INFO 6205 SPRING 2022 PROJECT

MENACE

PROGRAM STRUCTURES AND ALGORITHMS

INSTRUCTOR: ROBIN HILLYARD

SUBMITTED BY:

SAI HARISH REDDY GUNDA

NUID:002981776

SUSHMITA MAITY

NUID: 001092534

Git hub link: https://github.com/harish-gunda/INFO6205_Final_Project_TicTacToe

Table of Contents:

INTRODUCTION

Donald Michie designed a model of matchboxes in 1960 called MENACE. It contained number of beads and learned to play tic-tac-toe. The model optimizes itself depending on success, as games are kept on playing. It plays more favorable moves that have more chances to win.

MENACE makes a move when the human player randomly picks a bead out of the box that represents the game's current state. The color of the bead determines where MENACE will move. The human player chooses the beads at random the beads are adjusted when there is failure or success. If MENACE loses at the end of each game, each bead MENACE used is removed from each box. If MENACE wins, three beads the same as the color used during each individual turn are added to their respective box. If the game resulted in a draw, one bead is added. After 200 games play out, the matchboxes will have more of some beads than of others. This makes it more (or less) likely for a given bead to be picked.

AIM:

1. Implement the MENACE by replacing the matchboxes with the values in a hashtable with key as the state of the game.
2. Train the menace by choosing values for:
   The number of “beads” to add in matchbox
   alpha: at start of the game for first, second or different moves
   beta: in the event of a win
   gamma: in the event of a loss
   delta: in the event of a draw
3. Implement the human strategy to play with Menace:
   a. Win
   b. Block
   c. Fork
   d. Block Fork
   e. Center
   f. Opposite corner
   g. Empty Corner
   h. Empty side
4. Implement Logging
5. Unit tests

APPROACH:

1. The Menace is trained against human strategy for 10000 matches.
2. For every new move which creates a new combination of positions in the tic-tac-toe board, a key is created with a list of all possible moves by menace as the value and stored in a HashMap.
3. All the states and moves by the menace in the game are tracked.
4. The menace is updated by taking beta, gamma, delta values after each match and the list containing match positions is reset. Beta, Gamma and Delta contain the win, loss and draw results of the game.
5. The trained menace is played against human strategy with 0.9 probability for a total of 100 matches. Observations are made on different parameters for different runs and results are logged.

Program: Data Structures & Classes & methods

1. HumanStrategy.java

1.1. probablity() - decides to make a random move or follow human stategy

1.2 convert1dstringTo2darray() - converts String board to integer array

1.3 nextOptimalStep()-logic for human strategy with 8 rules

2. MenaceStrategy.java
   3.1. updateBeads() - updates beads after each step
   3.2. nextOptimalStep() - ouputs the best move
   3.3. createKeyValue() - creates key value pair of board positions and beads

3. Game.java
   3.1. run() - Runs one full game

   3.2. checkwin() -checks the player who has won the game by checkWin().

   3.3 peekCurrentState()- prints the current board state

   3.4 convert2dboardToString() - converts 2d int array board to string

4. Runner.java

4.1. Main method – runs the Game in loop multiple times.

Algorithm

1 . Initialize a HashMap which holds the current state of the board(matchbox) as its “key” and list of indexes of the empty board positions(beads) as “value”

2. Start the game

3.If the hashmap contains the current board state “key”, Pick a random index from its “value” and remember these values in a temporary Hashmap to update after the game.

3. get human marker position

3. Iterate step 2 until the human or the menace wins or the game ends in a draw

4. a)If the Human wins remove the indexes from the Hashmap “value” of the corresponding “key”

B) If the menace wins add the indexes from the Hashmap “value” of the corresponding “key”

c) If the game ends in draw add the indexes from the Hashmap “value” of the corresponding “key”

5. Iterate the game multiple times until the menace is trained

Invariants

Human Strategy Invariants used to play games with the Menace:

