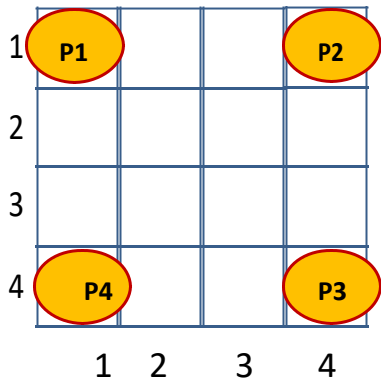
**The University of Texas at Dallas**
**CS 6364**
**Artificial Intelligence**
**Fall 2020**
**Instructor: Dr. Sanda Harabagiu**
**Grader/ Teaching Assistant: Maxwell Weinzierl**

**Homework 2: 100 points (20 points extra-credit)**
**Issued September 22, 2020**
**Due October 7, 2020 before midnight**
*Submit only in eLearning*

Name _____KAPIL GAUTAM_____
Student ID _____KXG180032_____
CS 6364

**PROBLEM 1:** Multi-player Games (**60 points**)
Four players are playing a game which is illustrated in the following Figure:




Player 1 is initially positioned in square [1,1], Player 2 is positioned in square [4,1], Player 3 is positioned in square [4,4] and Player 4 is positioned in square [1,4]. The player that will reach first the diagonally opposite position will win the game. Player 1 will start the game, followed by Player 2, Player 3 and Player 4.

All Players can move either up, down, left or right, if the square in that direction is available and not occupied by any other player.

For example, Player 1 initially can move only to the right or down.

**Question 1:** How do you represent each node of the game? (**3 points**) How is the initial node of the game represented? (**1 point**)

**Solution Question 1:**

The node is represented as the coordinates of the players along with some other details as the current_player, action, action name, game state with coordinates of the players, and next player.
It is represented as :
(curr_player, action, action_name, state, next_player)

Initial state is represented as:
(None, None, "", self.get_compact_state(self.players), self.players[0])

- where self.players is the list of all the players.
- self.players[0] is the first player
- self.get_compact_state(self.players) extract the coordinates of all the players in the current state

*Programming Assignment for Problem 1:*

A. Write code that generates the game tree for this multi-player game. (**15 points**)
The output should use the following format:

[*Current player* =???| Father node (if not initial node) =???| Action= ????| Current game node =???| if the game node is repeated, write REPEATED; if the current game node corresponds to a winning situation for one of the players, write WINS[PLAYER???]]

Hint: Successors of repeated game nodes should not be considered!

