INFO 6205 SPRING 2022 PROJECT
MENACE

Vidhi Rajesh Popat (NUID: 002135187)
Akshay Kumthi Matad (NUID: 002928833)
Vraj Himanshu Reshamdalal (NUID:002927484)

INTRODUCTION:

AIM

Implement "The Menace" by replacing matchboxes with values in a hash table (key will be the state of the game). Train the Menace by running games played against "human" strategy, which is based upon optimal strategy.

APPROACH

The maximum number of moves the bot can play is 4.
Human always plays first in our game, denotes as "X".
We use a different HastTable for each move played in the game.
We store states of the game and its respective points into HashTables as Key,Value pairs(where Key = State and Value = Scores of indexes).
The HashTable represents the match boxes in the video.
The beads used to reward or punish the Menace are:

- Alpha = 0
- Beta = +3
- Gamma = -1
- Delta = +2

We have created a function that plays the optimal strategy.
We make the optimal strategy to play with the Menace for 150,000 times in order to train it.

PROGRAM:

DATA STRUCTURE AND CLASSES

Data structures –

Hashtables

Customized class (Score.java)

Classes –

class Trainer

- private void markPosition() {} - This method allows two bots(Menace and Optimal) to play against each other
- public boolean check() {} - checks for Win/Loss
- public boolean checkfordraw() {} - check for draw
- public void xWins(int a, int b, int c) {} - action performed when O wins
- public void itsdraw() {} - action performed when draw
- public void xWins(int a, int b, int c) {} - action performed when X wins
- public List<Integer> getAllEmptySpacesOnBoard(JButton[] buttons) {} - get all emptySpaces on the board currently
- public boolean boardFull() {} - checks if board is Full

class TrainedMenace

- public Hashtable<String, Hashtable…) {} - Returns if state exists else it will create a new state in the HashTable
- public int getMaxValuefromInner(Hashtable<String, Hashtable…) {} - Gets the index of the max value from given HashTable
- public ParentHT menacemove(ParentHT pHT, JButton[]...) {} - returns the next move of the Menace; It will return index/move,ranks and playedPositions
- public void insertRewards(ParentHT pHT, String s…) {} - insert points to the HashTable based on Win/Loss at the end of a single game.

class Score - A class to store points into HashTable

class ParentHT - A container to store Menace moves, board position

class HumanStrategy -

- public void humanBotMove(Integer[] board) {} - Plays the game as per the Optimal Strategy.

class EasyMenace

- public int easyMove(JButton[] buttons) {} - Generates random board positions(that are empty) to play.

