?tile ?old-row ?new-row ?col ?block1 ?block2 ?up ?down)
:precondition (and (is-tile ?tile)
                  (is-block ?block1)
                  (is-block ?block2)
                  (is-miss-up ?up)
                  (is-miss-down ?down)
                  (next-row ?new-row ?old-row)
                  (tile-at ?tile ?old-row ?col)
                  (block-at ?block1 ?old-row ?col)
                  (block-missing ?block1 ?down)
                  (block-at ?block2 ?new-row ?col)
                  (block-missing ?block2 ?up)))
:effect (and (not (tile-at ?tile ?old-row ?col))
             (tile-at ?tile ?new-row ?col)))

(:action move-tile-up
:parameters (?tile ?old-row ?new-row ?col)
:precondition (and (is-tile ?tile)
                  (next-row ?new-row ?old-row)
                  (tile-at ?tile ?old-row ?col)
                  (is-blank ?new-row ?col)
                  (is-blank ?old-row ?col)))
:effect (and (not (tile-at ?tile ?old-row ?col))
             (tile-at ?tile ?new-row ?col)))

(:action move-tile-up-inside-block
:parameters (?tile ?old-row ?new-row ?col ?block ?down)
:precondition (and (is-tile ?tile)
                  (is-block ?block)
                  (is-miss-down ?down)
                  (next-row ?new-row ?old-row)
                  (tile-at ?tile ?old-row ?col)
                  (is-blank ?old-row ?col)
                  (not(is-blank ?new-row ?col))
                  (block-at ?block ?new-row ?col)
                  (block-missing ?block ?down)))

```



```

:effect (and (not (tile-at ?tile ?old-row ?col))
              (tile-at ?tile ?new-row ?col)))

(:action move-tile-up-outside-block
:parameters (?tile ?old-row ?new-row ?col ?block ?up)
:precondition (and (is-tile ?tile)
                  (is-block ?block)
                  (is-miss-up ?up)
                  (next-row ?new-row ?old-row)
                  (tile-at ?tile ?old-row ?col)
                  (is-blank ?new-row ?col)
                  (block-at ?block ?old-row ?col)
                  (block-missing ?block ?up))
:effect (and (not (tile-at ?tile ?old-row ?col))
              (tile-at ?tile ?new-row ?col)))

(:action move-tile-up-from-block-to-block
:parameters (?tile ?old-row ?new-row ?col ?block1 ?block2 ?up ?down)
:precondition (and (is-tile ?tile)
                  (is-block ?block1)
                  (is-block ?block2)
                  (is-miss-up ?up)
                  (is-miss-down ?down)
                  (next-row ?new-row ?old-row)
                  (tile-at ?tile ?old-row ?col)
                  (block-at ?block1 ?old-row ?col)
                  (block-missing ?block1 ?up)
                  (block-at ?block2 ?new-row ?col)
                  (block-missing ?block2 ?down))
:effect (and (not (tile-at ?tile ?old-row ?col))
              (tile-at ?tile ?new-row ?col)))

(:action move-tile-right
:parameters (?tile ?row ?old-col ?new-col)
:precondition (and (is-tile ?tile)
                  (next-column ?old-col ?new-col)
                  (tile-at ?tile ?row ?old-col)
                  (is-blank ?row ?new-col)
                  (is-blank ?row ?old-col))
:effect (and (not (tile-at ?tile ?row ?old-col))
              (tile-at ?tile ?row ?new-col)))

(:action move-tile-right-inside-block
:parameters (?tile ?row ?old-col ?new-col ?block ?left)
:precondition (and (is-tile ?tile)
                  (is-block ?block)
                  (is-miss-left ?left)
                  (next-column ?old-col ?new-col)
                  (tile-at ?tile ?row ?old-col)
                  (is-blank ?row ?old-col)
                  (not (is-blank ?row ?new-col))
                  (block-at ?block ?row ?new-col)
                  (block-missing ?block ?left))
:effect (and (not (tile-at ?tile ?row ?old-col))
              (tile-at ?tile ?row ?new-col)))

(:action move-tile-right-outside-block
:parameters (?tile ?row ?old-col ?new-col ?block ?right)
:precondition (and (is-tile ?tile)
                  (is-block ?block)

```

