Program Structures & Algorithms
Spring 2022
Project: The Menace

Team: 2 Members

NUID	Name	Email
002139730	Prathamesh Nemade	nemade.p@northeastern.edu
001568243	Urvang Patel	patel.ur@northeastern.edu

Aim:

To develop a MENACE machine that excels in the game of tic-tac-toe. The hashtables of a machine are modeled and trained by playing "n" games against a human/optimal strategy. Based on results of the games, make improvements to a model.

Approach:

MENACE,
Creating states for the MENACE machine, as well as providing the same number of initial beats to each of the playable places.
Considering that MENACE is the initial player, after every even move, hunt for the state in the HashTable and pick up the empty state using randomly weighted logic.
By rewarding and subtracting beats from the appropriate states and moves, the model is trained.

Human,
Human can play optimally or randomly.
As per the algorithm,
If the move is chosen randomly, human randomly selects the empty spot.

Or, if chosen optimally, it follows the sequence of following steps:

a. Play for win
b. Block opponent win
c. Fork
d. Block Fork
e. Play center, if empty
f. Play opposite corner
g. Play corners, if empty
h. Play sides, if empty

Data Structures & classes

Following are the **Data structures** used:

1. HashMaps
2. HashSet
3. ArrayList
4. Arrays
5. Graphs

Classes:

1. BaseState.java
   This class is used to keep a track on all the methods related to Menace. This is responsible to generate the combinations/states, validate them, rewarding the menace, a few methods related to mirror and rotating images/states of the board.

1. Human.java
   This class is used to keep a track on all the methods related to Human play. Methods related to optimal move i.e., Fork, Block Fork, Vertical win, Horizontal win, etc. are present in this class.

1. CSV.java
   This consists of a few dynamic methods which are responsible to read and write CSV files. This is used to persists the state of the game.

1. Utils.java
   This consists of a few methods(generic) which are been referred through the codebase.

1. MenaceLogger.java
   This class is used to initialize the logger and allow it to let log the statements in the codebase.

1. MenaceBegin.java
   This is the main method used to play the game. This accepts an argument i.e., true or false. By default, is false, which would train the MENACE. Whereas, true would allow use to play the game by inputting the spots to mark.

Algorithm:

Creating states by using three integers {0, 1, 2} to generate a nine-digit number.
The empty state is “0”, the MENACE move is “1”, and the human move is “2”.
We also store a nine-digit number in a HashMap once it is generated.
The structure of HashMap is as follows:
{
“100000002”: [0,9,9,9,9,9,9,9,9,0]
}

The key in the HashMap is the serialized state of the tic-tac-toe game and the value is the empty spots/locations holding respective no. of beats(alpha) in them.

The states are being validated before storing in the HashMap based on the following conditions:
a. As MENACE plays first, therefore there should be the same no. of moves (total no. of 1s and 2s in the state)
b. We consider the total moves less than 8 as those are the only crucial ones. After 7 the chances of winning are very less whereas, the chances of drawing are more.
c. That state should not be in a “win-win” state.
d. Checking if the mirror image is already present in the HashMap

By following all the validations/filtering conditions, the total no. of states obtained is 304(considering MENACE plays first).

Later, when the game begins,
We check the state (serialized version) of the game in the HashMap, if it is not present, we check for the mirror images of that particular state in the HashMap.
Once the mirror image is found in the HashMap, we replace/rotate or tic-tac-toe board in such a way that the serialized version of the state is equal to the one found in the HashMap.

To pick a position/spot to play, we again check the state (serialized version) of the game in the HashMap, if it is not present, we check for the mirror images of that particular state in the HashMap. If the game turn is less than 8 then the state is for the sure present in the HashMap and for later chance, a random no. is given the priority.
Once, the state is found in the HashMap we look for its value of it and pick a random index based on the no. of beats(weights). Therefore, a weighted random index algorithm has been used here to pick a random index based on the weights.
If the state is not found in the HashMap (for chance more than 7), we pick a random index and play the respective position on the board and move forward.

In the human turn, the algorithm has 2 options to play from. Humans can play in a defined optimal step, or it can play randomly. When it’s human’s turn, we first select which method to play with. The selection is done based on a **weighted random algorithm**, where the weights are assigned according to the probability. For example, if the probability of human playing optimally is 0.9 the weight associated with it will be 9 and the weight to play randomly will be 1. So, the total weight is 10 and it is adjusted accordingly.

If optimal algorithm is selected, human will follow defined steps which are,

1. Check if human can win after this move. If yes, play that move.
2. Check if menace is going to win in the next set. If yes, block that move
3. Check if human can play fork against menace. If yes, play the fork move
4. Check if menace is playing fork. If yes, block the fork
5. Check if the center of the board is empty. If yes, play center move.
6. Check if a corner opposite to the menace is empty. If yes, play that opposite corner.
7. Check if any other corner is empty. If yes, play corners.
8. Check if any side is empty. If yes, play that side.

If the random algorithm is selected, find all the empty positions on the board. Select one position out of these randomly and play that move.

Invariants:

The following are the invariants in the tic-tac-toe game:

1. It follows a class variant, as there are two players in the game, and the no. Xs or Os in each turn is relatable.
   E.g., No. of O's are less than X's, considering X plays first.

2. To win the game, there are exactly 8 conditions.
   a. Combination of 3 in a row - count 3
   b. Combination of 3 in a column - count 3
   c. Combination of diagonals - count 2

3. As there are two different players in the game, they play alternatively.

4. Maximum no. of moves possible is 9.

5. In case of mirror images and rotation of the board,
   The central positions are never affected while check mirror images, as it's going to be the same on rotation and mirror of the board.

   a. Middle row spots can never be at the corner.
   b. The corner spots are more affected by rotation.