**Solution Question 1:**

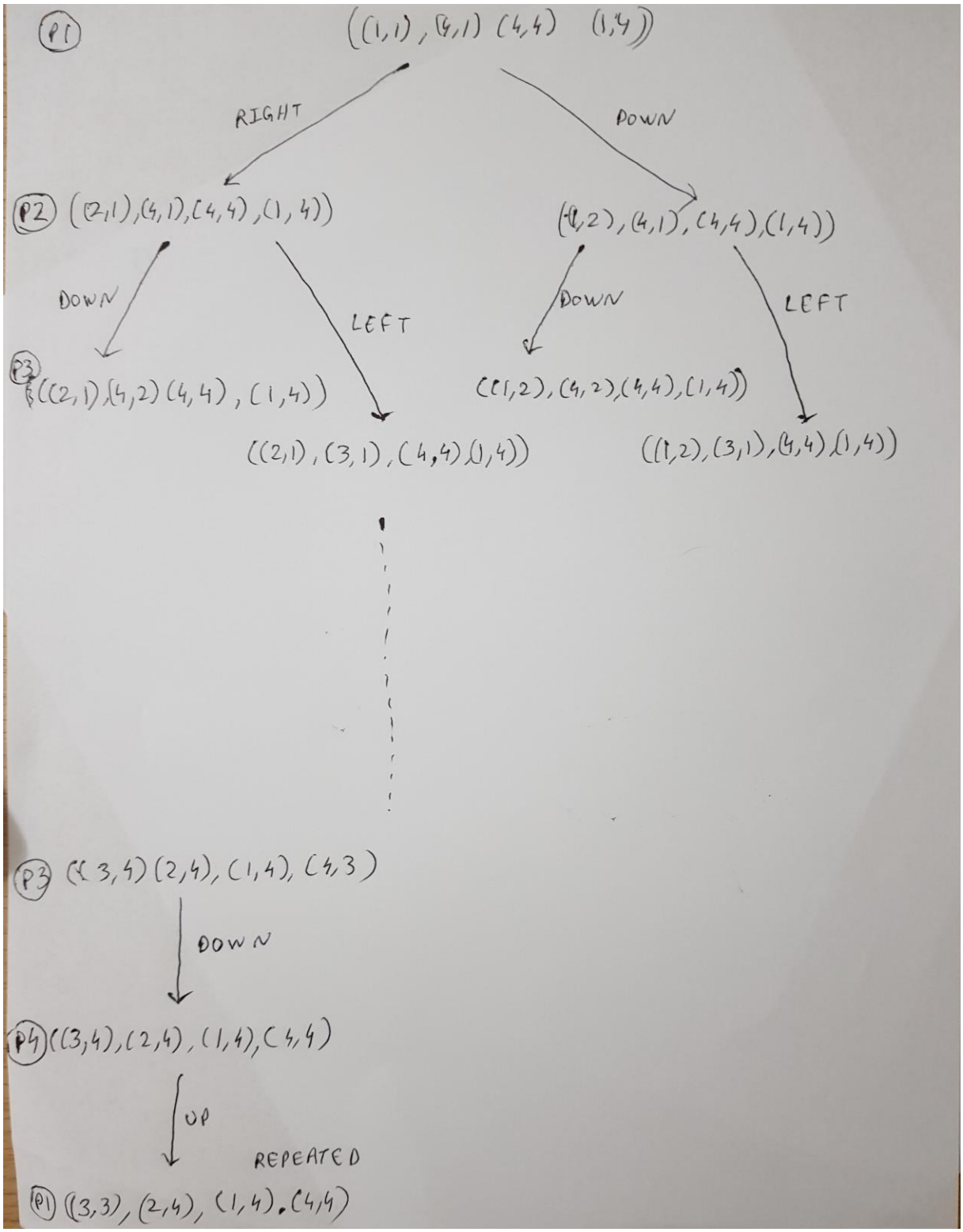
Check game_iterative.py file for the code for this problem.

**Question 2:** Draw the game tree based on the output of your code. (**11 points**)
Note: You can draw the game tree on a piece of paper, take a picture and attach it to your answers, which will be returned in a PDF file.

**Solution Question 2:**

Some of the winning states of the game are mentioned below:

- [Current player=P1 | Father node=((4, 3), (4, 2), (4, 1), (3, 1)) | Action=DOWN | Current game node=((4, 4), (4, 2), (4, 1), (3, 1)) | WINS=[PLAYER P1]]
- [Current player=P1 | Father node=((4, 3), (3, 3), (4, 1), (2, 2)) | Action=DOWN | Current game node=((4, 4), (3, 3), (4, 1), (2, 2)) | WINS=[PLAYER P1]]
- [Current player=P2 | Father node=((4, 2), (2, 4), (4, 3), (3, 1)) | Action=LEFT | Current game node=((4, 2), (1, 4), (4, 3), (3, 1)) | WINS=[PLAYER P2]]
- [Current player=P2 | Father node=((3, 3), (2, 4), (3, 4), (2, 2)) | Action=LEFT | Current game node=((3, 3), (1, 4), (3, 4), (2, 2)) | WINS=[PLAYER P2]]
- [Current player=P2 | Father node=((2, 2), (2, 4), (3, 4), (1, 1)) | Action=LEFT | Current game node=((2, 2), (1, 4), (3, 4), (1, 1)) | WINS=[PLAYER P2]]
- [Current player=P3 | Father node=((3, 1), (4, 3), (2, 1), (1, 3)) | Action=LEFT | Current game node=((3, 1), (4, 3), (1, 1), (1, 3)) | WINS=[PLAYER P3]]
- [Current player=P3 | Father node=((4, 2), (3, 2), (2, 1), (1, 3)) | Action=LEFT | Current game node=((4, 2), (3, 2), (1, 1), (1, 3)) | WINS=[PLAYER P3]]
- [Current player=P4 | Father node=((4, 2), (4, 3), (2, 2), (3, 1)) | Action=RIGHT | Current game node=((4, 2), (4, 3), (2, 2), (4, 1)) | WINS=[PLAYER P4]]
- [Current player=P4 | Father node=((3, 3), (3, 2), (4, 2), (3, 1)) | Action=RIGHT | Current game node=((3, 3), (3, 2), (4, 2), (4, 1)) | WINS=[PLAYER P4]]




