Formal Verification and Synthesis Project:
Submitted by: Shani Berkovitz and Hila Levy

Part 1:

1. **Define an FDS for a general n × m Sokoban board. Use XSB format to describe the board.**

   FDS:
   $V = \{b, i, j, n, m, d\}$

   b is the board and is represented via an nXm array.
   i,j (int) is the current location of the warehouse keeper on the board - rows, columns.
   d (char) is the direction the warehouse keeper wants to move in:
   L - left, R - right, U - up, D - down.
   n,m (int) are 2 constants of the size of the board.

   ρ(b, i, j, d):

   The warehouse keeper wants to move when next to her is a floor: (allow move)
   $ρ_L = (b[i][j] = @) \wedge (b[i][j-1] = -) \wedge (b'[i][j] = -) \wedge (b'[i][j-1] = @)$
   $ρ_R = (b[i][j] = @) \wedge (b[i][j+1] = -) \wedge (b'[i][j] = -) \wedge (b'[i][j+1] = @)$
   $ρ_U = (b[i][j] = @) \wedge (b[i-1][j] = -) \wedge (b'[i][j] = -) \wedge (b'[i-1][j] = @)$
   $ρ_D = (b[i][j] = @) \wedge (b[i+1][j] = -) \wedge (b'[i][j] = -) \wedge (b'[i+1][j] = @)$

   The warehouse keeper wants to move when next to her is a goal: (allow move)
   $ρ_L = (b[i][j] = @) \wedge (b[i][j-1] = .) \wedge (b'[i][j] = -) \wedge (b'[i][j-1] = +)$
   $ρ_R = (b[i][j] = @) \wedge (b[i][j+1] = .) \wedge (b'[i][j] = -) \wedge (b'[i][j+1] = +)$
   $ρ_U = (b[i][j] = @) \wedge (b[i-1][j] = .) \wedge (b'[i][j] = -) \wedge (b'[i-1][j] = +)$
   $ρ_D = (b[i][j] = @) \wedge (b[i+1][j] = .) \wedge (b'[i][j] = -) \wedge (b'[i+1][j] = +)$

   The warehouse keeper wants to move when next to her is a wall: (don't allow move)
   $ρ_L = (b[i][j] = @) \wedge (b[i][j-1] = \#) \wedge (b'[i][j] = @) \wedge (b'[i][j-1] = \#)$
   $ρ_R = (b[i][j] = @) \wedge (b[i][j+1] = \#) \wedge (b'[i][j] = @) \wedge (b'[i][j+1] = \#)$
   $ρ_U = (b[i][j] = @) \wedge (b[i-1][j] = \#) \wedge (b'[i][j] = @) \wedge (b'[i-1][j] = \#)$
   $ρ_D = (b[i][j] = @) \wedge (b[i+1][j] = \#) \wedge (b'[i][j] = @) \wedge (b'[i+1][j] = \#)$

The warehouse keeper wants to move when next to her is a box on goal: (don't allow move)
   $ρ_L = (b[i][j] = @) \wedge (b[i][j-1] = *) \wedge (b'[i][j] = @) \wedge (b'[i][j-1] = *)$
   $ρ_R = (b[i][j] = @) \wedge (b[i][j+1] = *) \wedge (b'[i][j] = @) \wedge (b'[i][j+1] = *)$
   $ρ_U = (b[i][j] = @) \wedge (b[i-1][j] = *) \wedge (b'[i][j] = @) \wedge (b'[i-1][j] = *)$

$\rho_D = (b[i][j] = @) \wedge (b[i+1][j] = *) \wedge (b'[i][j] = @) \wedge (b'[i+1][j] = *)$

The warehouse keeper wants to move when next to her is a box, and next to the box is a floor: (allow move)

$\rho_L = (b[i][j] = @) \wedge (b[i][j-1] = \$) \wedge (b[i][j-2] = -)$
$\wedge (b'[i][j] = -) \wedge (b'[i][j-1] = @) \wedge (b'[i][j-2] = \$)$

$\rho_R = (b[i][j] = @) \wedge (b[i][j+1] = \$) \wedge (b[i][j+2] = -)$
$\wedge (b'[i][j] = -) \wedge (b'[i][j+1] = @) \wedge (b'[i][j+2] = \$)$