Flow Charts:


MenaceBegin
Serialize Board
eward or Punish Men...
Menace Turn
Human Turn
Fetch the state from...
Check mirror images
Decide how to play
Pick one as per the probability configura...
Get the beads for th...
Play Optimal Strater...
Play Random postion
Store the state and...
Pick one bead random...
Check if it is win o...
No
Won / Draw / Loss
Text is not SVG - cannot display

Graphs:

The following graphs are plotted based on varying following properties,

Alpha – No. of beats at initial stage
Beta – No. of beats added on win state
Gamma – No. of beats removed on losing the turn
Delta – No. of beats added on “draw” state

We conducted different runs with various configurations. There were 3 types of system: High rewarded system, High Punished System, Average rewarded-punished system. From the results and the graphs, we can see the difference between the 3 systems.

Initially, Menace loses as it deals with the human i.e., optimal strategy with a bit of randomness. But after earning a few wins and picking up the randomly empty slots based on the weights(beats), Menace learns to block the human win and draw the game.

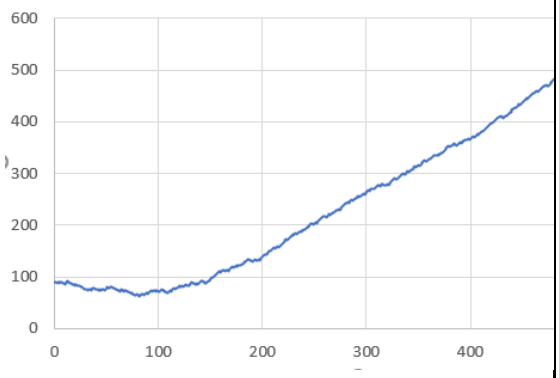
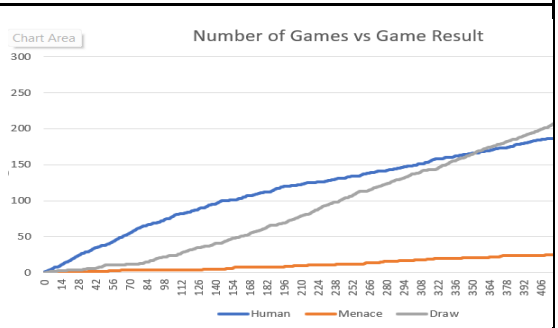
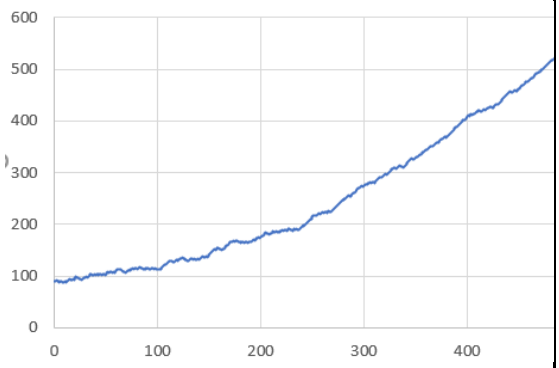
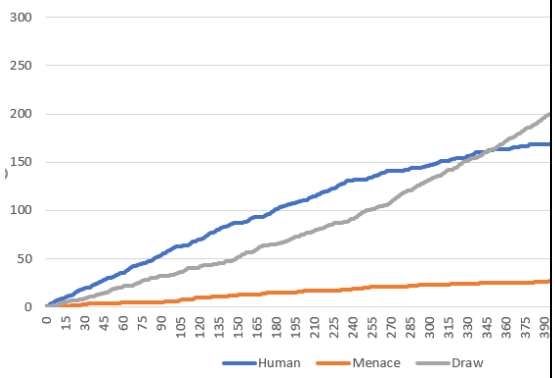
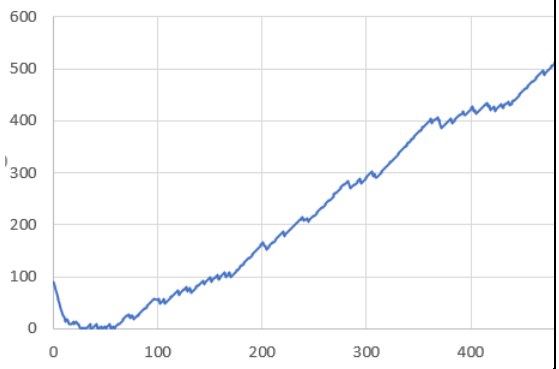
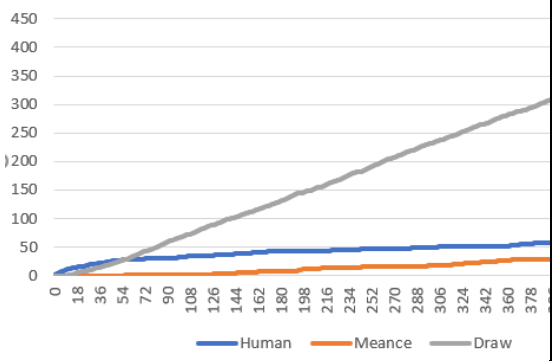
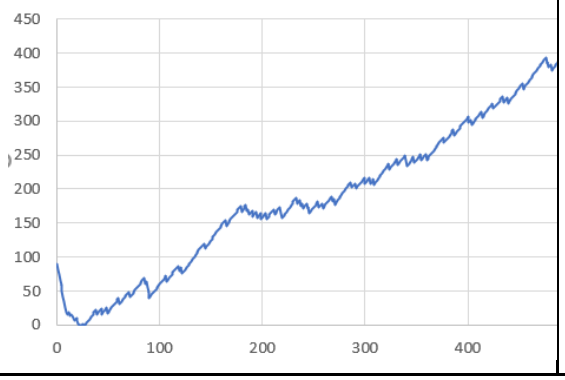
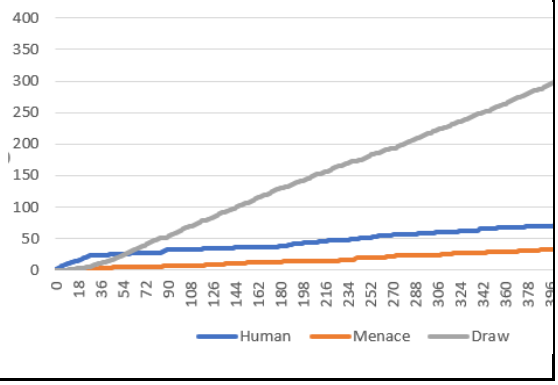
Therefore, after around 100 games (considering max. probability for picking up optimal strategy) the menace learns to block or win the games.
Note: For humans, the randomly picked spot is based on the “Picking up randomly index based on weights(beats)” algorithm.
As the probability decreases from 0.9 to 0.5, the chances of “draw” or “win” increases.