**Question 3:** If Player 1 wins, the utility should be 200. If Player 2 wins, the utility should be 300. If Player 3 wins, the utility should be 400 and if Player 4 wins, the Utility should be 500. In any case, because this is not a zero-sum game, the other players also receive utility values:

- When Player 1 wins, Player 2 receives utility=10, Player 3 receives utility=30, and Player 4 receives utility=10;
- When Player 2 wins, Player 1 receives utility=100, Player 3 receives utility=150, and Player 4 receives utility=200;
- When Player 3 wins, Player 1 receives utility=150, Player 2 receives utility=200, and Player 4 receives utility=300;
- When Player 4 wins, Player 1 receives utility=220, Player 2 receives utility=330, and Player 3 receives utility=440;

If the minimax values of the repeated states shall be ignored, compute the MINIMAX values in all other states of the game, using the following program assignment.

*Programming Assignment for Problem 1:* (**15 points**)

B. Write code that computes the MINIMAX values for all non-repeated game nodes and display the values of MINIMAX in the following format:

[*Current player* = …| Father node (if not initial node) = …| Action= ????| Current game node =…| if the current game node corresponds to a winning situation for one of the players, write WINS[PLAYER???]| MINIMAX = ?????]

**Solution Question 3:**

Check minimax_game.py file for the code for this problem.

**Question 4:** Place the MINIMAX values for each non-repeated game state in the drawing of the game tree based on the output of your code. (**15 points**)

**Solution Question 4:**

Some of the winning states of the game are mentioned below:

- Current player=P1 | Father node=((4, 3), (4, 2), (4, 1), (3, 1)) | Action=DOWN | Current game node=((4, 4), (4, 2), (4, 1), (3, 1)) | WINS=[PLAYER P1] | MINIMAX=[200, 10, 30, 10]]
- Current player=P1 | Father node=((4, 3), (3, 3), (3, 2), (1, 1)) | Action=DOWN | Current game node=((4, 4), (3, 3), (3, 2), (1, 1)) | WINS=[PLAYER P1] | MINIMAX=[200, 10, 30, 10]]
- Current player=P1 | Father node=((4, 3), (2, 2), (3, 2), (1, 1)) | Action=DOWN | Current game node=((4, 4), (2, 2), (3, 2), (1, 1)) | WINS=[PLAYER P1] | MINIMAX=[200, 10, 30, 10]]

- Current player=P2 | Father node=((4, 2), (2, 4), (4, 3), (3, 1)) | Action=LEFT | Current game node=((4, 2), (1, 4), (4, 3), (3, 1)) | WINS=[PLAYER P2] | MINIMAX=[100, 300, 150, 200]]

- Current player=P2 | Father node=((4, 2), (2, 4), (3, 4), (1, 1)) | Action=LEFT | Current game node=((4, 2), (1, 4), (3, 4), (1, 1)) | WINS=[PLAYER P2] | MINIMAX=[100, 300, 150, 200]]

- Current player=P2 | Father node=((2, 2), (2, 4), (4, 3), (1, 1)) | Action=LEFT | Current game node=((2, 2), (1, 4), (4, 3), (1, 1)) | WINS=[PLAYER P2] | MINIMAX=[100, 300, 150, 200]]

- Current player=P3 | Father node=((3, 1), (3, 2), (2, 1), (2, 2)) | Action=LEFT | Current game node=((3, 1), (3, 2), (1, 1), (2, 2)) | WINS=[PLAYER P3] | MINIMAX=[150, 200, 400, 300]]