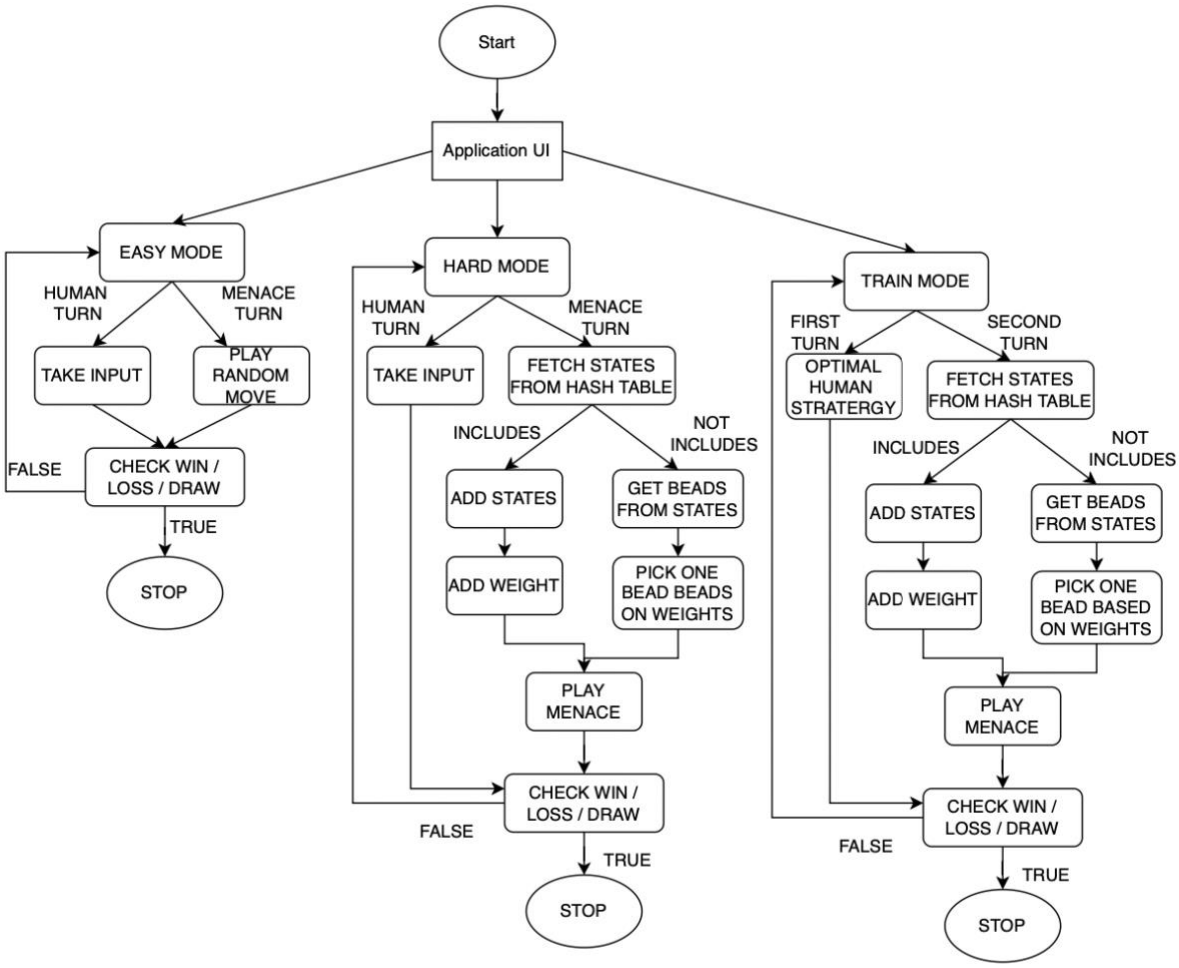
ALGORITHM

- We are making hash tables for each bot move, saying the bot can maximum play 4 times hence 4 hash tables. Let's call this hash table as ht1, ht2, ht3, ht4.
- We have created a custom class Score which will store board position and points (reward + punishment + draw). We will store the rewards in the object of the Score class and then we store this object's value to ht1, ht2, ht3, ht4.
- There will be one hash table which will have states of the board at each move (this is like matchboxes from a menace game). Basically, this hash table will have key as states and value as another hash tables whose key is board position and value as rewards (this is like number of beads in each matchbox), this hash table will be ht1, ht2, ht3 or ht4 depending on the move number.
- We have implemented an optimal human strategy to play against bot at every move, check if the opponent is winning if yes then block the winning position, also check if human strategy is winning then play that position. If none of the winning conditions satisfies then play the center first if it is empty, then play corners if it is empty, then play edges if it is empty. We added some randomness to human's move 1 , move 2 and move 4 for generating new states and training menace faster.
- Our train bot checks the values from hash tables according to the state at which it is and then gets the maximum weights from hash table).

INVARIANTS

- For a particular state, we will reward or punish particular positions only and other positions will have alpha beads.
- Menace cannot play more than four states in a game.
- Human always plays the first move.