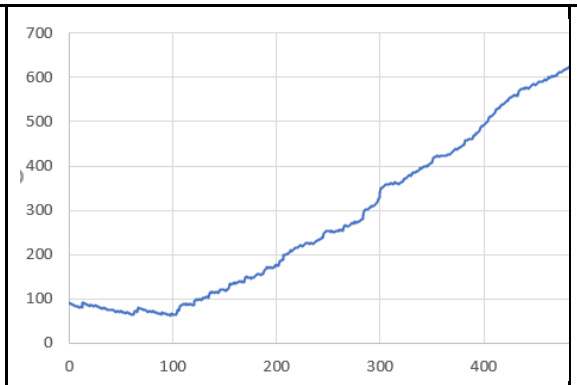
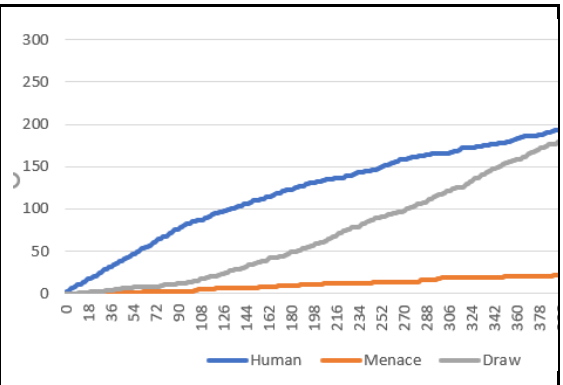
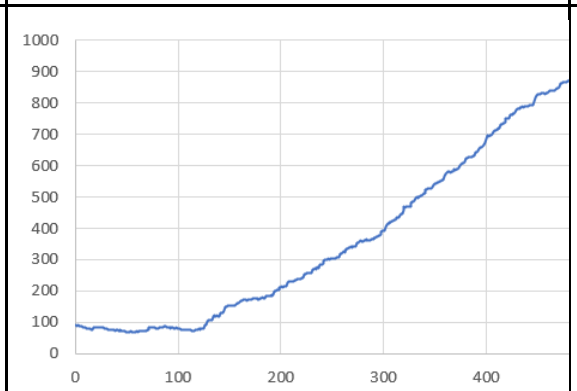
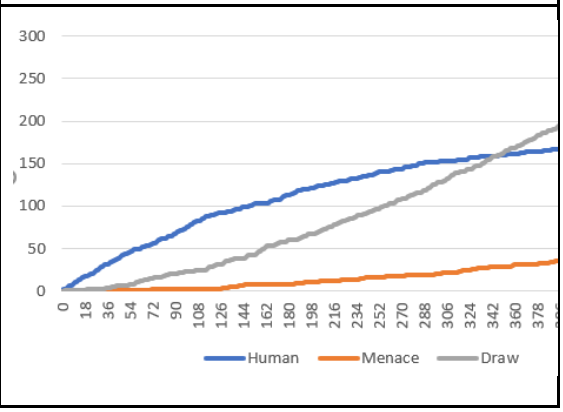
When the Menace is punished by deducting beats, the Menace learns to block the move by picking up a spot based on the weights(beats) available at the empty spots.

When the machine is punished high it quickly learns to draw the game, but it still cannot win many games against human strategy. We also observe that the beads in the first state take a dip in the beginning as it gets punished repeatedly, and then it slowly picks up beads in the correct positions. We also notice that it restricts menace into taking limited number of paths as other beads in other paths were reduced to zero or negligible.

When the machine is rewarded highly, we observe that it takes longer to learn and get draws or wins. This happens because multiple paths get more beads as they win, and it is not able to optimally select the correct paths as the options are high.

Whereas when the normal system on average takes few games to learn, but it allows the machine to train various paths and get better results than other machines.

Config	Games vs Total Beads	Games vs Results
Alpha=10 Beta=3 Gama=1 Delta=2 P=0.9		
Alpha=10 Beta=3 Gama=1 Delta=2 P=0.5		
Alpha=10 Beta=3 Gama=8 Delta=2 P=0.9		
Alpha=10 Beta=3 Gama=8 Delta=2 P=0.5		

Alpha=10 Beta=10 Gama=1 Delta=2 P=0.9		
Alpha=10 Beta=10 Gama=1 Delta=2 P=0.5		

Mathematical Analysis:

The total no. of positions a player can play is 9.
The total no. of values a spot can hold is 3. Following are the values:

1. Empty Spot
2. Player 1
3. Player 2

Therefore, total no. of possible combinations: 3*3*3*3*3*3*3*3*3 = 19,683. This count is including