1. Win
2. Block
3. Fork
4. Block Fork
5. Center
6. Opposite Corner
7. Empty Corner
8. Empty Side
9. Probabiltity(p*)

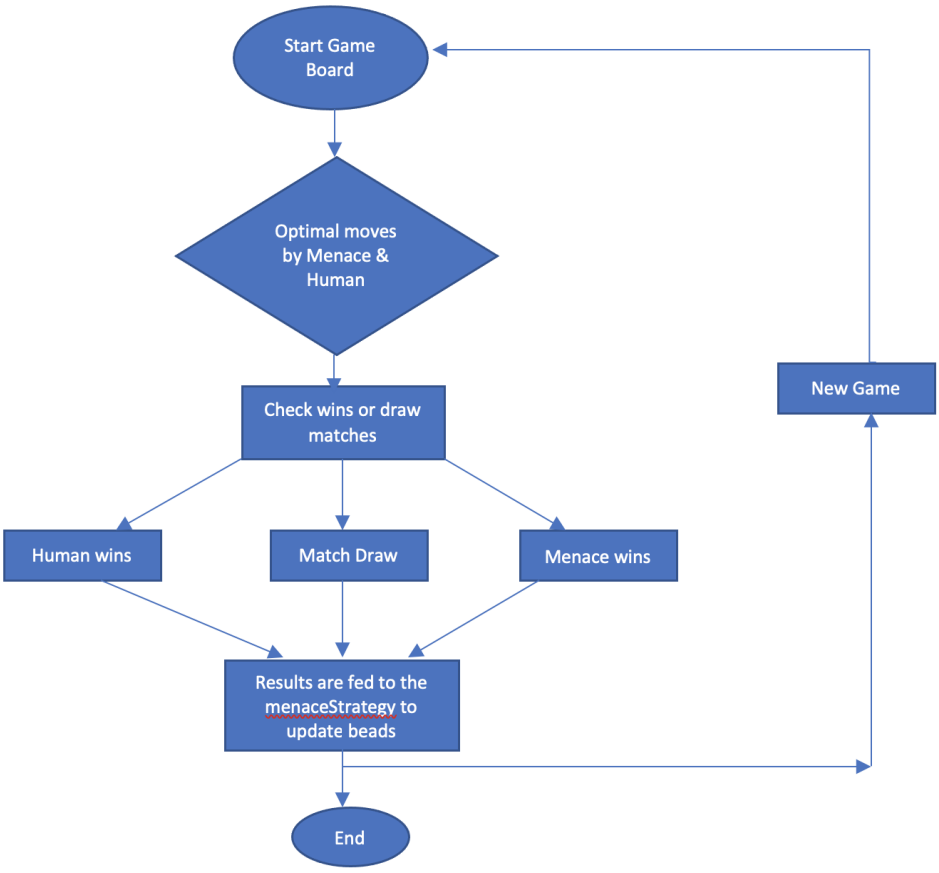
Menace Strategy invariants:

1. Alpha
2. Beta
3. Gamma
4. Delta

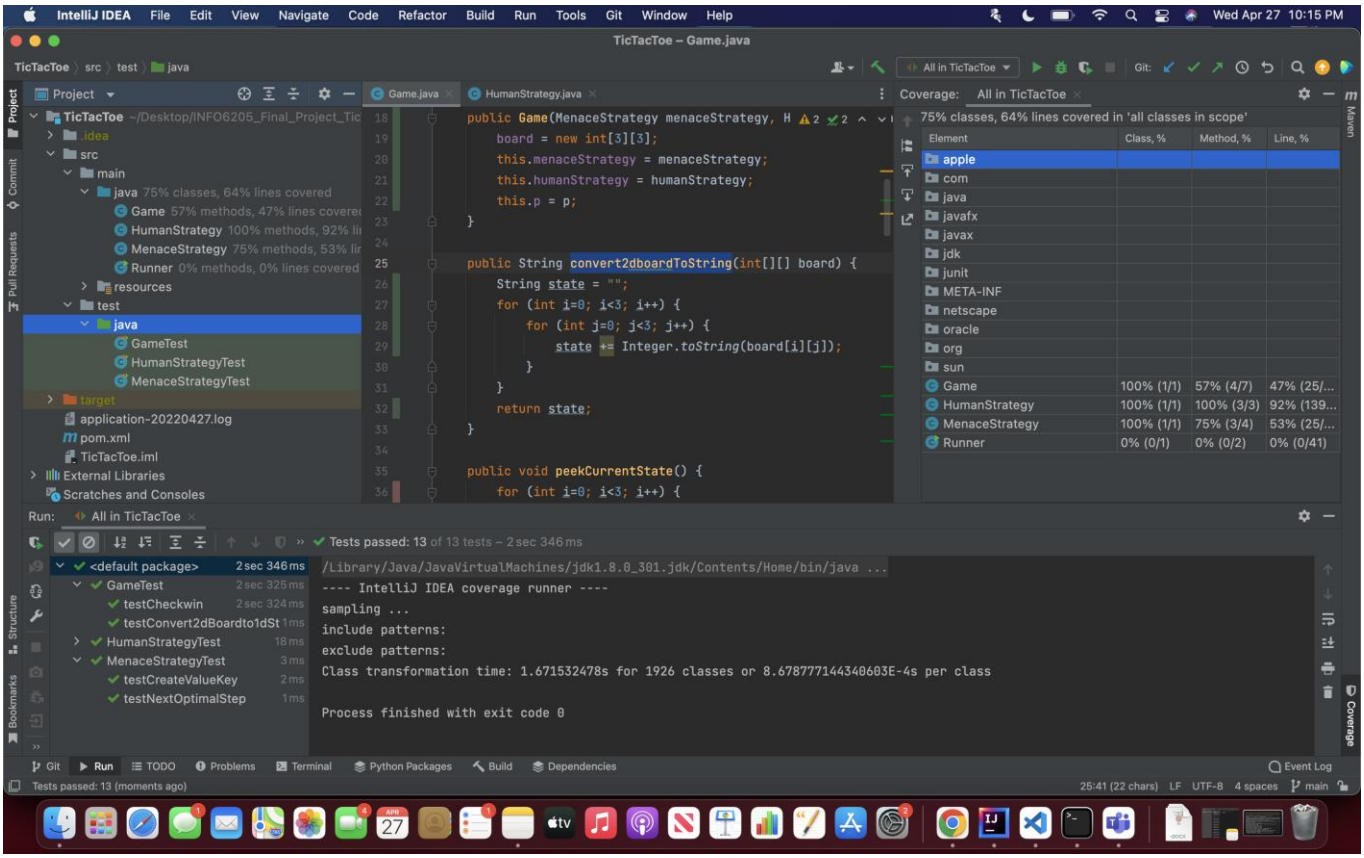
Game invariants:

1. Any player with 3 marks in a row/column/diagonal wins
2. If all the positions are marked, then the game is a draw
3. The grid is always 3X3