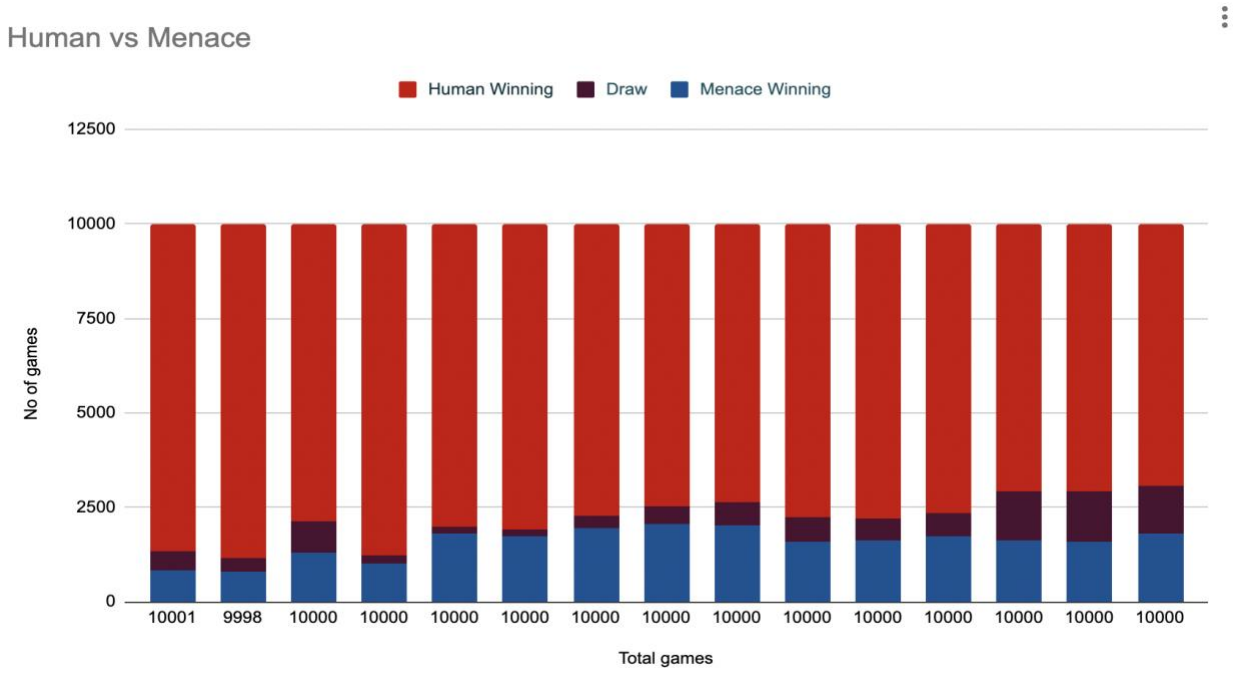
FLOW CHARTS:


Start
Application UI
EASY MODE
HUMAN TURN
MENACE TURN
TAKE INPUT
PLAY RANDOM MOVE
CHECK WIN / LOSS / DRAW
FALSE
TRUE
STOP
HARD MODE
HUMAN TURN
MENACE TURN
TAKE INPUT
FETCH STATES FROM HASH TABLE
INCLUDES
NOT INCLUDES
ADD STATES
GET BEADS FROM STATES
ADD WEIGHT
PICK ONE BEAD BEADS ON WEIGHTS
PLAY MENACE
CHECK WIN / LOSS / DRAW
FALSE
TRUE
STOP
TRAIN MODE
FIRST TURN
SECOND TURN
OPTIMAL HUMAN STRATERGY
FETCH STATES FROM HASH TABLE
INCLUDES
NOT INCLUDES
ADD STATES
GET BEADS FROM STATES
ADD WEIGHT
PICK ONE BEAD BASED ON WEIGHTS
PLAY MENACE
CHECK WIN / LOSS / DRAW
FALSE
TRUE
STOP

OBSERVATION AND GRAPHICAL REPRESENTATION:

- Since we chose **p = 0.85**, the learning process was slow but it was accurate and will get better if we train it more.
- The chances of Menace starts winning or drawing after a certain number of games.
- The number of games Menace wins might not be linear. It might drop a few times.