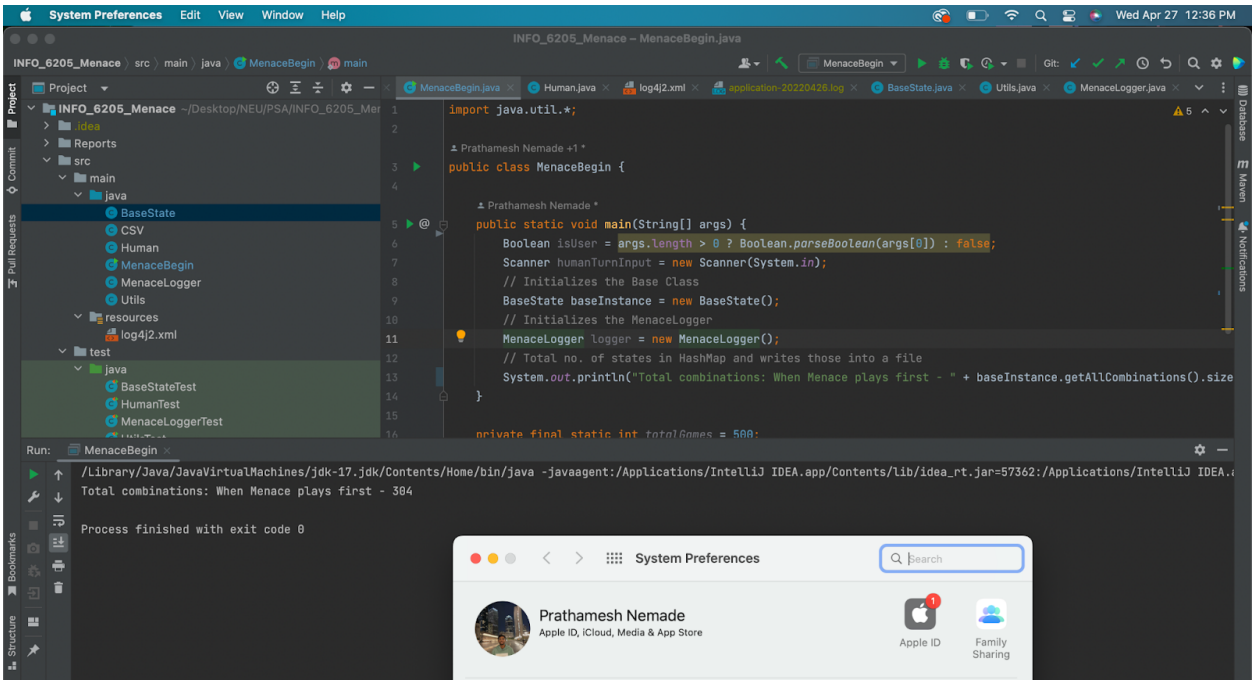
1. Duplicates
2. Invalid Moves
3. Win states
4. Mirror images
5. Rotation images

Minimum no. of overall moves required is 5.

Draw case,
Therefore, there are only: 2^9 (512) filled board states (including duplicates and repetitions).
The total moves Menace can play is 5, considering menace plays first.
Combination: 5!
The total moves Humans/Users can play is 4
Combination: 4!
Therefore, the final filled board: 9! /5!4! = 126 (including rotations)

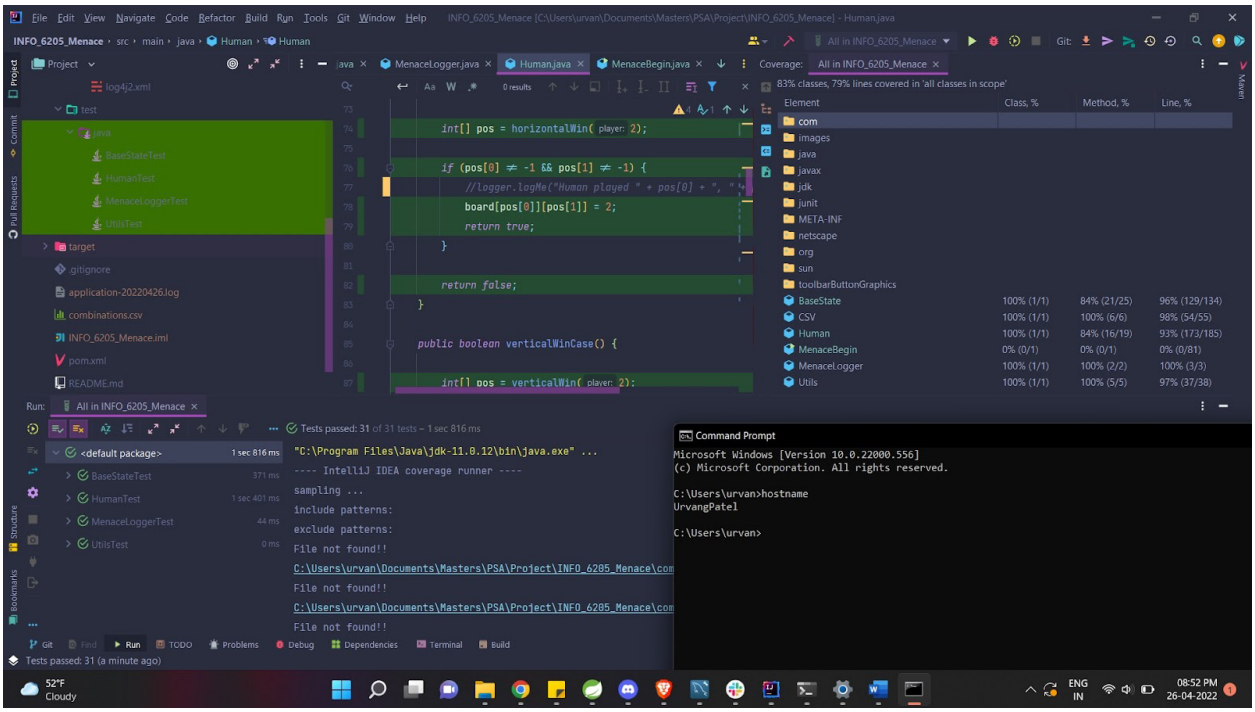
But, to train the menace(machine) we have to consider only 7 moves as Menace can play only 1,3,5, and 7 chances.
Therefore, valid = 3^7(containing rotations and mirror images) - as menace plays odd position and the very next turn/chance is the last “draw” change.