```

        (is-miss-right ?right)
        (next-column ?old-col ?new-col)
            (tile-at ?tile ?row ?old-col)
            (is-blank ?row ?new-col)
            (block-at ?block ?row ?old-col)
            (block-missing ?block ?right))
:effect (and (not (tile-at ?tile ?row ?old-col))
            (tile-at ?tile ?row ?new-col)))

(:action move-tile-right-from-block-to-block
:parameters (?tile ?row ?old-col ?new-col ?block1 ?block2 ?left ?right)
:precondition (and (is-tile ?tile)
    (is-block ?block1)
    (is-block ?block2)
    (is-miss-left ?left)
    (is-miss-right ?right)
    (next-column ?new-col ?old-col)
        (tile-at ?tile ?row ?old-col)
        (block-at ?block1 ?row ?old-col)
        (block-missing ?block1 ?right)
        (block-at ?block2 ?row ?new-col)
        (block-missing ?block2 ?left))
:effect (and (not (tile-at ?tile ?row ?old-col))
            (tile-at ?tile ?row ?new-col)))

(:action move-tile-left
:parameters (?tile ?row ?old-col ?new-col)
:precondition (and (is-tile ?tile)
    (next-column ?new-col ?old-col)
        (tile-at ?tile ?row ?old-col)
        (is-blank ?row ?new-col)
        (is-blank ?row ?old-col))
:effect (and (not (tile-at ?tile ?row ?old-col))
            (tile-at ?tile ?row ?new-col)))

(:action move-tile-left-inside-block
:parameters (?tile ?row ?old-col ?new-col ?block ?right)
:precondition (and (is-tile ?tile)
    (is-block ?block)
    (is-miss-right ?right)
    (next-column ?new-col ?old-col)
        (tile-at ?tile ?row ?old-col)
        (is-blank ?row ?old-col)
        (not(is-blank ?row ?new-col))
        (block-at ?block ?row ?new-col)
        (block-missing ?block ?right))
:effect (and (not (tile-at ?tile ?row ?old-col))
            (tile-at ?tile ?row ?new-col)))

(:action move-tile-left-outside-block
:parameters (?tile ?row ?old-col ?new-col ?block ?left)
:precondition (and (is-tile ?tile)
    (is-block ?block)
    (is-miss-left ?left)
    (next-column ?new-col ?old-col)
        (tile-at ?tile ?row ?old-col)
        (is-blank ?row ?new-col)
        (block-at ?block ?row ?old-col)
        (block-missing ?block ?left))
:effect (and (not (tile-at ?tile ?row ?old-col))
            (tile-at ?tile ?row ?new-col)))

```

```

(tile-at ?tile ?row ?new-col)))

(:action move-tile-left-from-block-to-block
:parameters (?tile ?row ?old-col ?new-col ?block1 ?block2 ?left ?right)
:precondition (and (is-tile ?tile)
  (is-block ?block1)
  (is-block ?block2)
  (is-miss-left ?left)
  (is-miss-right ?right)
  (next-column ?new-col ?old-col)
  (tile-at ?tile ?row ?old-col)
  (block-at ?block1 ?row ?old-col)
  (block-missing ?block1 ?left)
  (block-at ?block2 ?row ?new-col)
  (block-missing ?block2 ?right))
:effect (and (not (tile-at ?tile ?row ?old-col))
  (tile-at ?tile ?row ?new-col)))

; moves with blocks

(:action move-tile-down-with-block
:parameters (?tile ?old-row ?new-row ?col ?block1 ?block2 ?down ?up)
:precondition (and (is-tile ?tile)
  (is-block ?block1)
  (is-block ?block2)
  (is-miss-down ?down)
  (is-miss-up ?up)
  (next-row ?old-row ?new-row)
  (tile-at ?tile ?old-row ?col)
  (block-at ?block1 ?old-row ?col)
  (not (block-missing ?block1 ?down))
  (or (is-blank ?new-row ?col)
    (and (block-at ?block2 ?new-row ?col)
      (is-small-block ?block1)
      (is-big-block ?block2)
      (block-missing ?block2 ?up) )))
:effect (and (not (tile-at ?tile ?old-row ?col))
  (tile-at ?tile ?new-row ?col)
  (not (block-at ?block1 ?old-row ?col))
  (block-at ?block1 ?new-row ?col)
  (not (is-blank ?new-row ?col))
  (is-blank ?old-row ?col)))