$\rho_U = (b[i][j] = @) \wedge (b[i-1][j] = \$) \wedge (b[i-2][j] = -)$
$\wedge (b'[i][j] = -) \wedge (b'[i-1][j] = @) \wedge (b'[i-2][j] = \$)$

$\rho_D = (b[i][j] = @) \wedge (b[i+1][j] = \$) \wedge (b[i+2][j] = -)$
$\wedge (b'[i][j] = -) \wedge (b'[i+1][j] = @) \wedge (b'[i+2][j] = \$)$

The warehouse keeper wants to move when next to her is a box, and next to the box is a wall: (don’t allow move)

$\rho_L = (b[i][j] = @) \wedge (b[i][j-1] = \$) \wedge (b[i][j-2] = \#)$
$\wedge (b'[i][j] = @) \wedge (b'[i][j-1] = \$) \wedge (b'[i][j-2] = \#)$

$\rho_R = (b[i][j] = @) \wedge (b[i][j+1] = \$) \wedge (b[i][j+2] = \#)$
$\wedge (b'[i][j] = @) \wedge (b'[i][j+1] = \$) \wedge (b'[i][j+2] = \#)$

$\rho_U = (b[i][j] = @) \wedge (b[i-1][j] = \$) \wedge (b[i-2][j] = \#)$
$\wedge (b'[i][j] = @) \wedge (b'[i-1][j] = \$) \wedge (b'[i-2][j] = \#)$

$\rho_D = (b[i][j] = @) \wedge (b[i+1][j] = \$) \wedge (b[i+2][j] = \#)$
$\wedge (b'[i][j] = @) \wedge (b'[i+1][j] = \$) \wedge (b'[i+2][j] = \#)$

The warehouse keeper wants to move when next to her is a box, and next to the box is another box: (don’t allow move)

$\rho_L = (b[i][j] = @) \wedge (b[i][j-1] = \$) \wedge (b[i][j-2] = \$)$
$\wedge (b'[i][j] = @) \wedge (b'[i][j-1] = \$) \wedge (b'[i][j-2] = \$)$

$\rho_R = (b[i][j] = @) \wedge (b[i][j+1] = \$) \wedge (b[i][j+2] = \$)$
$\wedge (b'[i][j] = @) \wedge (b'[i][j+1] = \$) \wedge (b'[i][j+2] = \$)$

$\rho_U = (b[i][j] = @) \wedge (b[i-1][j] = \$) \wedge (b[i-2][j] = \$)$
$\wedge (b'[i][j] = @) \wedge (b'[i-1][j] = \$) \wedge (b'[i-2][j] = \$)$

$\rho_D = (b[i][j] = @) \wedge (b[i+1][j] = \$) \wedge (b[i+2][j] = \$)$
$\wedge (b'[i][j] = @) \wedge (b'[i+1][j] = \$) \wedge (b'[i+2][j] = \$)$

The warehouse keeper wants to move when next to her is a box, and next to the box is another box on goal: (don't allow move)

$\rho_L = (b[i][j] = @) \wedge (b[i][j-1] = \$) \wedge (b[i][j-2] = *)$
$\wedge (b'[i][j] = @) \wedge (b'[i][j-1] = \$) \wedge (b'[i][j-2] = *)$

$\rho_R = (b[i][j] = @) \wedge (b[i][j+1] = \$) \wedge (b[i][j+2] = *)$
$\wedge (b'[i][j] = @) \wedge (b'[i][j+1] = \$) \wedge (b'[i][j+2] = *)$

$\rho_U = (b[i][j] = @) \wedge (b[i-1][j] = \$) \wedge (b[i-2][j] = *)$
$\wedge (b'[i][j] = @) \wedge (b'[i-1][j] = \$) \wedge (b'[i-2][j] = *)$

$\rho_D = (b[i][j] = @) \wedge (b[i+1][j] = \$) \wedge (b[i+2][j] = *)$
$\wedge (b'[i][j] = @) \wedge (b'[i+1][j] = \$) \wedge (b'[i+2][j] = *)$

The warehouse keeper wants to move when next to her is a box, and next to the box is a goal: (allow move)

$\rho_L = (b[i][j] = @) \wedge (b[i][j-1] = \$) \wedge (b[i][j-2] = .)$
$\wedge (b'[i][j] = -) \wedge (b'[i][j-1] = @) \wedge (b'[i][j-2] = *)$