Once the mirror images and rotations are removed,
(Proof, check screenshot attached).
If menace goes first:
For every menace move, the chance is an odd chance therefore we need to consider only the same no. X’s played and O’s played turns.
Menace plays: {1, 3, 5, 7}
Therefore, the available combinations (excluding mirror images and rotations are) are:
1, 12, 108, 183, which results in 304.
If human goes first = 470

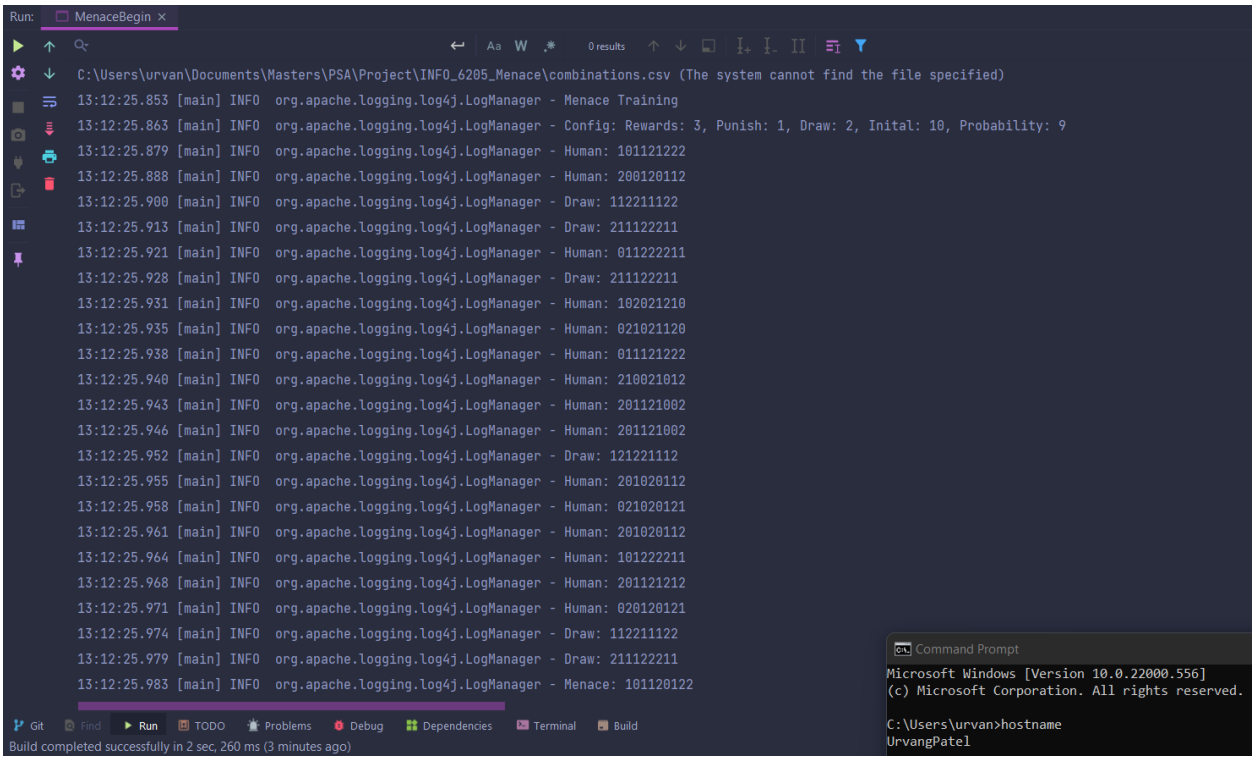

System Preferences Edit View Window Help
Wed Apr 27 12:36 PM
INFO_6205_Menace – MenaceBegin.java
Project
INFO_6205_Menace ~/Desktop/NEU/PSA/INFO_6205_Mer
.idea
Reports
src
main
java
BaseState
CSV
Human
MenaceBegin
MenaceLogger
Utils
resources
log4j2.xml
test
java
BaseStateTest
HumanTest
MenaceLoggerTest
MenaceBegin.java
Human.java
log4j2.xml
application-20220426.log
BaseState.java
Utils.java
MenaceLogger.java
import java.util.*;
Prathamesh Nemade +1 *
public class MenaceBegin {
Prathamesh Nemade *
public static void main(String[] args) {
Boolean isUser = args.length > 0 ? Boolean.parseBoolean(args[0]) : false;
Scanner humanTurnInput = new Scanner(System.in);
// Initializes the Base Class
BaseState baseInstance = new BaseState();
// Initializes the MenaceLogger
MenaceLogger logger = new MenaceLogger();
// Total no. of states in HashMap and writes those into a file
System.out.println("Total combinations: When Menace plays first - " + baseInstance.getAllCombinations().size
}
private final static int totalGames = 500;
Run:
MenaceBegin
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=57362:/Applications/IntelliJ IDEA.
Total combinations: When Menace plays first - 304
Process finished with exit code 0
System Preferences
Search
Prathamesh Nemade
Apple ID, iCloud, Media & App Store
Apple ID
Family Sharing

