CS4023D Artificial Intelligence Assignment 1

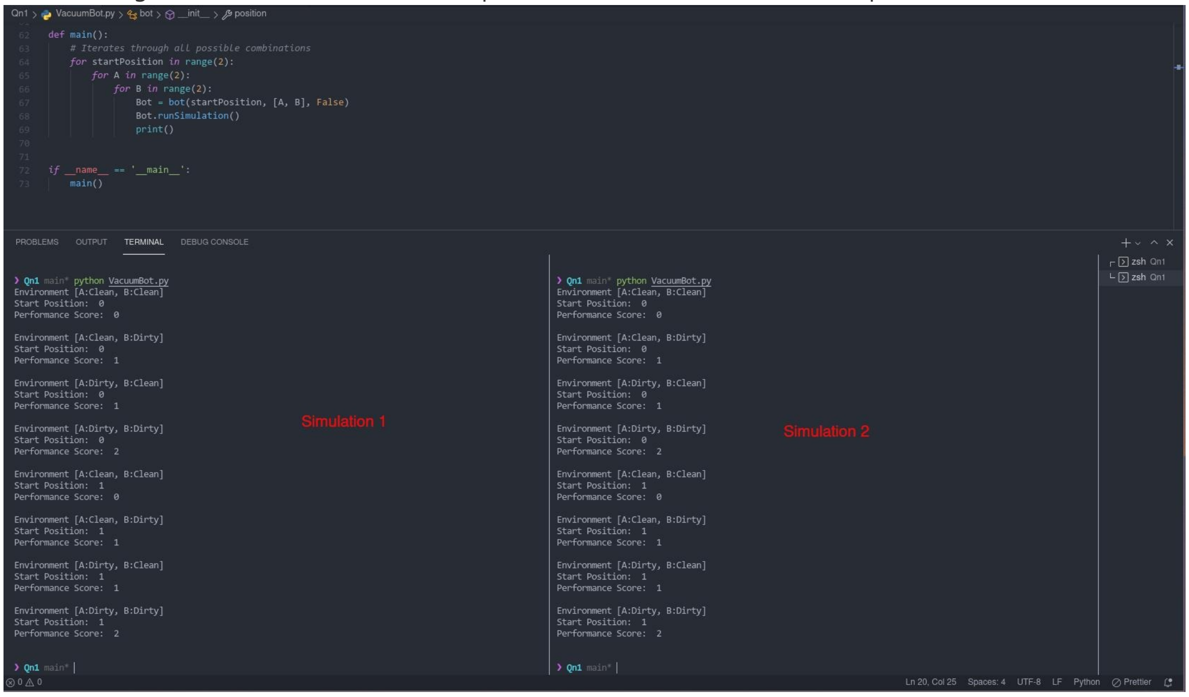
By Dev Sony, B180297CS

The question, report and source code can be found here. Github Repo

Solution 1

We assume the bot can correctly perceive if an area is dirty or not, and it will take either left or right from an area at random. If it moves outside the environment, the bot is restricted from moving. For a set of all assumptions, refer the State Space Search Graph or the Question's pdf.

After running 2 simulations for 1000 timesteps each, these are the results for all possible combinations.