(:action move-tile-up-with-block
:parameters (?tile ?old-row ?new-row ?col ?block1 ?block2 ?up ?down)
:precondition (and (is-tile ?tile)
  (is-block ?block1)
  (is-block ?block2)
  (is-miss-down ?down)
  (is-miss-up ?up)
  (next-row ?new-row ?old-row )
  (tile-at ?tile ?old-row ?col)
  (block-at ?block1 ?old-row ?col)
  (not (block-missing ?block1 ?up))
  (or (is-blank ?new-row ?col)
    (and (block-at ?block2 ?new-row ?col)
      (is-small-block ?block1)
      (is-big-block ?block2)
      (block-missing ?block2 ?down) )))
:effect (and (not (tile-at ?tile ?old-row ?col))

```

```

        (tile-at ?tile ?new-row ?col)
        (not (block-at ?block1 ?old-row ?col))
        (block-at ?block1 ?new-row ?col)
        (not(is-blank ?new-row ?col))
        (is-blank ?old-row ?col)))

(:action move-tile-right-with-block
:parameters (?tile ?row ?old-col ?new-col ?block1 ?block2 ?right ?left)
:precondition (and (is-tile ?tile)
  (is-block ?block1)
  (is-block ?block2)
  (is-miss-right ?right)
  (is-miss-left ?left)
  (next-column ?old-col ?new-col)
  (tile-at ?tile ?row ?old-col)
  (block-at ?block1 ?row ?old-col)
  (not (block-missing ?block1 ?right))
  (or (is-blank ?row ?new-col)
    (and (block-at ?block2 ?row ?new-col)
      (is-small-block ?block1)
      (is-big-block ?block2)
      (block-missing ?block2 ?left) )))
:effect (and (not (tile-at ?tile ?row ?old-col))
  (tile-at ?tile ?row ?new-col)
  (not (block-at ?block1 ?row ?old-col))
  (block-at ?block1 ?row ?new-col)
  (not(is-blank ?row ?new-col))
  (is-blank ?row ?old-col)))

(:action move-tile-left-with-block
:parameters (?tile ?row ?old-col ?new-col ?block1 ?block2 ?left ?right)
:precondition (and (is-tile ?tile)
  (is-block ?block1)
  (is-block ?block2)
  (is-miss-right ?right)
  (is-miss-left ?left)
  (next-column ?new-col ?old-col)
  (tile-at ?tile ?row ?old-col)
  (block-at ?block1 ?row ?old-col)
  (not (block-missing ?block1 ?left))
  (or (is-blank ?row ?new-col)
    (and (block-at ?block2 ?row ?new-col)
      (is-small-block ?block1)
      (is-big-block ?block2)
      (block-missing ?block2 ?right)
      )))
:effect (and (not (tile-at ?tile ?row ?old-col))
  (tile-at ?tile ?row ?new-col)
  (not (block-at ?block1 ?row ?old-col))
  (block-at ?block1 ?row ?new-col)
  (not(is-blank ?row ?new-col))
  (is-blank ?row ?old-col)))

(:action move-tile-up-with-two-block
:parameters (?tile ?old-row ?new-row ?col ?block1 ?block2 ?up)
:precondition (and (is-tile ?tile)
  (is-block ?block1)
  (is-block ?block2)
  (is-miss-up ?up)
  (next-row ?new-row ?old-row )

```

```

        (tile-at ?tile ?old-row ?col)
        (block-at ?block1 ?old-row ?col)
        (block-at ?block2 ?old-row ?col)
        (is-small-block ?block1)
        (is-big-block ?block2)
        (not (block-missing ?block2 ?up))
        (is-blank ?new-row ?col) )
:effect (and (not (tile-at ?tile ?old-row ?col))
            (tile-at ?tile ?new-row ?col)
            (not (block-at ?block1 ?old-row ?col))
            (block-at ?block1 ?new-row ?col)
            (not (block-at ?block2 ?old-row ?col))
            (block-at ?block2 ?new-row ?col)
            (not(is-blank ?new-row ?col))
            (is-blank ?old-row ?col)))