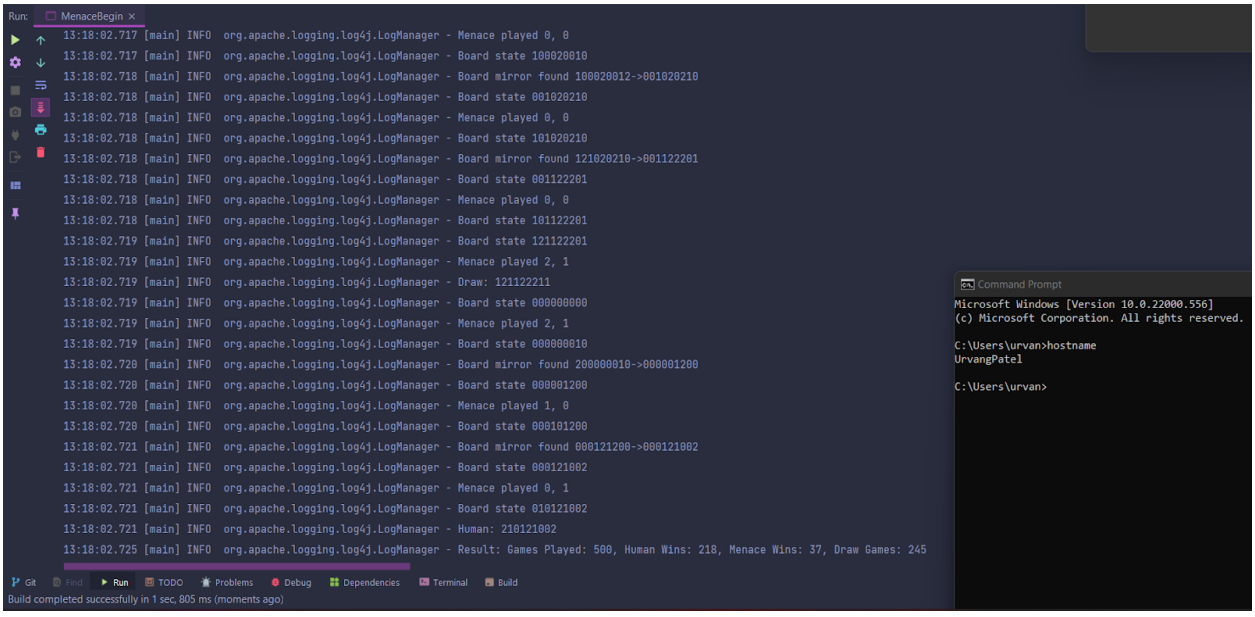
Unit Tests:

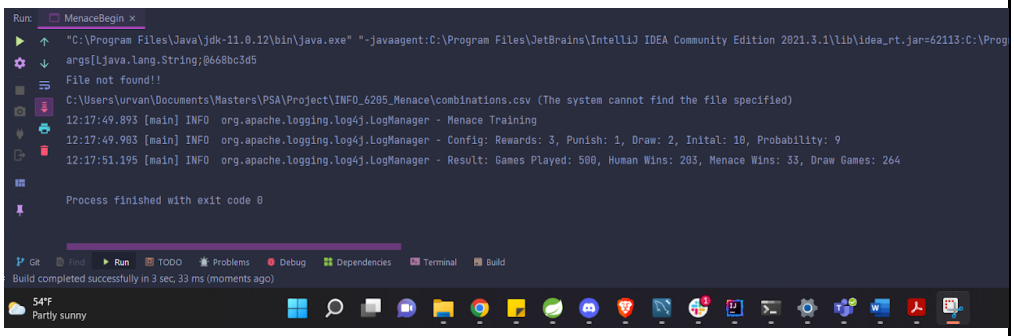
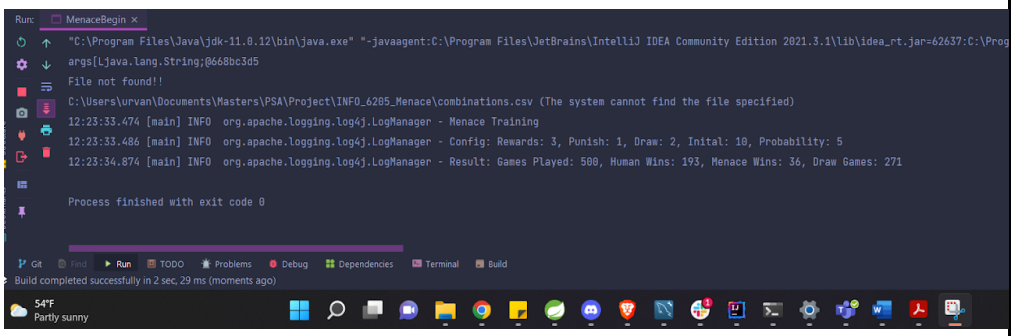
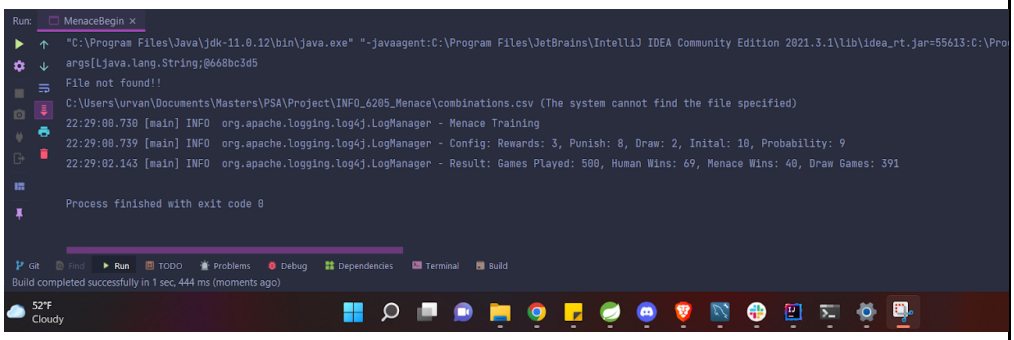
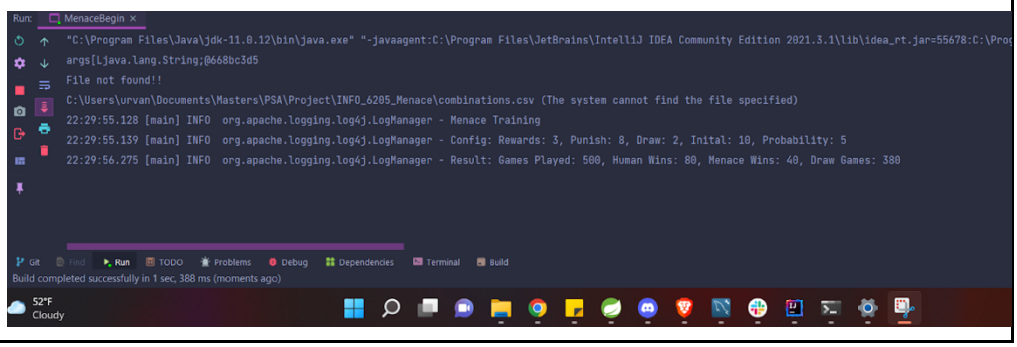
Test coverage covers around **80%** of the code lines.
No. of test cases written: **31**