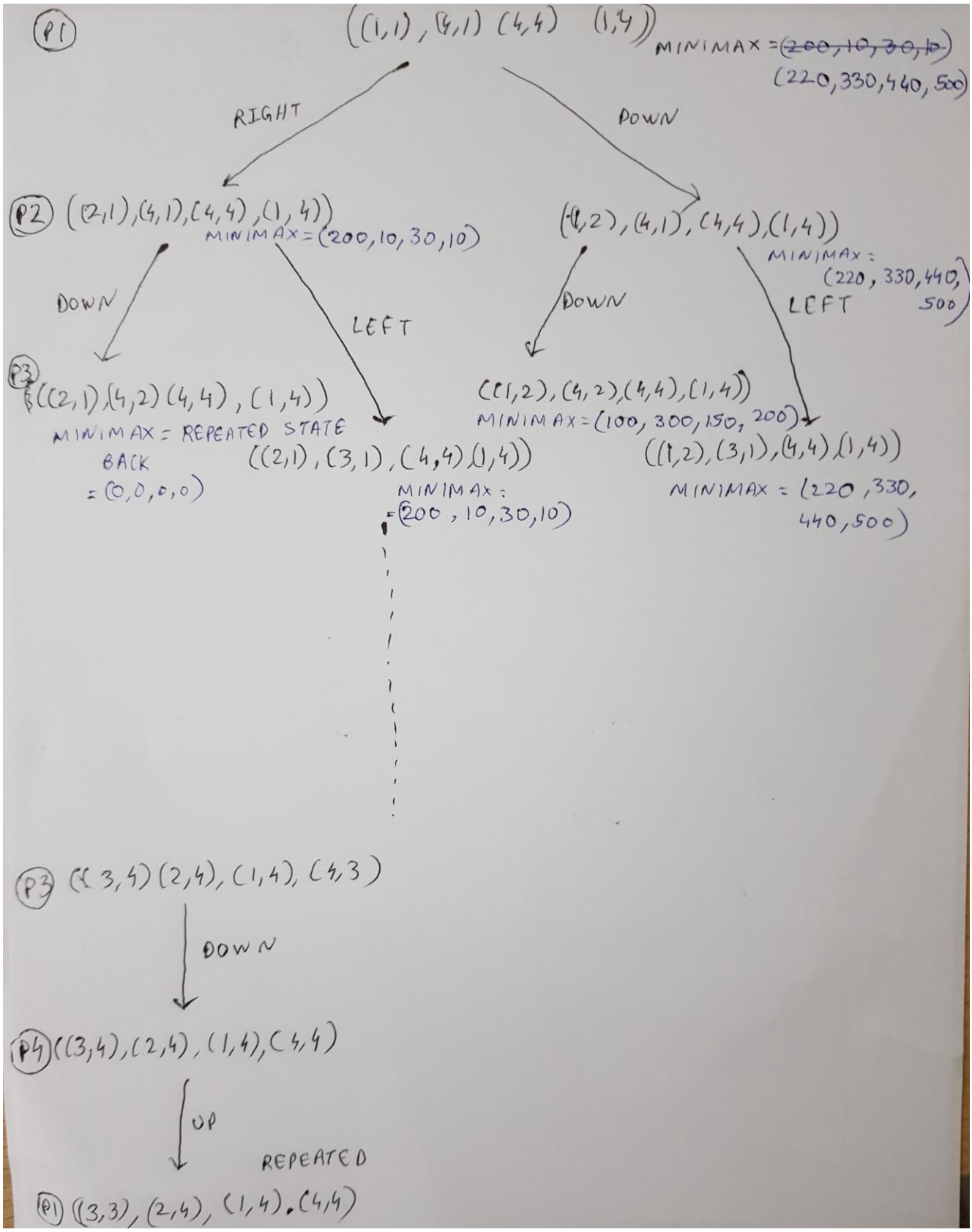
- Current player=P3 | Father node=((4, 2), (3, 4), (2, 1), (2, 2)) | Action=LEFT | Current game node=((4, 2), (3, 4), (1, 1), (2, 2)) | WINS=[PLAYER P3] | MINIMAX=[150, 200, 400, 300]]

- Current player=P3 | Father node=((3, 1), (1, 2), (2, 1), (1, 3)) | Action=LEFT | Current game node=((3, 1), (1, 2), (1, 1), (1, 3)) | WINS=[PLAYER P3] | MINIMAX=[150, 200, 400, 300]]

- Current player=P4 | Father node=((4, 2), (4, 3), (3, 3), (3, 1)) | Action=RIGHT | Current game node=((4, 2), (4, 3), (3, 3), (4, 1)) | WINS=[PLAYER P4] | MINIMAX=[220, 330, 440, 500]]

- Current player=P4 | Father node=((4, 2), (1, 2), (2, 2), (3, 1)) | Action=RIGHT | Current game node=((4, 2), (1, 2), (2, 2), (4, 1)) | WINS=[PLAYER P4] | MINIMAX=[220, 330, 440, 500]]

