Thipak B
1910110425

AI Assignment 1

List of classes used:

Class Name	Purpose
Public class HelloApplication extends Application	Main class containing the main function and the start function
Public class MagicCube	To generate the magic cube.
Public class XObot	Holds bot related functions
Public class Coordinate	A data type to store coordinates of the magic cube

Methods used

Public class HelloApplication extends Application

Int	**plr**	has the player value(Player 1 or 2)
Scene	**scene1**	a new scene
Scene	**scene2**	a new scene
Scene	**scene3**	a new scene
Scene	**scene4**	a new scene
Scene	**scene5**	a new scene
Void	**start(Stage stage)**	to create an interactive UI
Boolean	**haswon(Coordinate p)**	to check if the player has won
Void	**main(String ar[])**	the main() function

Public class MagicCube

int	**size**	size of the magic cube
Int	**mc[][][][]**	4D array to depict the magic cube Also the game cube
Void	**initmc()**	initializes all the playable position as zero.
Void	**positivefill(int n,String p,int f)**	fill number 1-9 in the magic cube
Void	**filldiag()**	fill the diagonals opposite to 1-9
Void	**fillrem()**	fill the remaining squares by simple arithmetics

Thipak B
1910110425

Void	**fill()**	when called, completes filling the magic cube
Void	**createCube()**	used to call the fill function
Void	**display()**	displays the magic cube in the terminal

Public class XObot

Int	**p**	flag
Int	**status**	holds the status of the game: 0 - on game 1 - bot wins 2 - player wins 3 - draw
List<Coordinate>	**trackm**	contains the moves of bot
List<Coordinate>	**trackp**	contains the moves made by player
constructor	**XObot(int v)**	constructor
Coordinate	**findc(int v)**	given a number from magic cube, returns the coordinates of the number
Boolean	**coll(Coordinate o1,Coordinate o2,Coordinate o3)**	checks if the coordinates are collinear
Coordinate	**isfree(int val)**	check if the coordinate is already filled
Boolean	**isok(int val1,int val2,int val3)**	check if it makes a valid 42 and call coll
Coordinate	**checkwinning(List<Coordinate> track)**	check if any winning moves are present
Void	**move(Button b1[],Button b2[],Button b3[])**	Plays as a bot.

Public class Coordinate

Int	**x**	holds the x-coordinate
Int	**y**	holds the y-coordinate
Int	**z**	holds the z-coordinate
constructor	**Coordinate(int i,int j,int k)**	initializes the object with i,j,k as x,y,z

Thipak B
1910110425

Algorithms

Creating the Magic cube

Start
Value of coordinate(1,1,1) = 14
The first coordinate to be assigned is (1,1,0)
and integer i = 1
Call function1(coordinate(x,y,z) , i) //The numbers from 1 - 9 would be filled

Loop through each coordinate of a(x,y,z)
    If value of coordinate != 0
    Then -> find the coordinates of the opposite point with
            coordinate(1,1,1) as pivot
            Value of new point = 28 - value of coordinate a(x,y,z)
Loop through each coordinate of a(x,y,z)
    If value of coordinate == 0
    Then -> coordinate(x,y,z) = 42 - coordinate(x,y,0) + coordinate(x,y,1) +
                                                        coordinate(x,y,2)

function1(coordinate(x,y,z) , i)
    If the coordinate is empty
    Then -> assign ‘i’ to the coordinate(x,y,z).
        y = y-1
        z = z-1
        If y == -1 then y =2
        If z == -1 then z = 2
    Else
        x = x +1
        If x == 3 then x = 0
    If i == 9
    Then -> return
    Else
        Call function(coordinate a(x,y,z) , i+1)

Thipak B
1910110425

BOT Playing the Game

Define two lists of coordinates
Trackm and trackp
Trackm for keeping track of the bots moves
Trackp for keep track of the players moves

```
//Check for draw
If size(trackm) or size(trackp) >= 14
Then->      game draw
            end

//first move
if size(trackm) == 0
Then->      check if coordinate(1,1,1) is free
                        then-> add (1,1,1) to trackm
                  Else   -> add(0,0,1)
                  Return to player move

//check for a winning move
Coordinate temp = checkwinning(  trackm  )

//check winning function checks if there is a winning move for the present list and
returns a coordinate if present
//algorithm for that is present after this algo

If temp != NULL
Then -> add temp to trackm
          Mark coordinate temp as filled
          BOT wins
          End

Check for a opponent winning move
Coordinate temp = checkwinning(  trackp  )

If temp != NULL
```

Then -> add temp to trackm
Mark coordinate temp as filled
Return to player's move

//if no winning conditions are present
Loop through all even indexes
//(an index is even index if all x,y,z are even numbers)
//even indexes are corners so basically loop through corners
If coordinate(x,y,z) is free
Then -> add coordinate(x,y,z) to trackm
Coordinate temp = checkwinning(coordinate(x,y,z))
If temp != NULL
then-> mark coordinate temp as filled
Return to player's move
Else-> remove last coordinate in trackm
Continue loop to next index

Algorithm for the checkwinning(list track) function

Start
If size(track) > 1
Then -> let last be the last coordinate inserted
Loop through all elements of track
Sum = value(last) + value(loop element)
Value(next element) = 42 - sum
If value of next element <= 27
//numbers in magic square of order 3 cant exceed 27
Then -> check if last, loop element, next element are collinear
Then -> return next element
Else return null