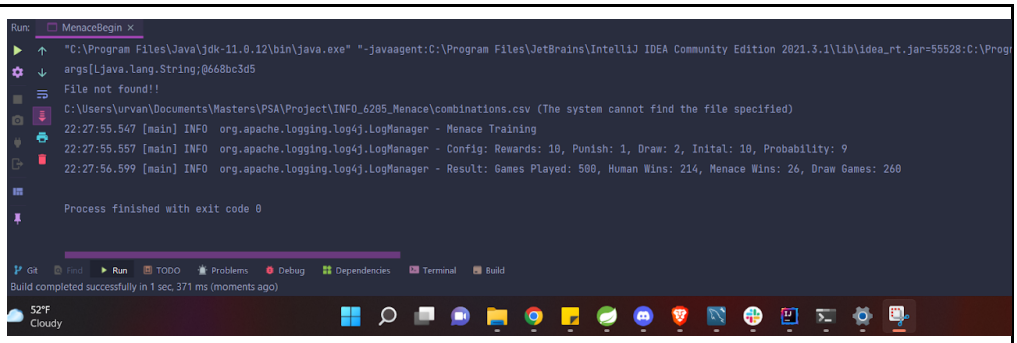
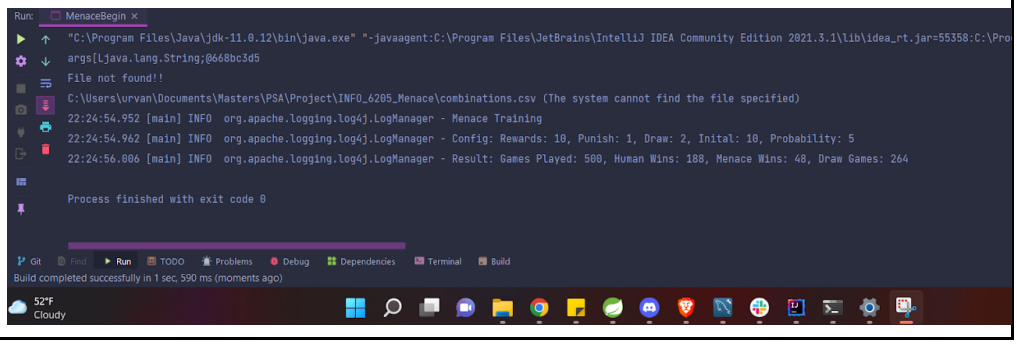


Outputs:


Run: MenaceBegin
C:\Users\urvan\Documents\Masters\PSA\Project\INFO_6205_Menace\combinations.csv (The system cannot find the file specified)
13:12:25.853 [main] INFO org.apache.logging.log4j.LogManager - Menace Training
13:12:25.863 [main] INFO org.apache.logging.log4j.LogManager - Config: Rewards: 3, Punish: 1, Draw: 2, Inital: 10, Probability: 9
13:12:25.879 [main] INFO org.apache.logging.log4j.LogManager - Human: 101121222
13:12:25.888 [main] INFO org.apache.logging.log4j.LogManager - Human: 200120112
13:12:25.900 [main] INFO org.apache.logging.log4j.LogManager - Draw: 112211122
13:12:25.913 [main] INFO org.apache.logging.log4j.LogManager - Draw: 211122211
13:12:25.921 [main] INFO org.apache.logging.log4j.LogManager - Human: 011222211
13:12:25.928 [main] INFO org.apache.logging.log4j.LogManager - Draw: 211122211
13:12:25.931 [main] INFO org.apache.logging.log4j.LogManager - Human: 102021210
13:12:25.935 [main] INFO org.apache.logging.log4j.LogManager - Human: 021021120
13:12:25.938 [main] INFO org.apache.logging.log4j.LogManager - Human: 011121222
13:12:25.940 [main] INFO org.apache.logging.log4j.LogManager - Human: 210021012
13:12:25.943 [main] INFO org.apache.logging.log4j.LogManager - Human: 201121002
13:12:25.946 [main] INFO org.apache.logging.log4j.LogManager - Human: 201121002
13:12:25.952 [main] INFO org.apache.logging.log4j.LogManager - Draw: 121221112
13:12:25.955 [main] INFO org.apache.logging.log4j.LogManager - Human: 201020112
13:12:25.958 [main] INFO org.apache.logging.log4j.LogManager - Human: 021020121
13:12:25.961 [main] INFO org.apache.logging.log4j.LogManager - Human: 201020112
13:12:25.964 [main] INFO org.apache.logging.log4j.LogManager - Human: 101222211
13:12:25.968 [main] INFO org.apache.logging.log4j.LogManager - Human: 201121212
13:12:25.971 [main] INFO org.apache.logging.log4j.LogManager - Human: 020120121
13:12:25.974 [main] INFO org.apache.logging.log4j.LogManager - Draw: 112211122
13:12:25.979 [main] INFO org.apache.logging.log4j.LogManager - Draw: 211122211
13:12:25.983 [main] INFO org.apache.logging.log4j.LogManager - Menace: 101120122
Git Find Run TODO Problems Debug Dependencies Terminal Build
Build completed successfully in 2 sec, 260 ms (3 minutes ago)
Command Prompt
Microsoft Windows [Version 10.0.22000.556]
(c) Microsoft Corporation. All rights reserved.
C:\Users\urvan>hostname
UrvangPatel