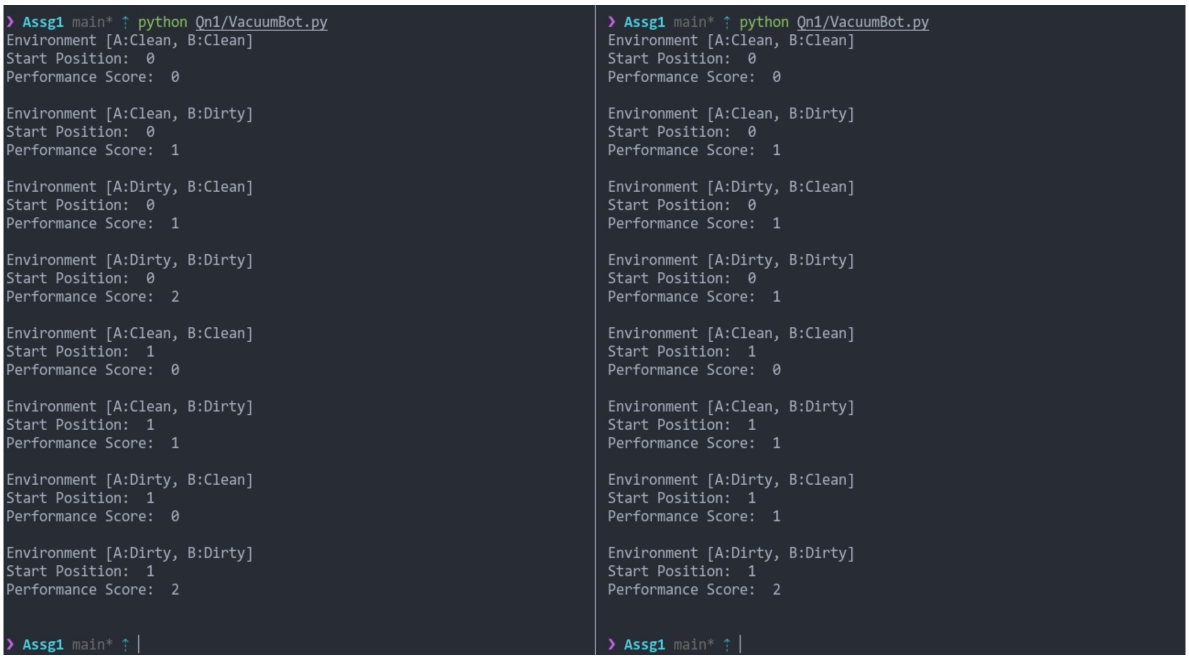


```python
def main():
    # Iterates through all possible combinations
    for startPosition in range(2):
        for A in range(2):
            for B in range(2):
                Bot = bot(startPosition, [A, B], False)
                Bot.runSimulation()
                print()


if __name__ == '__main__':
    main()
```

```
❯ Qn1 main* python VacuumBot.py
Environment [A:Clean, B:Clean]
Start Position:  0
Performance Score:  0

Environment [A:Clean, B:Dirty]
Start Position:  0
Performance Score:  1

Environment [A:Dirty, B:Clean]
Start Position:  0
Performance Score:  1

Environment [A:Dirty, B:Dirty]
Start Position:  0
Performance Score:  2

Environment [A:Clean, B:Clean]
Start Position:  1
Performance Score:  0

Environment [A:Clean, B:Dirty]
Start Position:  1
Performance Score:  1

Environment [A:Dirty, B:Clean]
Start Position:  1
Performance Score:  1

Environment [A:Dirty, B:Dirty]
Start Position:  1
Performance Score:  2

❯ Qn1 main* |
```

```
❯ Qn1 main* python VacuumBot.py
Environment [A:Clean, B:Clean]
Start Position:  0
Performance Score:  0

Environment [A:Clean, B:Dirty]
Start Position:  0
Performance Score:  1

Environment [A:Dirty, B:Clean]
Start Position:  0
Performance Score:  1

Environment [A:Dirty, B:Dirty]
Start Position:  0
Performance Score:  2

Environment [A:Clean, B:Clean]
Start Position:  1
Performance Score:  0

Environment [A:Clean, B:Dirty]
Start Position:  1
Performance Score:  1

Environment [A:Dirty, B:Clean]
Start Position:  1
Performance Score:  1

Environment [A:Dirty, B:Dirty]
Start Position:  1
Performance Score:  2

❯ Qn1 main* |
```

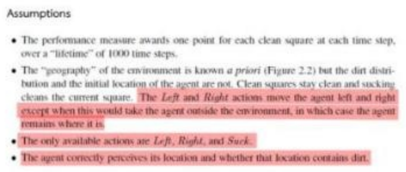
As it is pretty evident, it reaches a saturation point well before 1000 timesteps and hence all iterations would be the same. If the timesteps were less, say 3, the output would be as follows:



```
❯ Assg1 main* ⇡ python Qn1/VacuumBot.py
Environment [A:Clean, B:Clean]
Start Position:  0
Performance Score:  0

Environment [A:Clean, B:Dirty]
Start Position:  0
Performance Score:  1

Environment [A:Dirty, B:Clean]
Start Position:  0
Performance Score:  1

Environment [A:Dirty, B:Dirty]
Start Position:  0
Performance Score:  2

Environment [A:Clean, B:Clean]
Start Position:  1
Performance Score:  0

Environment [A:Clean, B:Dirty]
Start Position:  1
Performance Score:  1

Environment [A:Dirty, B:Clean]
Start Position:  1
Performance Score:  0

Environment [A:Dirty, B:Dirty]
Start Position:  1
Performance Score:  2


❯ Assg1 main* ⇡ |
```

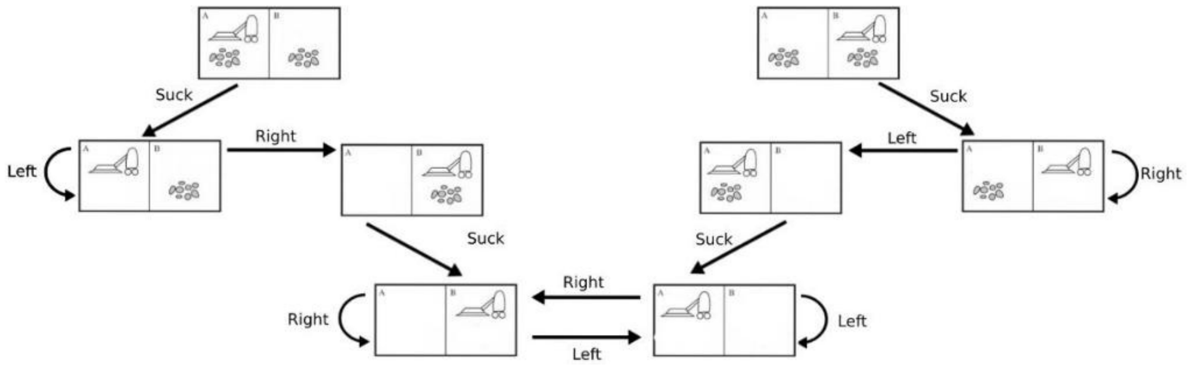
```
❯ Assg1 main* ⇡ python Qn1/VacuumBot.py
Environment [A:Clean, B:Clean]
Start Position:  0
Performance Score:  0

Environment [A:Clean, B:Dirty]
Start Position:  0
Performance Score:  1

Environment [A:Dirty, B:Clean]
Start Position:  0
Performance Score:  1

Environment [A:Dirty, B:Dirty]
Start Position:  0
Performance Score:  1

Environment [A:Clean, B:Clean]
Start Position:  1
Performance Score:  0

Environment [A:Clean, B:Dirty]
Start Position:  1
Performance Score:  1

Environment [A:Dirty, B:Clean]
Start Position:  1
Performance Score:  1

Environment [A:Dirty, B:Dirty]
Start Position:  1
Performance Score:  2


❯ Assg1 main* ⇡ |
```