$\rho_R = (b[i][j] = @) \wedge (b[i][j+1] = \$) \wedge (b[i][j+2] = .)$
$\wedge (b'[i][j] = -) \wedge (b'[i][j+1] = @) \wedge (b'[i][j+2] = *)$

$\rho_U = (b[i][j] = @) \wedge (b[i-1][j] = \$) \wedge (b[i-2][j] = .)$
$\wedge (b'[i][j] = -) \wedge (b'[i-1][j] = @) \wedge (b'[i-2][j] = *)$

$\rho_D = (b[i][j] = @) \wedge (b[i+1][j] = \$) \wedge (b[i+2][j] = -)$
$\wedge (b'[i][j] = -) \wedge (b'[i+1][j] = @) \wedge (b'[i+2][j] = *)$

2. **Define a general temporal logic specification for a win of the Sokoban board.**
Assuming the number of boxes and goals is equal (like we were told in class):
Winning in Sokoban board is achieved by placing all the boxes in a goal place, which means there are no $ by the end of the game (only *).
In logical expression: $F(\neg(\$))$.
Which means finally there are no boxes not on goals in the board.

Part 2:

1. Using Python, or another coding language, and your answers to Part 1, automate definition of given input boards into SMV models. These models should contain both the model and the temporal logic formulae defining a win.

In this section, we used python coding language in order to convert the boards into SMV models. In those SMV models we've also defined a winning state, in order to decide whether each board is winnable or not.

Our code can be found in -

https://github.com/ShaniBerkovitz/Sokoban/tree/main

2. Run each of these models in nuXmv. Indicate the commands you used to run nuXmv, as well as screenshots of the nuXmv outputs.

We will now present the general commands that we used in order to run the nuXmv files. Along with a screenshot of every board and its outputs.

In order to run the nuXmv file we used in python the command:

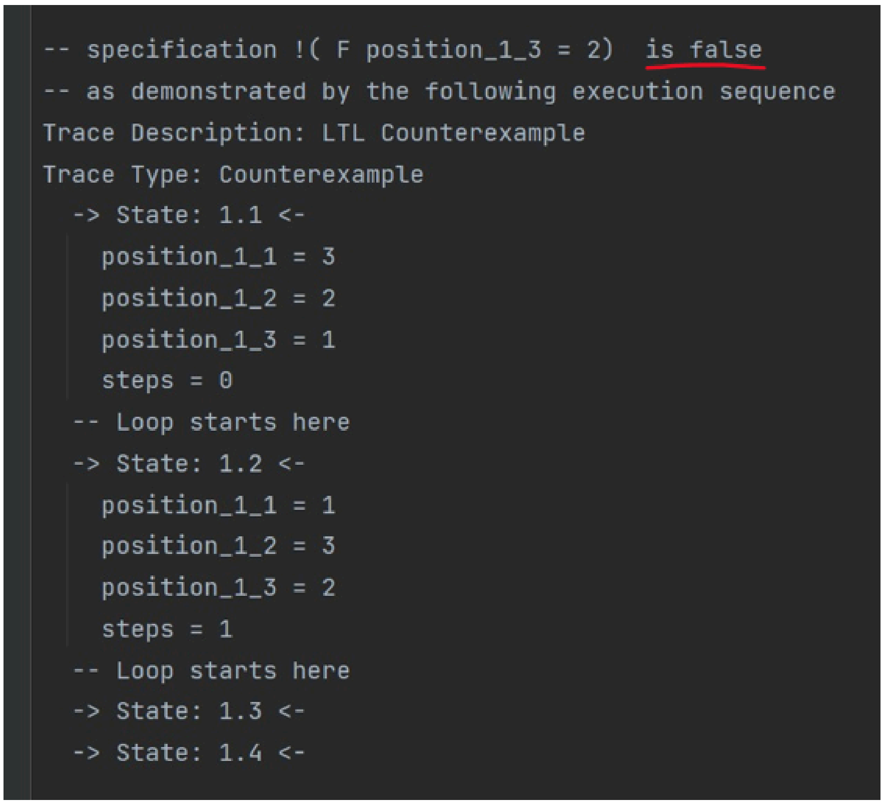
```
subprocess.Popen(["nuXmv", name_model], stdin=subprocess.PIPE,
stdout=subprocess.PIPE, universal_newlines=True)
```

