# PS5

Graded

**Student**

Chetan Hiremath

**Total Points**

**98 / 100 pts**

**Question 1**

**(no title)** **10** / 10 pts

✔ **+ 10 pts** all correct

**Question 2**

**(no title)** **20** / 20 pts

✔ **+ 20 pts** all correct

**Question 3**

**(no title)** **Resolved** **28** / 30 pts

✔ **+ 30 pts** correct

✔ **− 2 pts** Part A secondary observation: most games take less than 9 turns, inaccessible states, symmetry

Regrade Request **Submitted on: Mar 04**

How I have lost 2 points on Question 3? I don't understand your comment. Will you regrade Question 3 and let me know?

the number of games is less than 9! due to symmetry or inaccessible states.

Reviewed on: Mar 05

**Question 4**

**(no title)** **40** / 40 pts

✔ **+ 40 pts** Correct

Question assigned to the following page: 1

1. Monopoly:

State Descriptions- Monopoly has the board layout, players' pieces/positions on the board, players' properties, players' money, the states of chance and community chest cards, and other board game elements like houses and hotels.

Move Generators- Monopoly will allow players to roll the dice that will determine their next moves, so they will use many ways of moving their board pieces to properties and selling and buying properties.

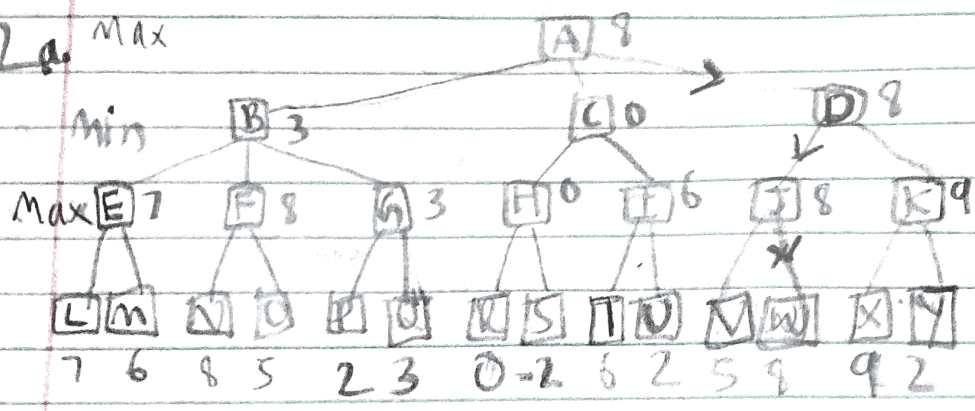
Terminal Tests- Monopoly ends when the last player with the money remains, and the other players are bankrupt.

Utility Functions- Monopoly mainly focuses on players' financial assts and properties. If players land on properties that are not owned, then they buy these properties. Players can provide high offers when they land on other players' properties. They use properties for future use due to bargaining.

Evaluation Functions- Monopoly has board positions that are complex, but these positions estimate properties' values and income from rents and check the potential to build houses and hotels.

Question assigned to the following page: 2

2a.




Player 1 should make the move to Node D, and the minimum payoff is 8.

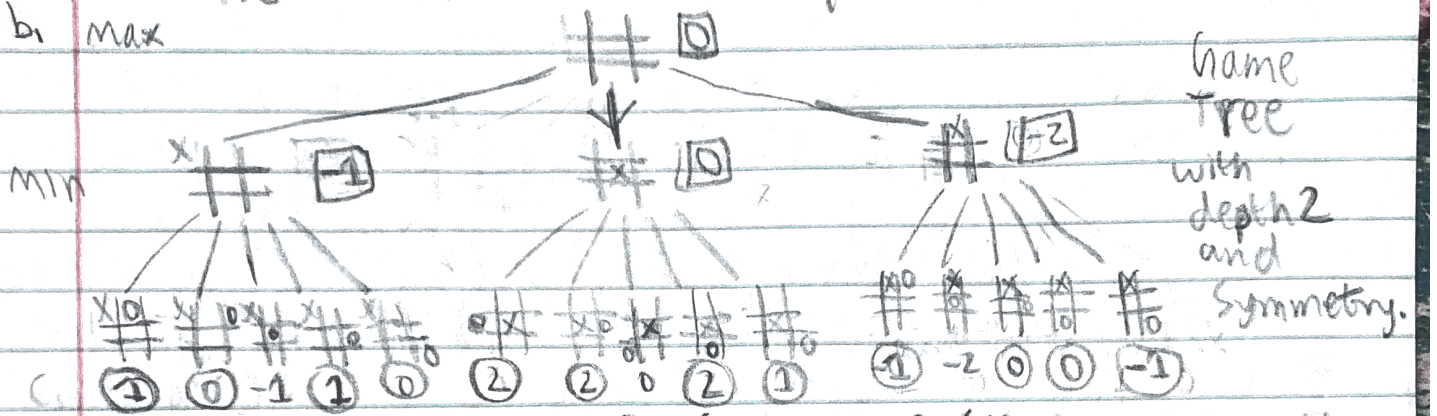
Question assigned to the following page: 2

b. max


The nodes that are not expanded and examined in left-to-right order are O, I, J, U, and Y.