State Space Search Graph



Assumptions

- The performance measure awards one point for each clean square at each time step, over a "lifetime" of 1000 time steps.
- The "geography" of the environment is known *a priori* (Figure 2.2) but the dirt distribution and the initial location of the agent are not. Clean squares stay clean and sucking cleans the current square. The *Left* and *Right* actions move the agent left and right except when this would take the agent outside the environment, in which case the agent remains where it is.
- The only available actions are *Left*, *Right*, and *Suck*.
- The agent correctly perceives its location and whether that location contains dirt.

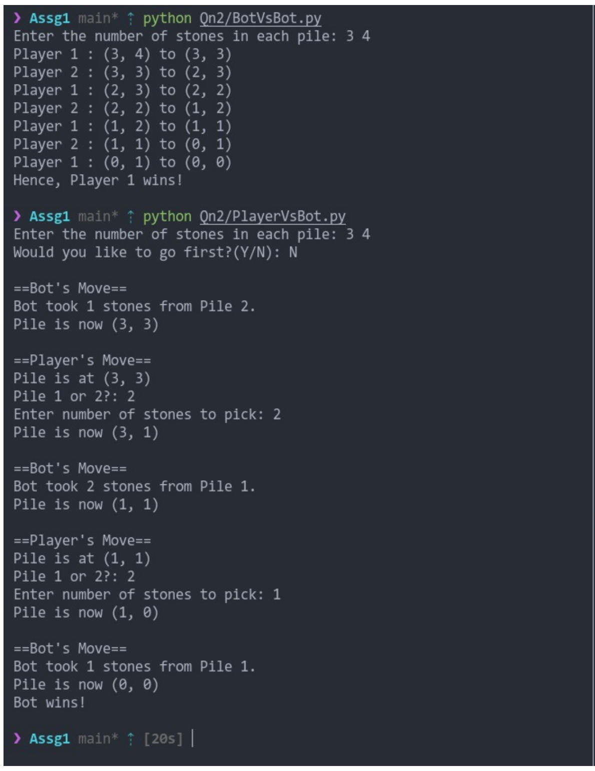


Solution 2

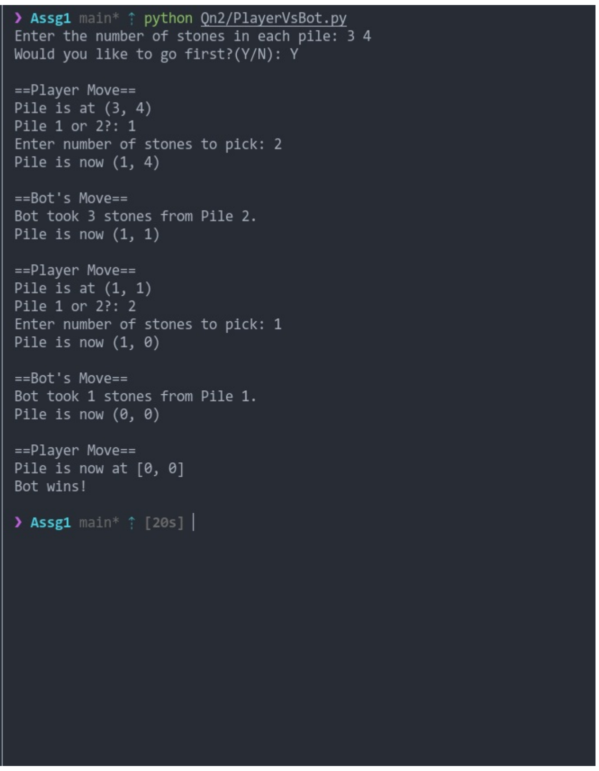
Here, I've used recursion for the minimax function. The algorithm stores the best paths for each player at each state in pathsTaken[]. In order for readability of code, I've split the Bot vs Bot and Player vs Bot options into two separate files. Both have the same minimax function but different main() functions.