In case that didn't work properly, we used the follow commands in CMD:

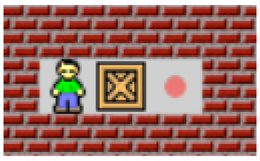
```
nuXmv -int
read_model -i sokoban.smv
flatten_hierarchy
encode_variables
build_model -f -m Threshold
check_ltlspec
```

The screenshot of boards:

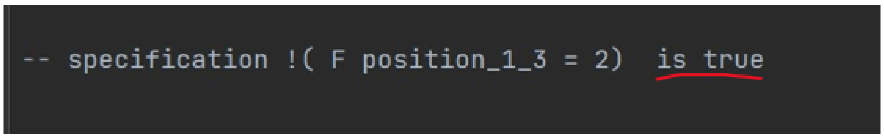
Board 1:



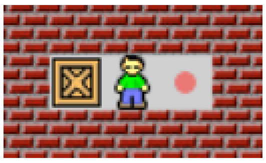
```
-- specification !( F position_1_3 = 2)  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    position_1_1 = 3
    position_1_2 = 2
    position_1_3 = 1
    steps = 0
  -- Loop starts here
  -> State: 1.2 <-
    position_1_1 = 1
    position_1_2 = 3
    position_1_3 = 2
    steps = 1
  -- Loop starts here
  -> State: 1.3 <-
  -> State: 1.4 <-
```



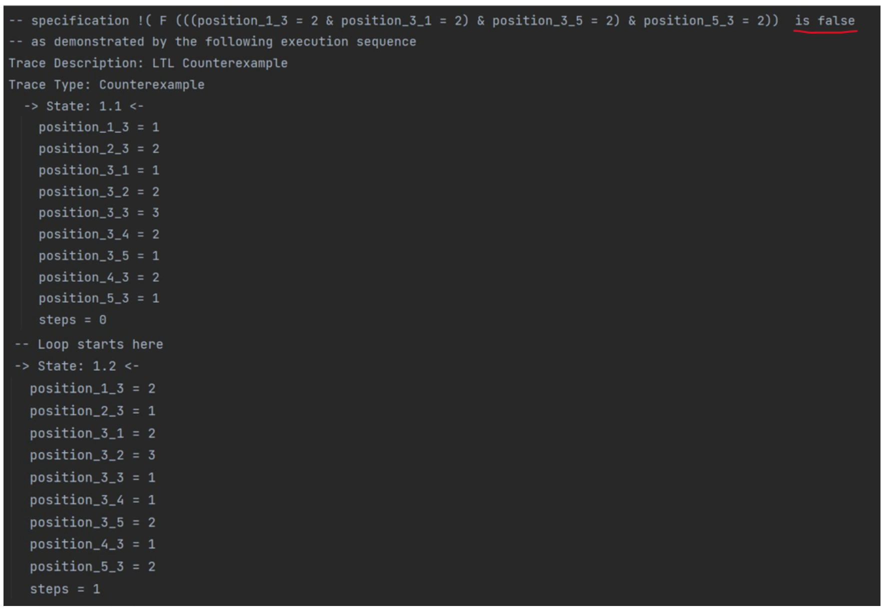
Board 2:



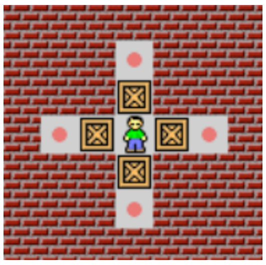
```
-- specification !( F position_1_3 = 2)  is true
```