- Current player=P4 | Father node=((3, 3), (1, 2), (4, 2), (3, 1)) | Action=RIGHT | Current game node=((3, 3), (1, 2), (4, 2), (4, 1)) | WINS=[PLAYER P4] | MINIMAX=[220, 330, 440, 500]]


P1
((1,1),(4,1),(4,4),(1,4))
MINIMAX = (200,10,30,10) (220,330,440,500)
RIGHT
DOWN
P2
((2,1),(4,1),(4,4),(1,4))
MINIMAX = (200,10,30,10)
((1,2),(4,1),(4,4),(1,4))
MINIMAX = (220,330,440,500)
DOWN
LEFT
DOWN
LEFT
P3
((2,1),(4,2),(4,4),(1,4))
MINIMAX = REPEATED STATE BACK = (0,0,0,0)
((2,1),(3,1),(4,4),(1,4))
MINIMAX = (200,10,30,10)
((1,2),(4,2),(4,4),(1,4))
MINIMAX = (100,300,150,200)
((1,2),(3,1),(4,4),(1,4))
MINIMAX = (220,330,440,500)
P3
((3,4),(2,4),(1,4),(4,3))
DOWN
P4
((3,4),(2,4),(1,4),(4,4))
UP
REPEATED
P1
((3,3),(2,4),(1,4),(4,4))

*Extra-credit:*

*Programming Assignment for Problem 1:*

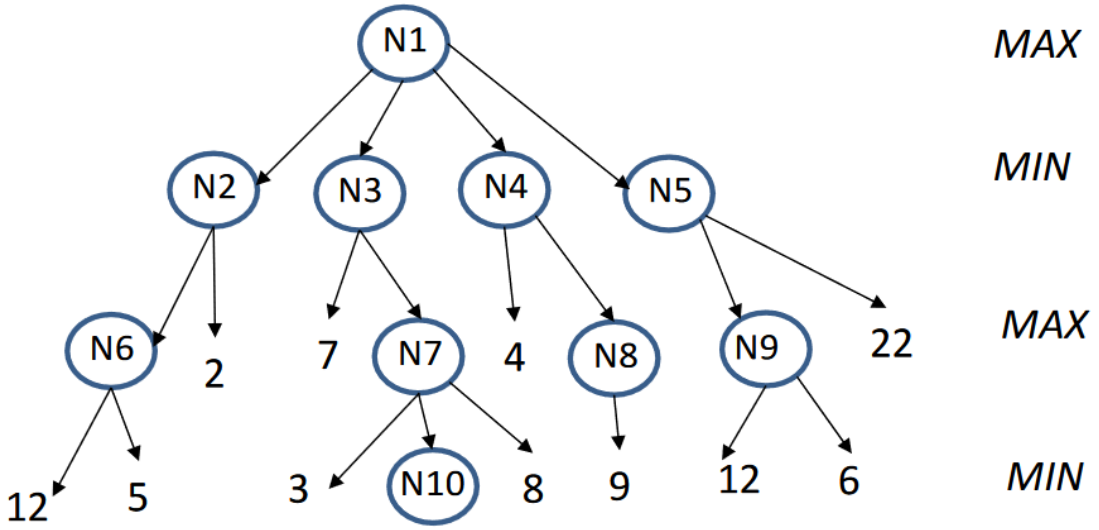
C. If you allow an additional action for the game, namely that any player can jump over an occupied square, and represent this new action as : Jump+Down, Jump+Left, Jump+Right or Jump+Down, modify your program to generate the new game tree.

Produce the output in the same format as when doing assignment A. (**10 points**)
Comment on the difference between the game trees: Which player won faster in which variant of the game??? Are the repeated states any different? (**10 points**)

**PROBLEM 2:** Alpha-Beta Search (**26 points**)

---

Find the move ordering (killer move) that allows you to prune the largest number of subtrees when using alpha-beta pruning on the following game tree:




You will receive **10 points** if you find the killer move and explain why it is a killer move. You should produce a trace of alpha-beta pruning using the format showing how the killer move operates (**10 points)**:

| | | | |
|---|---|---|---|
| **N1:** | alpha = -∞ | beta = +∞ | maxvalue=??? |
| **NX??:** | alpha = ?? | beta = ?? | minvalue=??? |
| **NY:** | alpha = ??? | beta = ?? | maxvalue=?? |
| **???:** | alpha = ??? | beta = ?? | minvalue=?? |
| **….** | | | |

Indicate on the Search Tree which subtrees shall be pruned and if it is alpha or beta pruning (**6 points)** when you indicate correctly which subtrees shall be pruned.