In the first few outputs, I've taken small piles of stones in 3 cases - Bot vs Bot, Player vs Bot and Bot vs Player (in their respective orders.) The path taken by the winner is displayed in the correct sequence.

Output 1

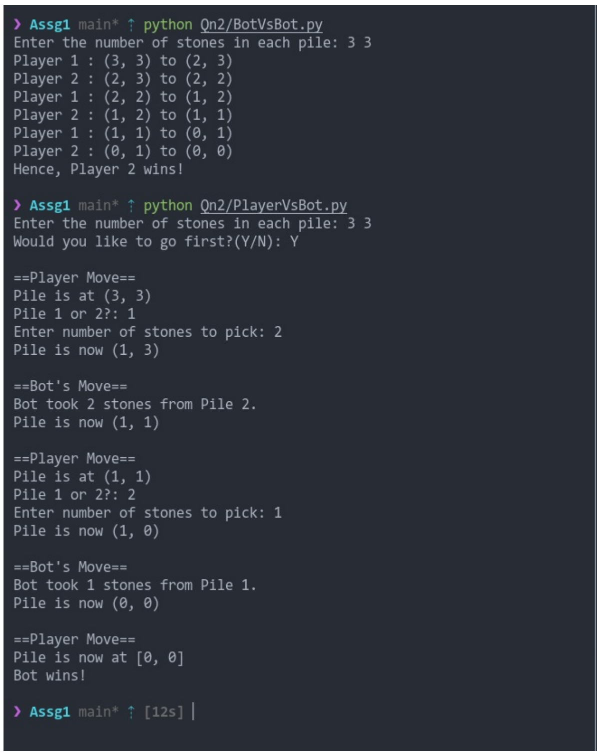


```
❯ Assg1 main* ⇡ python Qn2/BotVsBot.py
Enter the number of stones in each pile: 3 4
Player 1 : (3, 4) to (3, 3)
Player 2 : (3, 3) to (2, 3)
Player 1 : (2, 3) to (2, 2)
Player 2 : (2, 2) to (1, 2)
Player 1 : (1, 2) to (1, 1)
Player 2 : (1, 1) to (0, 1)
Player 1 : (0, 1) to (0, 0)
Hence, Player 1 wins!

❯ Assg1 main* ⇡ python Qn2/PlayerVsBot.py
Enter the number of stones in each pile: 3 4
Would you like to go first?(Y/N): N

==Bot's Move==
Bot took 1 stones from Pile 2.
Pile is now (3, 3)

==Player's Move==
Pile is at (3, 3)
Pile 1 or 2?: 2
Enter number of stones to pick: 2
Pile is now (3, 1)

==Bot's Move==
Bot took 2 stones from Pile 1.
Pile is now (1, 1)

==Player's Move==
Pile is at (1, 1)
Pile 1 or 2?: 2
Enter number of stones to pick: 1
Pile is now (1, 0)

==Bot's Move==
Bot took 1 stones from Pile 1.
Pile is now (0, 0)
Bot wins!

❯ Assg1 main* ⇡ [20s] |
```

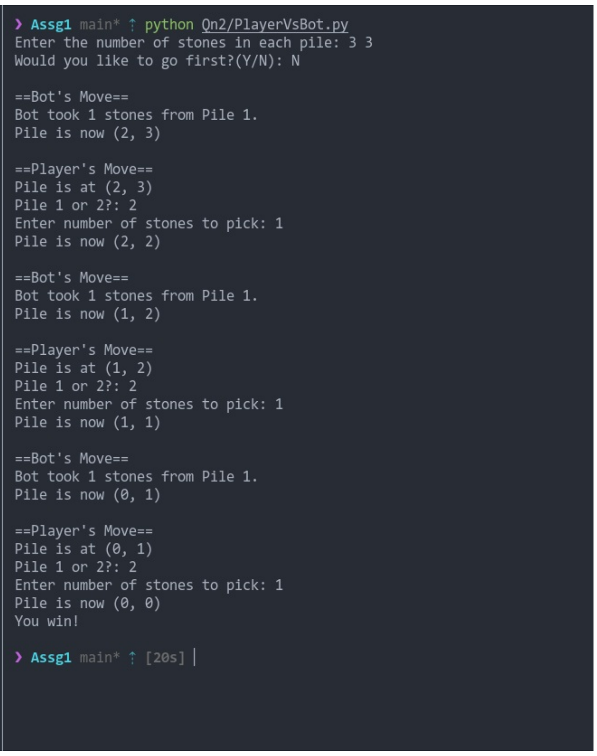


```
❯ Assg1 main* ⇡ python Qn2/PlayerVsBot.py
Enter the number of stones in each pile: 3 4
Would you like to go first?(Y/N): Y

==Player Move==
Pile is at (3, 4)
Pile 1 or 2?: 1
Enter number of stones to pick: 2
Pile is now (1, 4)

==Bot's Move==
Bot took 3 stones from Pile 2.
Pile is now (1, 1)

==Player Move==
Pile is at (1, 1)
Pile 1 or 2?: 2
Enter number of stones to pick: 1
Pile is now (1, 0)

==Bot's Move==
Bot took 1 stones from Pile 1.
Pile is now (0, 0)

==Player Move==
Pile is now at [0, 0]
Bot wins!

❯ Assg1 main* ⇡ [20s] |
```