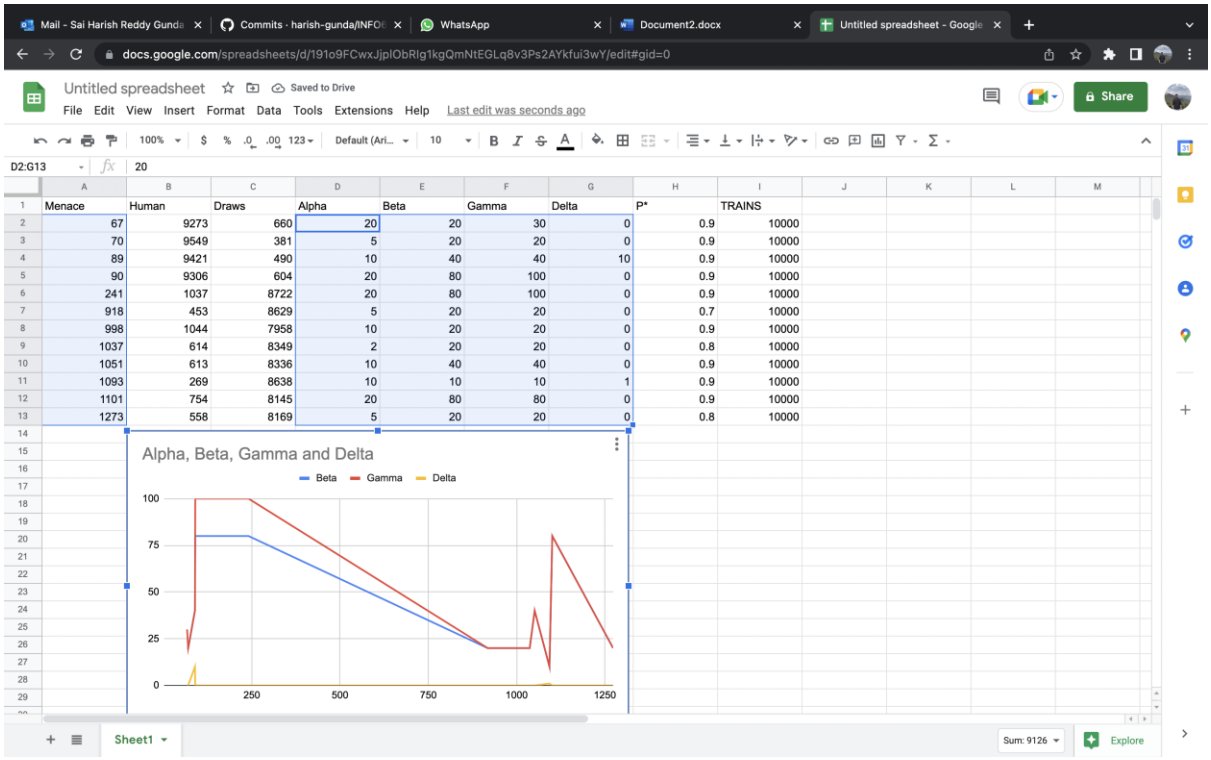
Flow Charts


Start Game Board
Optimal moves by Menace & Human
Check wins or draw matches
Human wins
Match Draw
Menace wins
Results are fed to the menaceStrategy to update beads
End
New Game

Unit Tests:


IntelliJ IDEA File Edit View Navigate Code Refactor Build Run Tools Git Window Help
Wed Apr 27 10:15 PM
TicTacToe – Game.java
Project
Game.java
HumanStrategy.java
Coverage: All in TicTacToe
75% classes, 64% lines covered in 'all classes in scope'
Element Class, % Method, % Line, %
Game 100% (1/1) 57% (4/7) 47% (25/...
HumanStrategy 100% (1/1) 100% (3/3) 92% (139...
MenaceStrategy 100% (1/1) 75% (3/4) 53% (25/...
Runner 0% (0/1) 0% (0/2) 0% (0/41)
Run: All in TicTacToe
Tests passed: 13 of 13 tests – 2 sec 346 ms
/Library/Java/JavaVirtualMachines/jdk1.8.0_301.jdk/Contents/Home/bin/java ...
---- IntelliJ IDEA coverage runner ----
sampling ...
include patterns:
exclude patterns:
Class transformation time: 1.671532478s for 1926 classes or 8.678777144340603E-4s per class
Process finished with exit code 0

Analysis



Menace	Human	Draws	Alpha	Beta	Gamma	Delta	P*	TRAINS
67	9273	660	20	20	30	0	0.9	10000
70	9549	381	5	20	20	0	0.9	10000
89	9421	490	10	40	40	10	0.9	10000
90	9306	604	20	80	100	0	0.9	10000
241	1037	8722	20	80	100	0	0.9	10000
918	453	8629	5	20	20	0	0.7	10000
998	1044	7958	10	20	20	0	0.9	10000
1037	614	8349	2	20	20	0	0.8	10000
1051	613	8336	10	40	40	0	0.9	10000
1093	269	8638	10	10	10	1	0.9	10000
1101	754	8145	20	80	80	0	0.9	10000
1273	558	8169	5	20	20	0	0.8	10000

Conclusion

1) For small values of alpha(~2) and beta(~10) gamma(~10) menace performed very well
2) For large values of Alpha(~50) and beta (100), gamma(~100) decreasing the probability helped menace perform better
3) Also for the same alpha, beta, gamma values menace performed differently
4) Delta value did not affect the performance of the menace

References:

1. https://odsc.medium.com/how-300-matchboxes-learned-to-play-tic-tac-toe-using-menace-35e0e4c29fc

2. https://en.wikipedia.org/wiki/Tic-tac-toe

3. https://opendatascience.com/menace-donald-michie-tic-tac-toe-machine-learning/

4. https://www.youtube.com/watch?v=R9c-_neaxeU