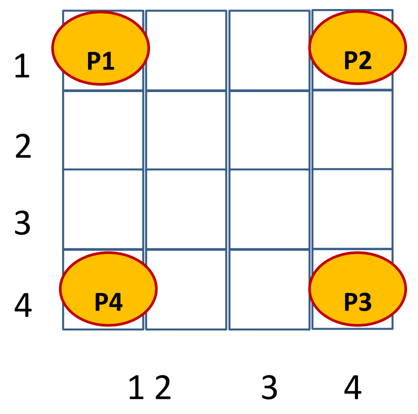
Problem Statement for Multiplayer Game

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 [1,4], 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**) *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!

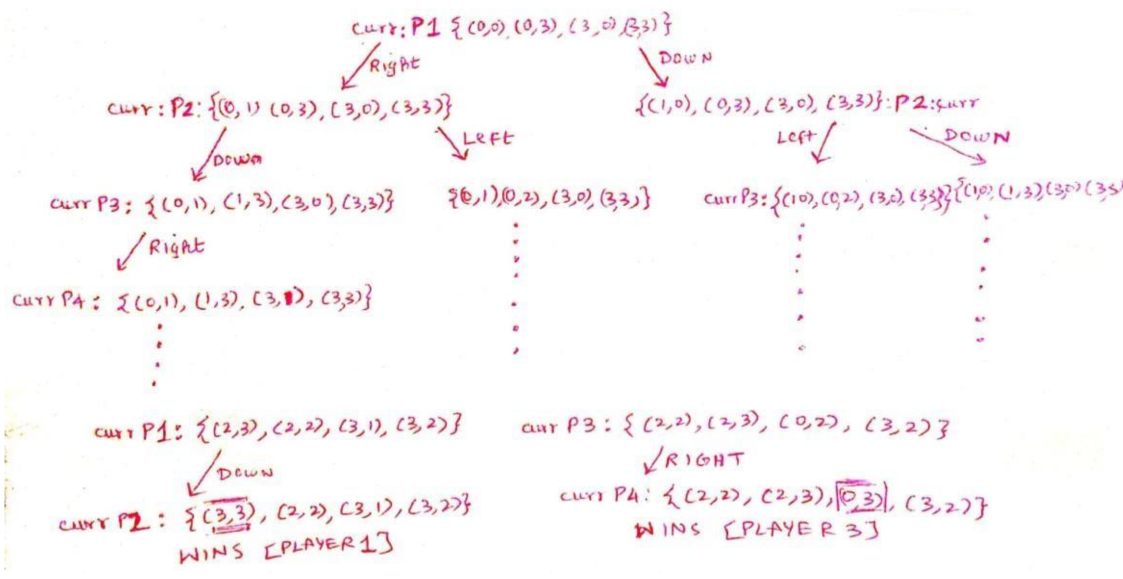
ANSWER)

Each node is represented as x and y coordinates for each player. For example [(x1,y1),(x2,y2),(x3,y3),(x4,y4)] where x1,y1 are coordinates of Player 1 and x2, y2 are coordinates of player 2 and so on.

Initial Node of the game: {(0,0), (0,3), (3,0), (3,3)} Coordinates start from 0 index.

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.

ANSWER)



curr: P1 {(0,0), (0,3), (3,0), (3,3)}
Right
Down
curr: P2: {(0,1) (0,3), (3,0), (3,3)}
{(1,0), (0,3), (3,0), (3,3)} : P2: curr
Down
Left
Left
Down
curr P3: {(0,1), (1,3), (3,0), (3,3)}
{(0,1), (0,2), (3,0), (3,3)}
curr P3: {(1,0), (0,2), (3,0), (3,3)}
{(1,0), (1,3), (3,0), (3,3)}
Right
curr P4: {(0,1), (1,3), (3,1), (3,3)}
curr P1: {(2,3), (2,2), (3,1), (3,2)}
Down
curr P2: {(3,3), (2,2), (3,1), (3,2)}
WINS [PLAYER 1]
curr P3: {(2,2), (2,3), (0,2), (3,2)}
RIGHT
curr P4: {(2,2), (2,3), (0,3), (3,2)}
WINS [PLAYER 3]

Many wins by many players are there in the game tree, I have represented terminal states for two of the players.