Output 2

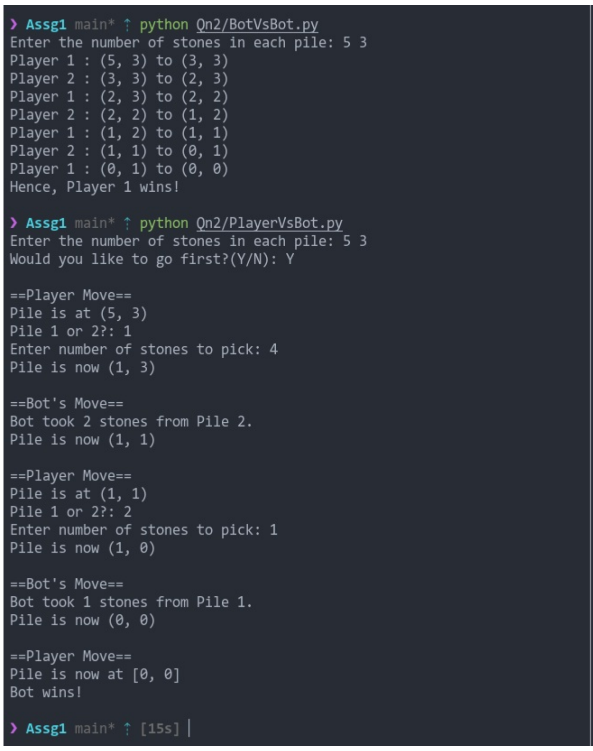


```
❯ Assg1 main* ⇡ python Qn2/BotVsBot.py
Enter the number of stones in each pile: 3 3
Player 1 : (3, 3) to (2, 3)
Player 2 : (2, 3) to (2, 2)
Player 1 : (2, 2) to (1, 2)
Player 2 : (1, 2) to (1, 1)
Player 1 : (1, 1) to (0, 1)
Player 2 : (0, 1) to (0, 0)
Hence, Player 2 wins!

❯ Assg1 main* ⇡ python Qn2/PlayerVsBot.py
Enter the number of stones in each pile: 3 3
Would you like to go first?(Y/N): Y

==Player Move==
Pile is at (3, 3)
Pile 1 or 2?: 1
Enter number of stones to pick: 2
Pile is now (1, 3)

==Bot's Move==
Bot took 2 stones from Pile 2.
Pile is now (1, 1)

==Player Move==
Pile is at (1, 1)
Pile 1 or 2?: 2
Enter number of stones to pick: 1
Pile is now (1, 0)

==Bot's Move==
Bot took 1 stones from Pile 1.
Pile is now (0, 0)

==Player Move==
Pile is now at [0, 0]
Bot wins!

❯ Assg1 main* ⇡ [12s] |
```

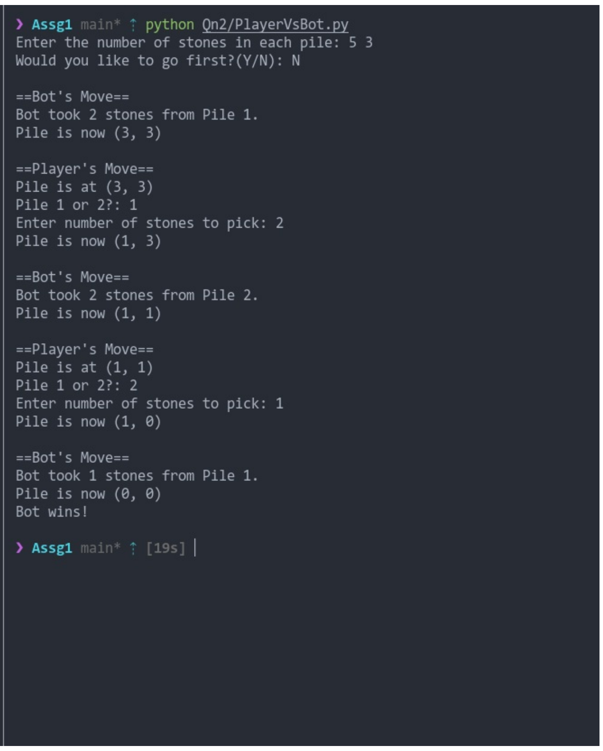


```
❯ Assg1 main* ⇡ python Qn2/PlayerVsBot.py
Enter the number of stones in each pile: 3 3
Would you like to go first?(Y/N): N

==Bot's Move==
Bot took 1 stones from Pile 1.
Pile is now (2, 3)

==Player's Move==
Pile is at (2, 3)
Pile 1 or 2?: 2
Enter number of stones to pick: 1
Pile is now (2, 2)

==Bot's Move==
Bot took 1 stones from Pile 1.
Pile is now (1, 2)

==Player's Move==
Pile is at (1, 2)
Pile 1 or 2?: 2
Enter number of stones to pick: 1
Pile is now (1, 1)

==Bot's Move==
Bot took 1 stones from Pile 1.
Pile is now (0, 1)

==Player's Move==
Pile is at (0, 1)
Pile 1 or 2?: 2
Enter number of stones to pick: 1
Pile is now (0, 0)
You win!

❯ Assg1 main* ⇡ [20s] |
```