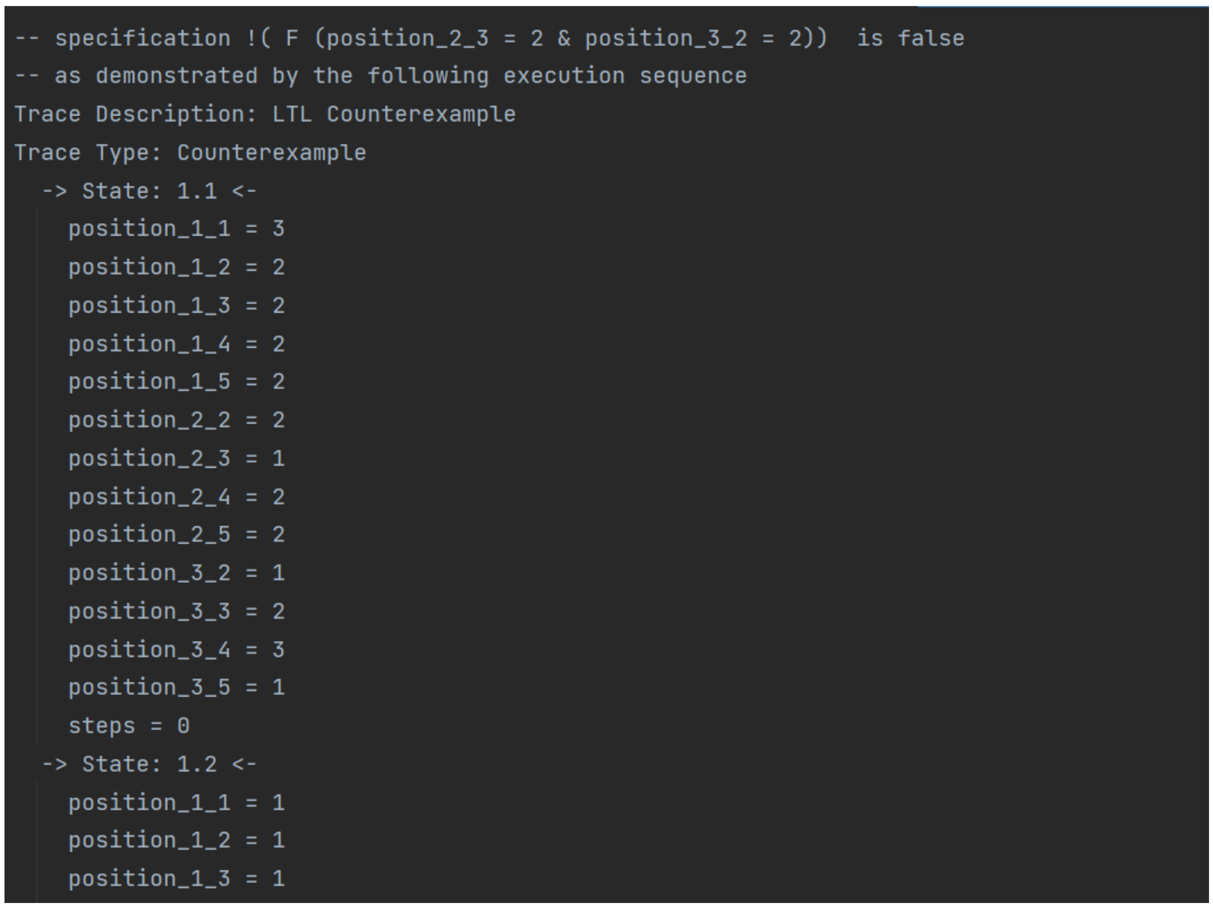
Board 4:



```
-- specification !( F (((position_1_3 = 2 & position_3_1 = 2) & position_3_5 = 2) & position_5_3 = 2))  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    position_1_3 = 1
    position_2_3 = 2
    position_3_1 = 1
    position_3_2 = 2
    position_3_3 = 3
    position_3_4 = 2
    position_3_5 = 1
    position_4_3 = 2
    position_5_3 = 1
    steps = 0
 -- Loop starts here
 -> State: 1.2 <-
   position_1_3 = 2
   position_2_3 = 1
   position_3_1 = 2
   position_3_2 = 3
   position_3_3 = 1
   position_3_4 = 1
   position_3_5 = 2
   position_4_3 = 1
   position_5_3 = 2
   steps = 1
```



Another board, that we have made:



```
-- specification !( F (position_2_3 = 2 & position_3_2 = 2))  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    position_1_1 = 3
    position_1_2 = 2
    position_1_3 = 2
    position_1_4 = 2
    position_1_5 = 2
    position_2_2 = 2
    position_2_3 = 1
    position_2_4 = 2
    position_2_5 = 2
    position_3_2 = 1
    position_3_3 = 2
    position_3_4 = 3
    position_3_5 = 1
    steps = 0
  -> State: 1.2 <-
    position_1_1 = 1
    position_1_2 = 1
    position_1_3 = 1
```

(The outfile is too big to fit in this document, the output is present in the git repository)
This is our board in XSB:

New Board 1:

```
#######
#@----#
##$.$-#
##.---#
#######
```

3. For each board, indicate if it is winnable. If it is, indicate your winning solution in LURD format.

Board 1:

R

Board 4:

U-D-R-L-D-U-L-R

Board 7:

L-L-U-R-R-R-R-R-D-R-U

Part 3:

1. Measure performance of nuXmv’s BDD and SAT Solver engines on each of the models.

	BDD	**SAT**
Board 1	0.03126 [Sec]	0.0159 [Sec] (!)
Board 4	0.03538 [Sec]	0.02785 [Sec] (!)
New Board 1	2.63687 [Sec]	2.44507 [Sec] (!)
New Board 2	0.21014 [Sec]	0.18137 [Sec] (!)

(!) - The better one.

New Board 1:
```
#######
#@----#
##$.$-#
##.---#
#######
```

New Board 2:
```
#######
#@#####
#$-####
#.-$-.#
##-$-.#
#######
```

2. Compare the performance of the two engines. Is one engine more efficient than the other?
After comparing between these two engines, we can see that with solvable boards the SAT engine is better in a significant difference.

Part 4:

1. Break the problem into sub-problems by solving the boards iteratively. For example, solve for one box at a time. Indicate the temporal logic formulae used for each iteration.

In this part we replaced the board_solving with the sokoban_iterative function that solves the sokoban puzzle iteratively.

we first parse the board and identify the positions of boxes and goals, and then iterate over each pair of box and goal:
for each pair we generate a corresponding smv file, run it, and check the output to determine if the board is winnable.

for each iteration, we measure and print the runtime and whether the box has reached its goal. After all of the iterations finished, we print the total runtime of the program.

The temporal logic formula for each iteration is $F(\neg(\$_i))$, Where $\$_i$ indicates the box we chose in the i-th iteration.

essentially, we check if the box we chose has reached its goal.

2. Indicate runtime for each iteration, as well as the total number of iterations needed for a given board.

In this part we were asked to choose only winnable boards so we ran the boards from part3.

	# Of Iterations	**Runtime For Each Iteration**	**Total Runtime**
Board 1	1	0.0170 [Sec]	0.0170 [Sec]
Board 4	4	0.0323, 0.0322, 0.0297, 0.0253 [Sec]	0.1245 [Sec]
New Board 1	2	1.8661, 3.3395 [Sec]	5.2222 [Sec]
New Board 2	3	0.1737, 0.2052, 0.2043 [Sec]	0.5989 [Sec]

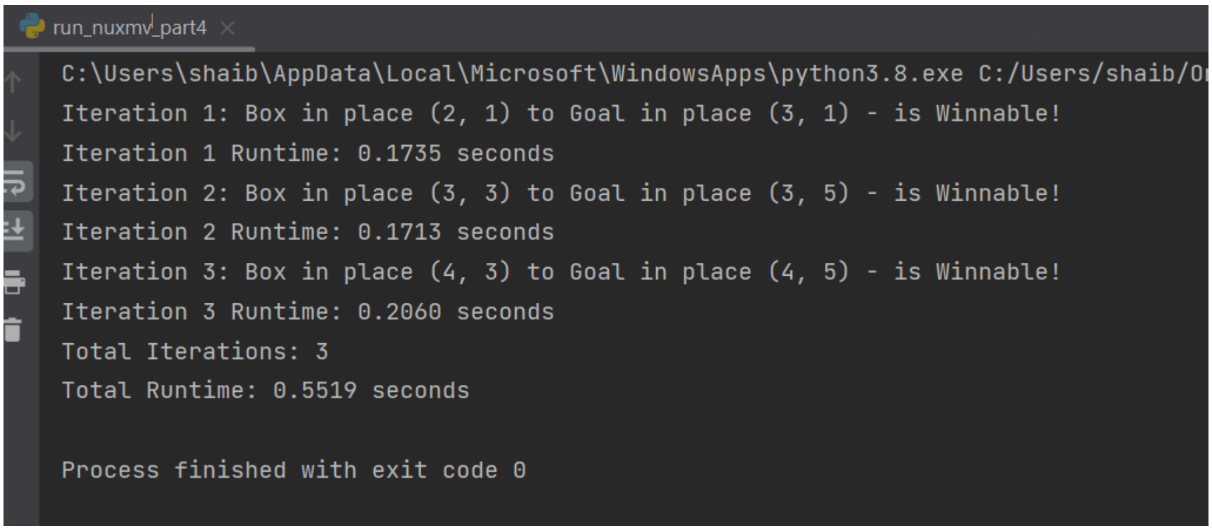
3. Test your iterative solution on larger more complex Sokoban boards.