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;

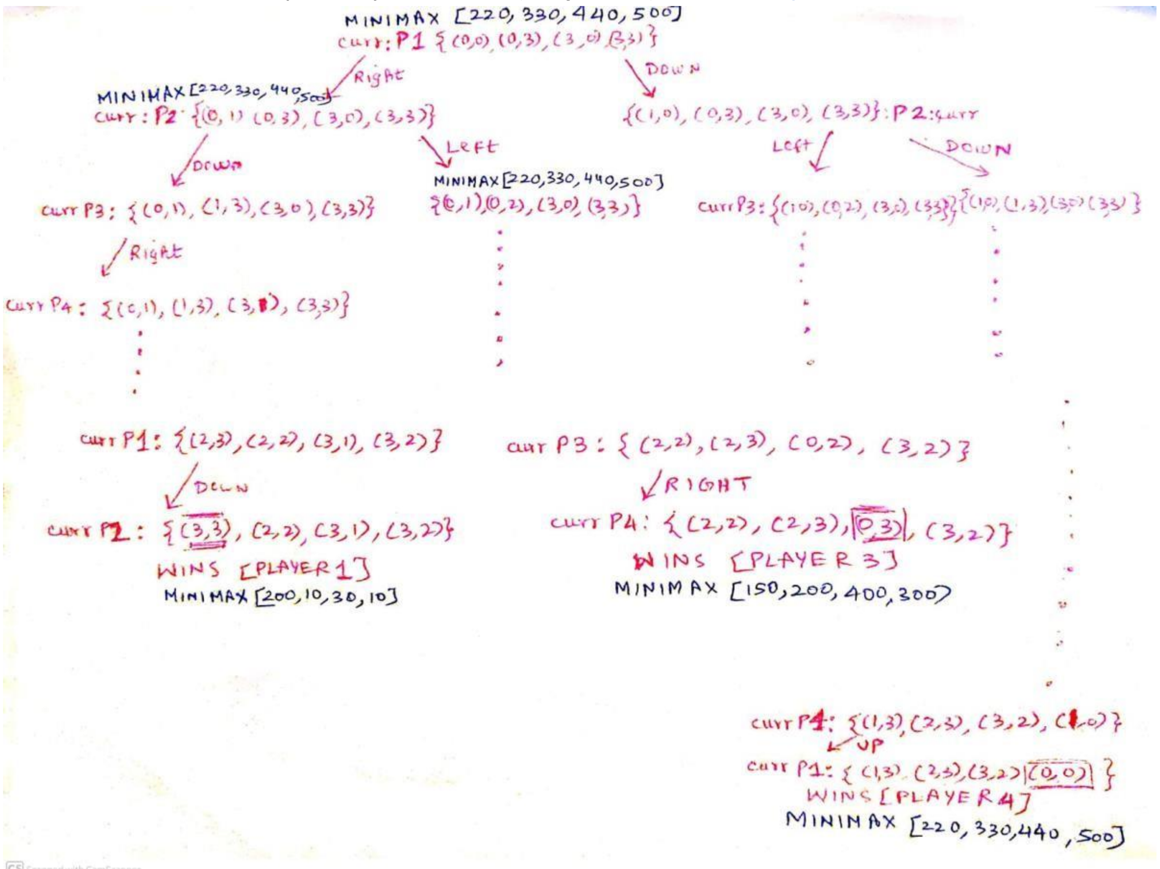
If the minimax values of the repeated states shall be ignored, compute the MINIMAX values in all other states of the game, using the following program assignment.

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 = ?????]

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**) **ANSWER)**


MINIMAX [220, 330, 440, 500]
curr: P1 {(0,0), (0,3), (3,0), (3,3)}
Right
DOWN
MINIMAX [220, 330, 440, 500]
curr: P2 {(0,1) (0,3), (3,0), (3,3)}
{(1,0), (0,3), (3,0), (3,3)}: P2: curr
Down
Left
Left
DOWN
curr P3: {(0,1), (1,3), (3,0), (3,3)}
MINIMAX [220, 330, 440, 500]
{(0,1), (0,2), (3,0), (3,3)}
curr P3: {(1,0), (0,2), (3,0), (3,3)}
{(1,0), (1,3), (3,0), (3,3)}
Right
curr P4: {(0,1), (1,3), (3,0), (3,3)}
curr P1: {(2,3), (2,2), (3,1), (3,2)}
curr P3: {(2,2), (2,3), (0,2), (3,2)}
DOWN
RIGHT
curr P2: {(3,3), (2,2), (3,1), (3,2)}
WINS [PLAYER 1]
MINIMAX [200, 10, 30, 10]
curr P4: {(2,2), (2,3), (0,3), (3,2)}
WINS [PLAYER 3]
MINIMAX [150, 200, 400, 300]
curr P4: {(1,3), (2,3), (3,2), (1,0)}
UP
curr P1: {(1,3), (2,3), (3,2), (0,0)}
WINS [PLAYER 4]
MINIMAX [220, 330, 440, 500]

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**)

ANSWER)
The explored set size come up to 19,069 when there is no Jump. But with Jump included the explored set size increased to 35,268.
In the initial game without jumps, Player 1 had the fastest win with 77 moves. However in the game with Jumps, Player 1 wins fastest with only 17 moves (lesser number of moves).
The number of repeated states for the game **without** Jumps is **24,950** states. The number of repeated states for the game **with** Jumps is **50,784** states.