Output 3



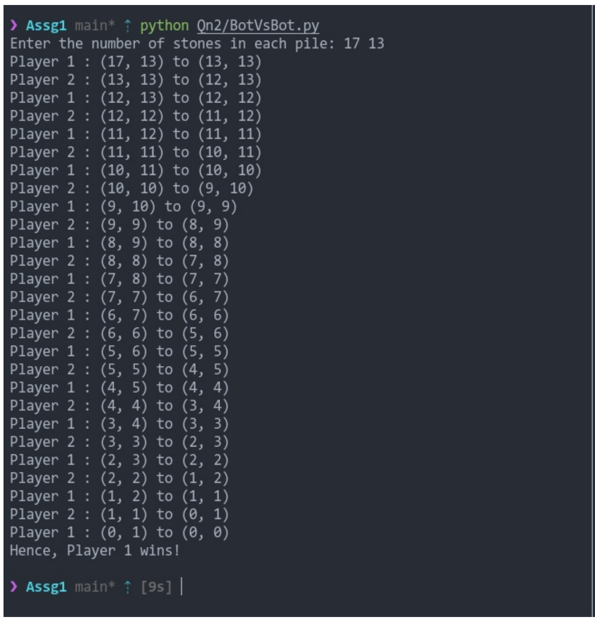
```
❯ Assg1 main* ⇡ python Qn2/BotVsBot.py
Enter the number of stones in each pile: 5 3
Player 1 : (5, 3) to (3, 3)
Player 2 : (3, 3) to (2, 3)
Player 1 : (2, 3) to (2, 2)
Player 2 : (2, 2) to (1, 2)
Player 1 : (1, 2) to (1, 1)
Player 2 : (1, 1) to (0, 1)
Player 1 : (0, 1) to (0, 0)
Hence, Player 1 wins!

❯ Assg1 main* ⇡ python Qn2/PlayerVsBot.py
Enter the number of stones in each pile: 5 3
Would you like to go first?(Y/N): Y

==Player Move==
Pile is at (5, 3)
Pile 1 or 2?: 1
Enter number of stones to pick: 4
Pile is now (1, 3)

==Bot's Move==
Bot took 2 stones from Pile 2.
Pile is now (1, 1)

==Player Move==
Pile is at (1, 1)
Pile 1 or 2?: 2
Enter number of stones to pick: 1
Pile is now (1, 0)

==Bot's Move==
Bot took 1 stones from Pile 1.
Pile is now (0, 0)

==Player Move==
Pile is now at [0, 0]
Bot wins!

❯ Assg1 main* ⇡ [15s] |
```



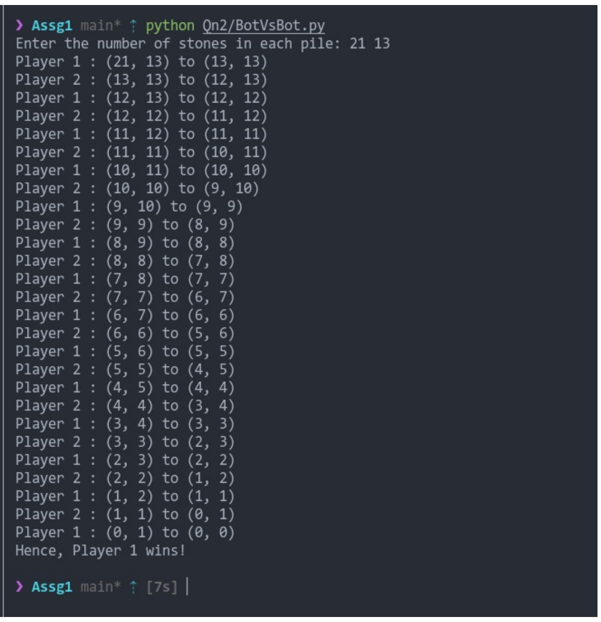
```
❯ Assg1 main* ⇡ python Qn2/PlayerVsBot.py
Enter the number of stones in each pile: 5 3
Would you like to go first?(Y/N): N

==Bot's Move==
Bot took 2 stones from Pile 1.
Pile is now (3, 3)

==Player's Move==
Pile is at (3, 3)
Pile 1 or 2?: 1
Enter number of stones to pick: 2
Pile is now (1, 3)

==Bot's Move==
Bot took 2 stones from Pile 2.
Pile is now (1, 1)

==Player's Move==
Pile is at (1, 1)
Pile 1 or 2?: 2
Enter number of stones to pick: 1
Pile is now (1, 0)

==Bot's Move==
Bot took 1 stones from Pile 1.
Pile is now (0, 0)
Bot wins!

❯ Assg1 main* ⇡ [19s] |
```

Now, due to the time complexity being exponential, I've used the functools.cache() wrapper around the minimax function to help with larger numbers. Here are 4 examples of big piles in a Bot vs Bot scenario.