We tested our iterative solution on the NewBoard2 and NewBoard3.

NewBoard2:

```
#######
#@#####
#$-####
#.-$-.#
##-$-.#
#######
```

The resultes:



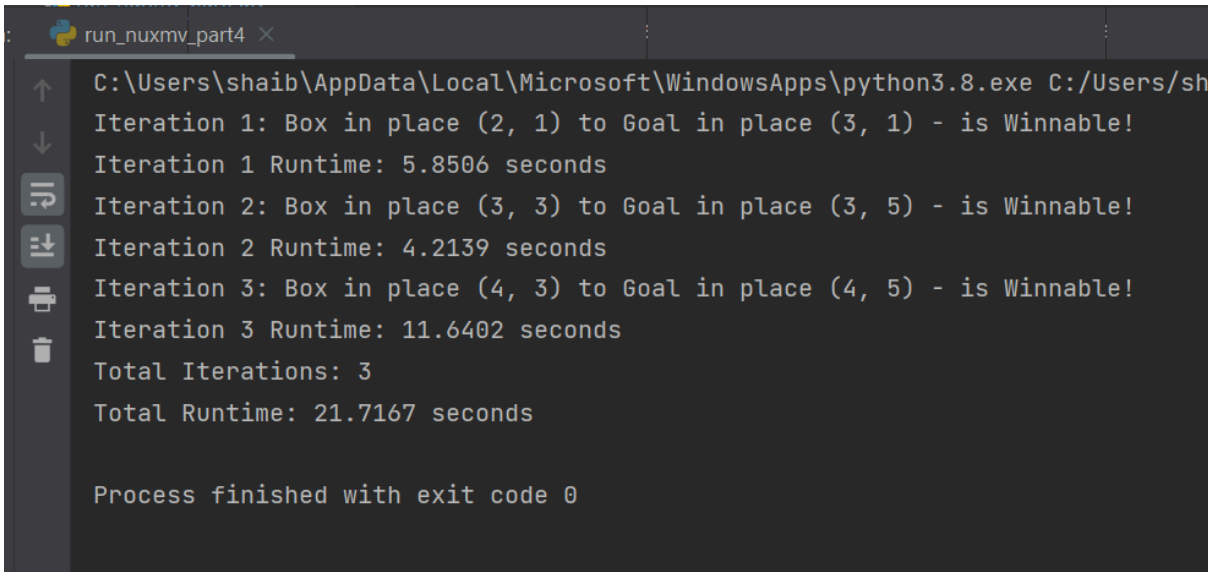
```
C:\Users\shaib\AppData\Local\Microsoft\WindowsApps\python3.8.exe C:/Users/shaib/O
Iteration 1: Box in place (2, 1) to Goal in place (3, 1) - is Winnable!
Iteration 1 Runtime: 0.1735 seconds
Iteration 2: Box in place (3, 3) to Goal in place (3, 5) - is Winnable!
Iteration 2 Runtime: 0.1713 seconds
Iteration 3: Box in place (4, 3) to Goal in place (4, 5) - is Winnable!
Iteration 3 Runtime: 0.2060 seconds
Total Iterations: 3
Total Runtime: 0.5519 seconds

Process finished with exit code 0
```

NewBoard3:

```
#######
#@#####
#$----#
#.-$-.#
##-$-.#
#######
```

The resultes:



```
C:\Users\shaib\AppData\Local\Microsoft\WindowsApps\python3.8.exe C:/Users/sh
Iteration 1: Box in place (2, 1) to Goal in place (3, 1) - is Winnable!
Iteration 1 Runtime: 5.8506 seconds
Iteration 2: Box in place (3, 3) to Goal in place (3, 5) - is Winnable!
Iteration 2 Runtime: 4.2139 seconds
Iteration 3: Box in place (4, 3) to Goal in place (4, 5) - is Winnable!
Iteration 3 Runtime: 11.6402 seconds
Total Iterations: 3
Total Runtime: 21.7167 seconds

Process finished with exit code 0
```