(:action move-tile-down-with-two-block
:parameters (?tile ?old-row ?new-row ?col ?block1 ?block2 ?up ?down)
:precondition (and (is-tile ?tile)
                  (is-block ?block1)
                  (is-block ?block2)
                  (is-miss-down ?down)
                  (next-row ?old-row ?new-row)
                  (tile-at ?tile ?old-row ?col)
                  (block-at ?block1 ?old-row ?col)
                  (block-at ?block2 ?old-row ?col)
                  (is-small-block ?block1)
                  (is-big-block ?block2)
                  (not (block-missing ?block2 ?down))
                  (is-blank ?new-row ?col) )
:effect (and (not (tile-at ?tile ?old-row ?col))
            (tile-at ?tile ?new-row ?col)
            (not (block-at ?block1 ?old-row ?col))
            (block-at ?block1 ?new-row ?col)
            (not (block-at ?block2 ?old-row ?col))
            (block-at ?block2 ?new-row ?col)
            (not(is-blank ?new-row ?col))
            (is-blank ?old-row ?col)))

(:action move-tile-left-with-two-block
:parameters (?tile ?row ?old-col ?new-col ?block1 ?block2 ?left)
:precondition (and (is-tile ?tile)
                  (is-block ?block1)
                  (is-block ?block2)
                  (is-miss-left ?left)
                  (next-column ?new-col ?old-col)
                  (tile-at ?tile ?row ?old-col)
                  (block-at ?block1 ?row ?old-col)
                  (block-at ?block2 ?row ?old-col)
                  (is-small-block ?block1)
                  (is-big-block ?block2)
                  (not (block-missing ?block2 ?left))
                  (is-blank ?row ?new-col))
:effect (and (not (tile-at ?tile ?row ?old-col))
            (tile-at ?tile ?row ?new-col)
            (not (block-at ?block1 ?row ?old-col))
            (block-at ?block1 ?row ?new-col)
            (not (block-at ?block2 ?row ?old-col))
            (block-at ?block2 ?row ?new-col)
            (not(is-blank ?row ?new-col)))

```

```

(is-blank ?row ?old-col)))

(:action move-tile-right-with-two-block
 :parameters (?tile ?row ?old-col ?new-col ?block1 ?block2 ?right)
 :precondition (and (is-tile ?tile)
 (is-block ?block1)
 (is-block ?block2)
 (is-miss-right ?right)
 (next-column ?old-col ?new-col)
 (tile-at ?tile ?row ?old-col)
 (block-at ?block1 ?row ?old-col)
 (block-at ?block2 ?row ?old-col)
 (is-small-block ?block1)
 (is-big-block ?block2)
 (not (block-missing ?block2 ?right))
 (is-blank ?row ?new-col))
 :effect (and (not (tile-at ?tile ?row ?old-col))
 (tile-at ?tile ?row ?new-col)
 (not (block-at ?block1 ?row ?old-col))
 (block-at ?block1 ?row ?new-col)
 (not (block-at ?block2 ?row ?old-col))
 (block-at ?block2 ?row ?new-col)
 (not(is-blank ?row ?new-col))
 (is-blank ?row ?old-col)))

)

```

2. Problem 1

```

(define (problem eight-puzzle)
 (:domain sliding-tile)
 (:objects
 tile
 red-block blue-block
 row1 row2 row3
 col1 col2 col3
 miss-up miss-down miss-left miss-right
 )
 (:init
 (next-row row1 row2) (next-column col1 col2)
 (next-row row2 row3) (next-column col2 col3)
 (tile-at tile row2 col2)
 (is-blank row1 col1)
 (is-blank row1 col2)
 (is-blank row1 col3)
 ;(is-blank row2 col1)
 (is-blank row2 col2)
 ;(is-blank row2 col3)
 (is-blank row3 col1)
 (is-blank row3 col2)
 (is-blank row3 col3)
 (block-at red-block row2 col1)
 (block-missing red-block miss-up)
 (block-at blue-block row2 col3)
 (block-missing blue-block miss-left)
 (is-tile tile)
 (is-block red-block)
 (is-small-block red-block)
 (is-block blue-block)
 (is-big-block blue-block)
 (is-miss-left miss-left)

```

```

(is-miss-right miss-right)
(is-miss-up miss-up)
(is-miss-down miss-down)
)
(:goal
(or
(and (block-at blue-block row1 col1) (block-at red-block row1 col1))
(and (block-at blue-block row1 col2) (block-at red-block row1 col2))
(and (block-at blue-block row1 col3) (block-at red-block row1 col3))
(and (block-at blue-block row2 col1) (block-at red-block row2 col1))
(and (block-at blue-block row2 col2) (block-at red-block row2 col2))
(and (block-at blue-block row2 col3) (block-at red-block row2 col3))
(and (block-at blue-block row3 col1) (block-at red-block row3 col1))
(and (block-at blue-block row3 col2) (block-at red-block row3 col2))
(and (block-at blue-block row3 col3) (block-at red-block row3 col3))))
)

```

3. Problem 2