Output 1

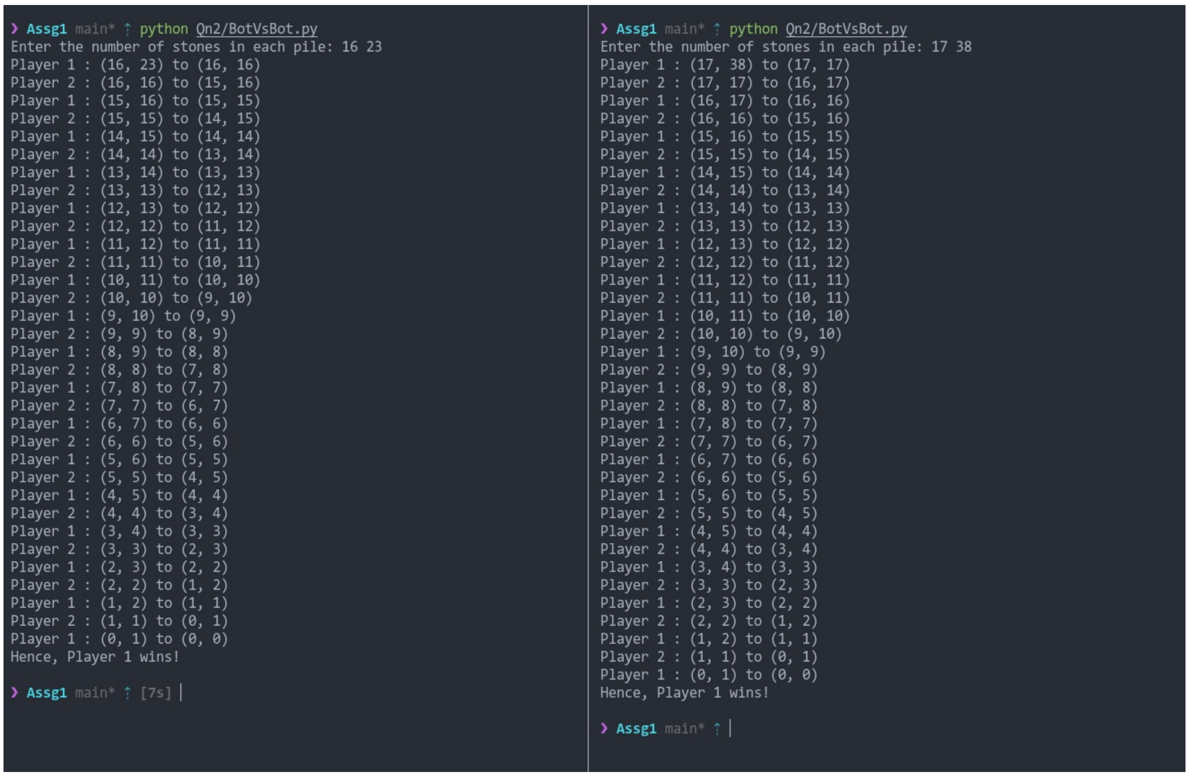


```
❯ Assg1 main* ⇡ python Qn2/BotVsBot.py
Enter the number of stones in each pile: 17 13
Player 1 : (17, 13) to (13, 13)
Player 2 : (13, 13) to (12, 13)
Player 1 : (12, 13) to (12, 12)
Player 2 : (12, 12) to (11, 12)
Player 1 : (11, 12) to (11, 11)
Player 2 : (11, 11) to (10, 11)
Player 1 : (10, 11) to (10, 10)
Player 2 : (10, 10) to (9, 10)
Player 1 : (9, 10) to (9, 9)
Player 2 : (9, 9) to (8, 9)
Player 1 : (8, 9) to (8, 8)
Player 2 : (8, 8) to (7, 8)
Player 1 : (7, 8) to (7, 7)
Player 2 : (7, 7) to (6, 7)
Player 1 : (6, 7) to (6, 6)
Player 2 : (6, 6) to (5, 6)
Player 1 : (5, 6) to (5, 5)
Player 2 : (5, 5) to (4, 5)
Player 1 : (4, 5) to (4, 4)
Player 2 : (4, 4) to (3, 4)
Player 1 : (3, 4) to (3, 3)
Player 2 : (3, 3) to (2, 3)
Player 1 : (2, 3) to (2, 2)
Player 2 : (2, 2) to (1, 2)
Player 1 : (1, 2) to (1, 1)
Player 2 : (1, 1) to (0, 1)
Player 1 : (0, 1) to (0, 0)
Hence, Player 1 wins!

❯ Assg1 main* ⇡ [9s] |
```



```
❯ Assg1 main* ⇡ python Qn2/BotVsBot.py
Enter the number of stones in each pile: 21 13
Player 1 : (21, 13) to (13, 13)
Player 2 : (13, 13) to (12, 13)
Player 1 : (12, 13) to (12, 12)
Player 2 : (12, 12) to (11, 12)
Player 1 : (11, 12) to (11, 11)
Player 2 : (11, 11) to (10, 11)
Player 1 : (10, 11) to (10, 10)
Player 2 : (10, 10) to (9, 10)
Player 1 : (9, 10) to (9, 9)
Player 2 : (9, 9) to (8, 9)
Player 1 : (8, 9) to (8, 8)
Player 2 : (8, 8) to (7, 8)
Player 1 : (7, 8) to (7, 7)
Player 2 : (7, 7) to (6, 7)
Player 1 : (6, 7) to (6, 6)
Player 2 : (6, 6) to (5, 6)
Player 1 : (5, 6) to (5, 5)
Player 2 : (5, 5) to (4, 5)
Player 1 : (4, 5) to (4, 4)
Player 2 : (4, 4) to (3, 4)
Player 1 : (3, 4) to (3, 3)
Player 2 : (3, 3) to (2, 3)
Player 1 : (2, 3) to (2, 2)
Player 2 : (2, 2) to (1, 2)
Player 1 : (1, 2) to (1, 1)
Player 2 : (1, 1) to (0, 1)
Player 1 : (0, 1) to (0, 0)
Hence, Player 1 wins!

❯ Assg1 main* ⇡ [7s] |
```