Question assigned to the following page: 3

3a. There are approximately 9! or 362880 possible games of tic-tac toe since there are 9 possible moves of placing the first mark on the square.

b. 


c. $Eval(s) = 3X_2(s) + X_1(s) - (3O_2(s) + O_1(s))$ is used on the game tree.

d. Squares represent the values for minimax algorithm. Place X at the center square for the best move.

e. Circles represent the nodes for alpha-beta pruning. These nodes aren't evaluated if this algorithm is applied. The circled nodes are pruned since they don't affect the final value. You get the same value by minimax and alpha-beta pruning.

Question assigned to the following page: 4

4a.
Sources- This code is modified, used, and borrowed from PS 4 Code Solution, R&N Textbook's Figure 6.9's Pseudo-Code, 8queens.pynb's Python code, and Aimacode Algorithms' Python Code Repository.

Min-Conflict General Implementation's Python Code-

```
import numpy as np
import random

#Return the number of queens conflicting in grid with global variables.
def fitness(queens, N):
    global total_f
    global f
    f += 1
    total_f += 1
    c = 0
    for q in range(len(queens)):
        for q_y in range(len(queens)):
            q_x = queens[q_y]
            if q_y != q: #Check if queens are not attacking each other.
                if q_x == queens[q]:
                    c += 1
                if abs(q - q_y) == abs(queens[q] - q_x):
                    c += 1
    return (N*(N-1))/2 – c
```

Question assigned to the following page: 4

```
#Choose the successor by the Min-Conflict algorithm.
def successor(queens, N):
    global rand_choice
    global last_q
    if rand_choice: #Pick a choice queen randomly.
        q = random.choice(queens)
    else: #Pick a choice queen in the Cyclic order.
        last_q = (last_q + 1) % N
        q = last_q
    successors = [] #Store the Successor states for queen q.
    for i in range(N):
        qc = queens.copy()
        qc[q] = i
        successors.append(qc)
    best_f = -1
    for s in successors:
        f = fitness(s,N)
        if f > best_f:
            best_s = s #Update the best successors.
            best_f = f #Update the best fitness evaluations.
    return best_s

#Code Parameters
success = False
x = 0 #The iteration of the evaluation function.
total_f = 0 #The total number of fitness evaluations.
f = 0 #The number of evaluations per attempt.
s_total_f = 0 #The successful number of evaluations.
```

Question assigned to the following page: 4

```
#Use the while loop to minimize the conflict and print the final results.
while x < 1:
    rand_choice = False #Choose random queen to minimize conflict if it’s
true. If it's false, then choose cyclic order.
    last_q = -1
    N = 8 #Grid size or number of queens.
    i = 0 #Evaluation count.
    queens = []
    for n in range(N):
        queens.append(random.randint(0,N-1))
    for i in range(25):
        queens = successor(queens,N)
        #If there is a solution, then it breaks the loop and prints the
solution and results.
        if fitness(queens,N) == (N*(N-1))/2:
            print(queens)
            success = True
            total_f += i
            break
    if success == True:
        success = False
        x+=1
print('Total number of fitness evaluations:',total_f)
print('Average:',(total_f/100))
```

Question assigned to the following page: 4

Result-

```
[5, 3, 6, 0, 7, 1, 4, 2]
Total number of fitness evaluations: 1664
Average: 16.64
```

b. I have tested this implementation for several times, and I am getting the appropriate solutions randomly because I am using the random library. This part is very similar to the second part from PS 4 Question 5.

c. I have gathered statistics by running it for 100 times and calculating the average of the number of fitness evaluations after the solution is found. If I run the Python code to generate results for the solutions, then the results will be random due to the random library. The results will manage to find the solutions as strings of digits, their respective fitness evaluations' counts, and their respective fitness evaluations' counts' averages. This part is very similar to the third and the last part from PS 4 Question 5. The table shows the comparison of the results of this problem set/PS 5 and the previous problem set/PS 4 since the Python code from the previous problem set is used and modified in this problem set for the min-conflict algorithm. The values that are obtained in PS 5 are less than the values that are obtained in PS 4 due to the min-conflict algorithm. The table's results prove that the min-conflict has amortized time and effort because it minimizes the number of conflicts and solves the 8-queens' problem with very few iterations and fitness evaluations.

| Problem Set | Total Number of Fitness Evaluations | Average |
|---|---|---|
| PS 4 | 353644 | 3536.44 |
| PS 5 | 1664 | 16.64 |