```

(define (problem eight-puzzle)
  (:domain sliding-tile)
  (:objects
    tile
    red-block blue-block
    row1 row2 row3
    col1 col2 col3
    miss-up miss-down miss-left miss-right
  )
  (:init
    (next-row row1 row2)          (next-column col1 col2)
    (next-row row2 row3)          (next-column col2 col3)
    (tile-at tile row2 col2)
    (is-blank row1 col1)
    (is-blank row1 col2)
    (is-blank row1 col3)
    ;(is-blank row2 col1)
    (is-blank row2 col2)
    ;(is-blank row2 col3)
    (is-blank row3 col1)
    (is-blank row3 col2)
    (is-blank row3 col3)
    (block-at red-block row2 col3)
    (block-missing red-block miss-left)
    (block-at blue-block row2 col1)
    (block-missing blue-block miss-up)
    (is-tile tile)
    (is-block red-block)
    (is-small-block red-block)
    (is-block blue-block)
    (is-big-block blue-block)
    (is-miss-left miss-left)
    (is-miss-right miss-right)
    (is-miss-up miss-up)
    (is-miss-down miss-down)
  )
  (:goal
    (or
      (and (block-at blue-block row1 col1) (block-at red-block row1 col1))
      (and (block-at blue-block row1 col2) (block-at red-block row1 col2))
      (and (block-at blue-block row1 col3) (block-at red-block row1 col3))
      (and (block-at blue-block row2 col1) (block-at red-block row2 col1))
    )
  )

```

```

    (and (block-at blue-block row2 col2) (block-at red-block row2 col2))
    (and (block-at blue-block row2 col3) (block-at red-block row2 col3))
    (and (block-at blue-block row3 col1) (block-at red-block row3 col1))
    (and (block-at blue-block row3 col2) (block-at red-block row3 col2))
    (and (block-at blue-block row3 col3) (block-at red-block row3 col3)))
)

```

4. Problem 3

```

(define (problem eight-puzzle)
  (:domain sliding-tile)
  (:objects
    tile
    red-block blue-block black-block
    row1 row2 row3
    col1 col2 col3
    miss-up miss-down miss-left miss-right
  )
  (:init
    (next-row row1 row2)          (next-column col1 col2)
    (next-row row2 row3)          (next-column col2 col3)
    (tile-at tile row1 col1)
    (is-blank row1 col1)
    (is-blank row1 col2)
    ;(is-blank row1 col3)
    (is-blank row2 col1)
    ;(is-blank row2 col2)
    (is-blank row2 col3)
    ;(is-blank row3 col1)
    (is-blank row3 col2)
    (is-blank row3 col3)
    (block-at red-block row3 col1)
    (block-missing red-block miss-up)
    (is-block red-block)
    (is-small-block red-block)
    (block-at blue-block row2 col2)
    (block-missing blue-block miss-up)
    (is-block blue-block)
    (is-big-block blue-block)
    (block-at black-block row1 col3)
    (block-missing black-block miss-left)
    (is-block black-block)
    (is-big-block black-block)
    (is-tile tile)
    (is-miss-left miss-left)
    (is-miss-right miss-right)
    (is-miss-up miss-up)
    (is-miss-down miss-down)
  )
  (:goal
    (or
      (and (block-at blue-block row1 col1) (block-at red-block row1 col1))
      (and (block-at blue-block row1 col2) (block-at red-block row1 col2))
      (and (block-at blue-block row1 col3) (block-at red-block row1 col3))
      (and (block-at blue-block row2 col1) (block-at red-block row2 col1))
      (and (block-at blue-block row2 col2) (block-at red-block row2 col2))
      (and (block-at blue-block row2 col3) (block-at red-block row2 col3))
      (and (block-at blue-block row3 col1) (block-at red-block row3 col1))
      (and (block-at blue-block row3 col2) (block-at red-block row3 col2))
      (and (block-at blue-block row3 col3) (block-at red-block row3 col3)))
    )
  )
)

```