Output 2



```
❯ Assg1 main* ⇡ python Qn2/BotVsBot.py
Enter the number of stones in each pile: 16 23
Player 1 : (16, 23) to (16, 16)
Player 2 : (16, 16) to (15, 16)
Player 1 : (15, 16) to (15, 15)
Player 2 : (15, 15) to (14, 15)
Player 1 : (14, 15) to (14, 14)
Player 2 : (14, 14) to (13, 14)
Player 1 : (13, 14) to (13, 13)
Player 2 : (13, 13) to (12, 13)
Player 1 : (12, 13) to (12, 12)
Player 2 : (12, 12) to (11, 12)
Player 1 : (11, 12) to (11, 11)
Player 2 : (11, 11) to (10, 11)
Player 1 : (10, 11) to (10, 10)
Player 2 : (10, 10) to (9, 10)
Player 1 : (9, 10) to (9, 9)
Player 2 : (9, 9) to (8, 9)
Player 1 : (8, 9) to (8, 8)
Player 2 : (8, 8) to (7, 8)
Player 1 : (7, 8) to (7, 7)
Player 2 : (7, 7) to (6, 7)
Player 1 : (6, 7) to (6, 6)
Player 2 : (6, 6) to (5, 6)
Player 1 : (5, 6) to (5, 5)
Player 2 : (5, 5) to (4, 5)
Player 1 : (4, 5) to (4, 4)
Player 2 : (4, 4) to (3, 4)
Player 1 : (3, 4) to (3, 3)
Player 2 : (3, 3) to (2, 3)
Player 1 : (2, 3) to (2, 2)
Player 2 : (2, 2) to (1, 2)
Player 1 : (1, 2) to (1, 1)
Player 2 : (1, 1) to (0, 1)
Player 1 : (0, 1) to (0, 0)
Hence, Player 1 wins!

❯ Assg1 main* ⇡ [7s]
```

```
❯ Assg1 main* ⇡ python Qn2/BotVsBot.py
Enter the number of stones in each pile: 17 38
Player 1 : (17, 38) to (17, 17)
Player 2 : (17, 17) to (16, 17)
Player 1 : (16, 17) to (16, 16)
Player 2 : (16, 16) to (15, 16)
Player 1 : (15, 16) to (15, 15)
Player 2 : (15, 15) to (14, 15)
Player 1 : (14, 15) to (14, 14)
Player 2 : (14, 14) to (13, 14)
Player 1 : (13, 14) to (13, 13)
Player 2 : (13, 13) to (12, 13)
Player 1 : (12, 13) to (12, 12)
Player 2 : (12, 12) to (11, 12)
Player 1 : (11, 12) to (11, 11)
Player 2 : (11, 11) to (10, 11)
Player 1 : (10, 11) to (10, 10)
Player 2 : (10, 10) to (9, 10)
Player 1 : (9, 10) to (9, 9)
Player 2 : (9, 9) to (8, 9)
Player 1 : (8, 9) to (8, 8)
Player 2 : (8, 8) to (7, 8)
Player 1 : (7, 8) to (7, 7)
Player 2 : (7, 7) to (6, 7)
Player 1 : (6, 7) to (6, 6)
Player 2 : (6, 6) to (5, 6)
Player 1 : (5, 6) to (5, 5)
Player 2 : (5, 5) to (4, 5)
Player 1 : (4, 5) to (4, 4)
Player 2 : (4, 4) to (3, 4)
Player 1 : (3, 4) to (3, 3)
Player 2 : (3, 3) to (2, 3)
Player 1 : (2, 3) to (2, 2)
Player 2 : (2, 2) to (1, 2)
Player 1 : (1, 2) to (1, 1)
Player 2 : (1, 1) to (0, 1)
Player 1 : (0, 1) to (0, 0)
Hence, Player 1 wins!

❯ Assg1 main* ⇡
```