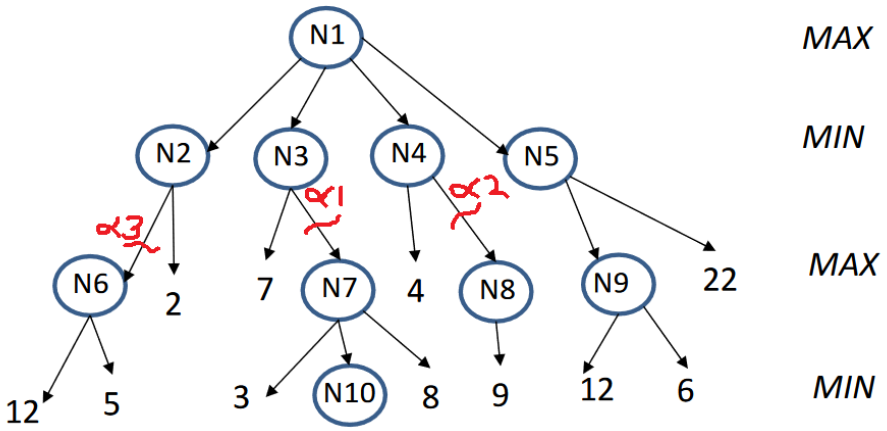
**SOLUTION 2:**

The move ordering or the killer move would be to go in following order:

a.) N5 (utility – 12)
b.) N3 (utility - 7)
c.) N4 (utility – 4)
d.) N2 (utility - 2)

The below given trace is of a killer move because it saves up on time by avoiding the inspection of following nodes by α pruning:

- α1 cut avoids (N7, N10, 3, 8)
- α2 cut avoids (N8, 9)
- α3 cut avoids (N6, 12, 5)

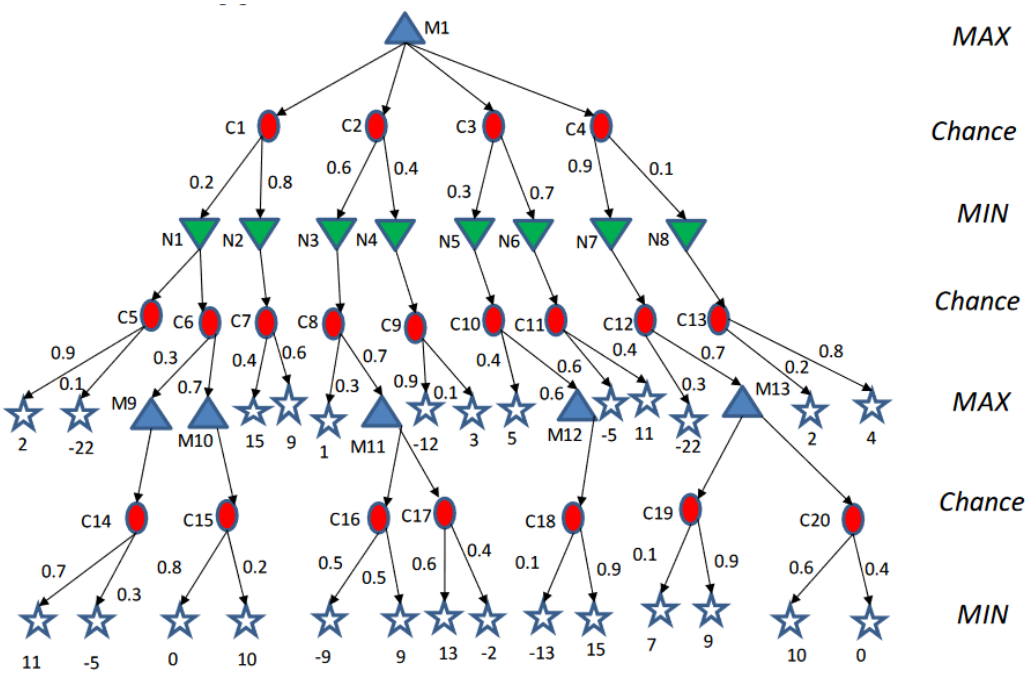



Below table describes how the killer move operates:

| Node | α value | β value | Utility |
|---|---|---|---|
| N1 | alpha = -∞ | beta = +∞ | maxvalue = -∞ |
| N5 | alpha = -∞ | beta = +∞ | minvalue = +∞ |
| N9 | alpha = -∞ | beta = +∞ | maxvalue = -∞ |
| 12 | | | |
| N9 | alpha = 12 | beta = +∞ | maxvalue = 12 |
| 6 | | | |
| N9 | alpha = 12 | beta = +∞ | maxvalue = 12 |
| N5 | alpha = -∞ | beta = 12 | minvalue = 12 |
| 22 | | | |
| N5 | alpha = -∞ | beta = 12 | minvalue = 12 |
| N1 | alpha = 12 | beta = +∞ | maxvalue = 12 |
| N3 | alpha = 12 | beta = +∞ | minvalue = +∞ |
| 7 | 7 | 7 | |
| N3 | alpha = 12 | beta = +∞ | minvalue = 7<br>VALUE <= 12<br>α1 CUT |
| N1 | alpha = 12 | beta = +∞ | maxvalue = 12 |
| N4 | alpha = 12 | beta = +∞ | minvalue = +∞ |

| | | | |
|---|---|---|---|
| 4 | 4 | 4 | |
| N4 | alpha = 12 | beta = +∞ | minvalue = 4<br>VALUE <= 12<br>α2 CUT |
| N1 | alpha = 12 | beta = +∞ | maxvalue = 12 |
| N2 | alpha = 12 | beta = +∞ | minvalue = +∞ |
| 2 | 2 | 2 | |
| N2 | alpha = 12 | beta = +∞ | minvalue = 2<br>VALUE <= 12<br>α3 CUT |
| N1 | alpha = 12 | beta = +∞ | maxvalue = 12 |

PROBLEM 3: Chance Games (14 points)
Given the following game tree:




Compute the *Expectiminimax* value in the following nodes, making sure you show how you computed the value:

(**1 point**) In M1:
(**1 point**) In M9:
(**1 point**) In M10:
(**1 point**) In M11:
(**1 point**) In M12:
(**1 point**) In M13:
(**1 point**) In N1:
(**1 point**) In N2:

(**1 point**) In N3:
(**1 point**) In N4:
(**1 point**) In N5:
(**1 point**) In N6:
(**1 point**) In N7:
(**1 point**) In N8:

**SOLUTION 3:**
PS: M1 solved in the last

(**1 point**) In M9:
Expectiminimax (M9) = Expectiminimax (C14) = 0.7*11 + 0.3*(-5) = 7.7 – 1.5 = 6.2
Expectiminimax (M9) = 6.2

(**1 point**) In M10:
Expectiminimax (M10) = Expectiminimax (C15) = 0.8*0 + 0.2*10 = 0 + 2.0 = 2.0
Expectiminimax (M10) = 2.0

(**1 point**) In M11:
Expectiminimax (M11) = MAX (Expectiminimax (C16), Expectiminimax (C17))
= MAX (0.5*(-9) + 0.5*9, 0.6*13 + 0.4 * -2)
= MAX (0, 7.8 – 0.8)
= MAX (0, 7.0) = 7.0
Expectiminimax (M11) = 7.0

(**1 point**) In M12:
Expectiminimax (M12) = Expectiminimax (C18) = 0.1*(-13) + 0.9*15 = -1.3 + 13.5 = 12.2
Expectiminimax (M12) = 12.2

(**1 point**) In M13:
Expectiminimax (M13) = MAX (Expectiminimax (C19), Expectiminimax (C20))
= MAX (0.1*7 + 0.9*9, 0.6*10 + 0.4*0)
= MAX (0.7+8.1, 6)
= MAX (8.8, 6) = 8.8
MINIMAX(M13) = 8.8