Run: MenaceBegin
13:18:02.717 [main] INFO org.apache.logging.log4j.LogManager - Menace played 0, 0
13:18:02.717 [main] INFO org.apache.logging.log4j.LogManager - Board state 100020010
13:18:02.718 [main] INFO org.apache.logging.log4j.LogManager - Board mirror found 100020012->001020210
13:18:02.718 [main] INFO org.apache.logging.log4j.LogManager - Board state 001020210
13:18:02.718 [main] INFO org.apache.logging.log4j.LogManager - Menace played 0, 0
13:18:02.718 [main] INFO org.apache.logging.log4j.LogManager - Board state 101020210
13:18:02.718 [main] INFO org.apache.logging.log4j.LogManager - Board mirror found 121020210->001122201
13:18:02.718 [main] INFO org.apache.logging.log4j.LogManager - Board state 001122201
13:18:02.718 [main] INFO org.apache.logging.log4j.LogManager - Menace played 0, 0
13:18:02.718 [main] INFO org.apache.logging.log4j.LogManager - Board state 101122201
13:18:02.719 [main] INFO org.apache.logging.log4j.LogManager - Board state 121122201
13:18:02.719 [main] INFO org.apache.logging.log4j.LogManager - Menace played 2, 1
13:18:02.719 [main] INFO org.apache.logging.log4j.LogManager - Draw: 121122211
13:18:02.719 [main] INFO org.apache.logging.log4j.LogManager - Board state 000000000
13:18:02.719 [main] INFO org.apache.logging.log4j.LogManager - Menace played 2, 1
13:18:02.719 [main] INFO org.apache.logging.log4j.LogManager - Board state 000000010
13:18:02.720 [main] INFO org.apache.logging.log4j.LogManager - Board mirror found 200000010->000001200
13:18:02.720 [main] INFO org.apache.logging.log4j.LogManager - Board state 000001200
13:18:02.720 [main] INFO org.apache.logging.log4j.LogManager - Menace played 1, 0
13:18:02.720 [main] INFO org.apache.logging.log4j.LogManager - Board state 000101200
13:18:02.721 [main] INFO org.apache.logging.log4j.LogManager - Board mirror found 000121200->000121002
13:18:02.721 [main] INFO org.apache.logging.log4j.LogManager - Board state 000121002
13:18:02.721 [main] INFO org.apache.logging.log4j.LogManager - Menace played 0, 1
13:18:02.721 [main] INFO org.apache.logging.log4j.LogManager - Board state 010121002
13:18:02.721 [main] INFO org.apache.logging.log4j.LogManager - Human: 210121002
13:18:02.725 [main] INFO org.apache.logging.log4j.LogManager - Result: Games Played: 500, Human Wins: 218, Menace Wins: 37, Draw Games: 245
Git Find Run TODO Problems Debug Dependencies Terminal Build
Build completed successfully in 1 sec, 805 ms (moments ago)
Command Prompt
Microsoft Windows [Version 10.0.22000.556]
(c) Microsoft Corporation. All rights reserved.
C:\Users\urvan>hostname
UrvangPatel
C:\Users\urvan>

Config	Runs
Alpha=10 Beta=3 Gama=1 Delta=2 P=0.9	
Alpha=10 Beta=3 Gama=1 Delta=2 P=0.5	
Alpha=10 Beta=3 Gama=8 Delta=2 P=0.9	
Alpha=10 Beta=3 Gama=8 Delta=2 P=0.5	

Alpha=10 Beta=10 Gama=1 Delta=2 P=0.9	
Alpha=10 Beta=10 Gama=1 Delta=2 P=0.5	

Conclusions:

Based on the graphs, when the Menace plays with the flawlessly playing human, the menace loses in the beginning, but later after around 100 games, Menace learns to block a move to make the game “draw” or “win”.
Based on the Human’s random strategy, Menace produces a positive trend after specific no. of games. The positive trend is, it learns to block a move or tries to win by opting a win-win spot. Whereas, based on the Human’s optimal strategy, the response or chances of wins reduces but eventually (after ~95 games) it learns to block a win-win state.

References:

2016, O. C. M., Child, O., & Oliver ChildOliver Child is a high school student living in Brussels who is interested in all kinds of maths and computing. Oliver was first introduced to Menace by Matthew Scroggs. (2016, October 25). Menace: The machine educable noughts and crosses engine. Chalkdust. Retrieved April 27, 2022, from https://chalkdustmagazine.com/features/menace-machine-educable-noughts-crosses-engine/

How 300 matchboxes learned to play tic-tac-toe using menace. Open Data Science - Your News Source for AI, Machine Learning & more. (2019, June 26). Retrieved April 27, 2022, from https://opendatascience.com/menace-donald-michie-tic-tac-toe-machine-learning/

Sall, M. (2019, March 25). Teaching 304 matchboxes to beat you at tic-tac-toe. Bell of Lost Souls. Retrieved April 27, 2022, from https://www.belloflostsouls.net/2019/03/teaching-304-matchboxes-to-beat-you-at-tic-tac-toe.html

Scroggs, M. (2015, August 27). Menace. mscroggs.co.uk Blog: MENACE. Retrieved April 27, 2022, from https://www.mscroggs.co.uk/blog/19

Wikimedia Foundation. (2021, October 9). Matchbox Educable Noughts and crosses engine. Wikipedia. Retrieved April 27, 2022, from https://en.wikipedia.org/wiki/Matchbox_Educable_Noughts_and_Crosses_Engine