(**1 point**) In N1:
Expectiminimax (N1) = MIN (Expectiminimax (C5), Expectiminimax (C6))
Expectiminimax (C5) = 0.9*2 + 0.1*(-22) = 1.8 – 2.2 = -0.4
Expectiminimax (C6) = 0.3 * Expectiminimax (M9) + 0.7 * Expectiminimax (M10)
From above Expectiminimax (M9) = 6.2 and Expectiminimax (M10) = 2.0
Expectiminimax (C6) = 0.3 * 6.2 + 0.7 * 2 = 1.86 + 1.4 = 3.26
Expectiminimax (N1) = MIN (Expectiminimax (C5), Expectiminimax (C6))

= MIN ( -0.4, 3.26) = -0.4
Expectiminimax (N1) = -0.4

(**1 point**) In N2:
Expectiminimax (N2) = Expectiminimax (C7) = 0.4*15 + 0.6*9 = 6 + 5.4 = 11.4
Expectiminimax (N2) = 11.4

(**1 point**) In N3:
Expectiminimax (N3) = Expectiminimax (C8) = 0.3*1 + 0.7 * Expectiminimax (M11)
From above Expectiminimax (M11) = 7.0
Expectiminimax (N3) = 0.3*1 + 0.7 * Expectiminimax (M11) = 0.3*1 + 0.7 * 7 = 0.3 + 4.9 = 5.2
Expectiminimax (N3) = 5.2

(**1 point**) In N4:
Expectiminimax (N4) = Expectiminimax (C9) = 0.9*(-12) + 0.1*3 = -10.8 + 0.3 = -10.5

(**1 point**) In N5:
Expectiminimax (N5) = Expectiminimax (C10) = 0.4*5 + 0.6* Expectiminimax (M12)
From above Expectiminimax (M12) = 12.2
= 2 + 0.6*12.2 = 2 + 7.32 = 9.32
Expectiminimax (N5) = 9.32

(**1 point**) In N6:
Expectiminimax (N6) = Expectiminimax (C11) = 0.6* (-5) + 0.4*11 = -3 + 4.4 = 1.4

(**1 point**) In N7:
Expectiminimax (N7) = Expectiminimax (C12) = 0.3*(-22) + 0.7* Expectiminimax (M13)
From above Expectiminimax (M13) = 8.8
= -6.6 + 0.7*8.8 = -6.6 + 6.16 = -0.44
Expectiminimax (N7) = -0.44

(**1 point**) In N8:
Expectiminimax (N8) = Expectiminimax (C13) = 0.2*2 + 0.8*4 = 0.4 + 3.2 = 3.6
Expectiminimax (N8) = 3.6

(**1 point**) In M1:
Expectiminimax (M1) = MAX (Expectiminimax (C1), Expectiminimax (C2), Expectiminimax (C3), Expectiminimax (C4))
Putting in values for above calculated values:
Expectiminimax (C1) = 0.2 * Expectiminimax (N1) + 0.8 * Expectiminimax (N2)
= 0.2* (-0.4) + 0.8 * 11.4 = -0.08 + 9.12 = 9.04
Expectiminimax (C2) = 0.6 * Expectiminimax (N3) + 0.4 * Expectiminimax (N4)
= 0.6 * 5.2 + 0.4 * (-10.5) = 3.12 – 4.2 = -1.08
Expectiminimax (C3) = 0.3 * Expectiminimax (N5) + 0.7 * Expectiminimax (N6)

= 0.3 * 9.32 + 0.7 * 1.4 = 2.796 + 0.98 = 3.776

Expectiminimax (C4) = 0.9 * Expectiminimax (N7) + 0.1 * Expectiminimax (N8)

= 0.9 * (-0.44) + 0.1 * 3.6 = -0.396 + 0.36 = -0.036

Expectiminimax (M1) = MAX (Expectiminimax (C1), Expectiminimax (C2), Expectiminimax (C3), Expectiminimax (C4))

=MAX (9.04, -1.08, 3.776, -0.036) = 9.04

Expectiminimax (M1) = 9.04

**Software Engineering (includes documentation for your programming assignments)**

**Your README file must include the following:**

- Your name and email address.
- *Homework number* for this class (AI CS6364), and the *number of the problem* it solves.
- A description of every file for your solution, the programming language used, supporting files from the aima code used, etc.
- How your code operates, in detail.
- A description of special features (or limitations) of your code.

**Within Code Documentation:**

- Methods/functions/procedures should be documented in a meaningful way. This can mean expressive function/variable names as well as explicit documentation.
- Informative method/procedure/function/variable names.
- Efficient implementation
- Don't hardcode variable values, etc