Graph:



Values:

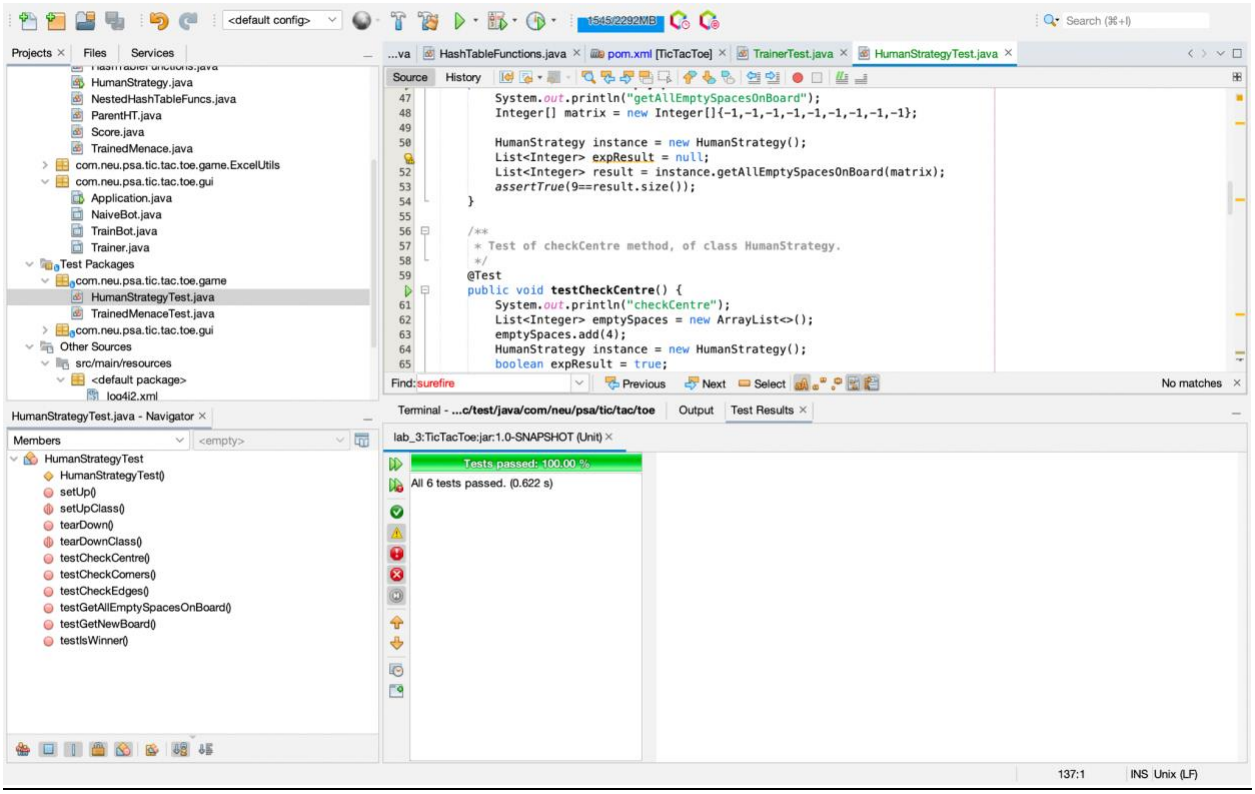
Human Winning	Menace Winning	Draw	Total
8652	817	532	10001
8842	804	352	9998
7880	1284	836	10000
8763	1021	216	10000
8015	1806	179	10000
8099	1720	181	10000
7725	1934	341	10000
7472	2070	458	10000
7374	2009	617	10000
7744	1595	661	10000
7787	1631	582	10000
7642	1721	637	10000
7084	1623	1293	10000
7085	1592	1323	10000
6946	1800	1254	10000

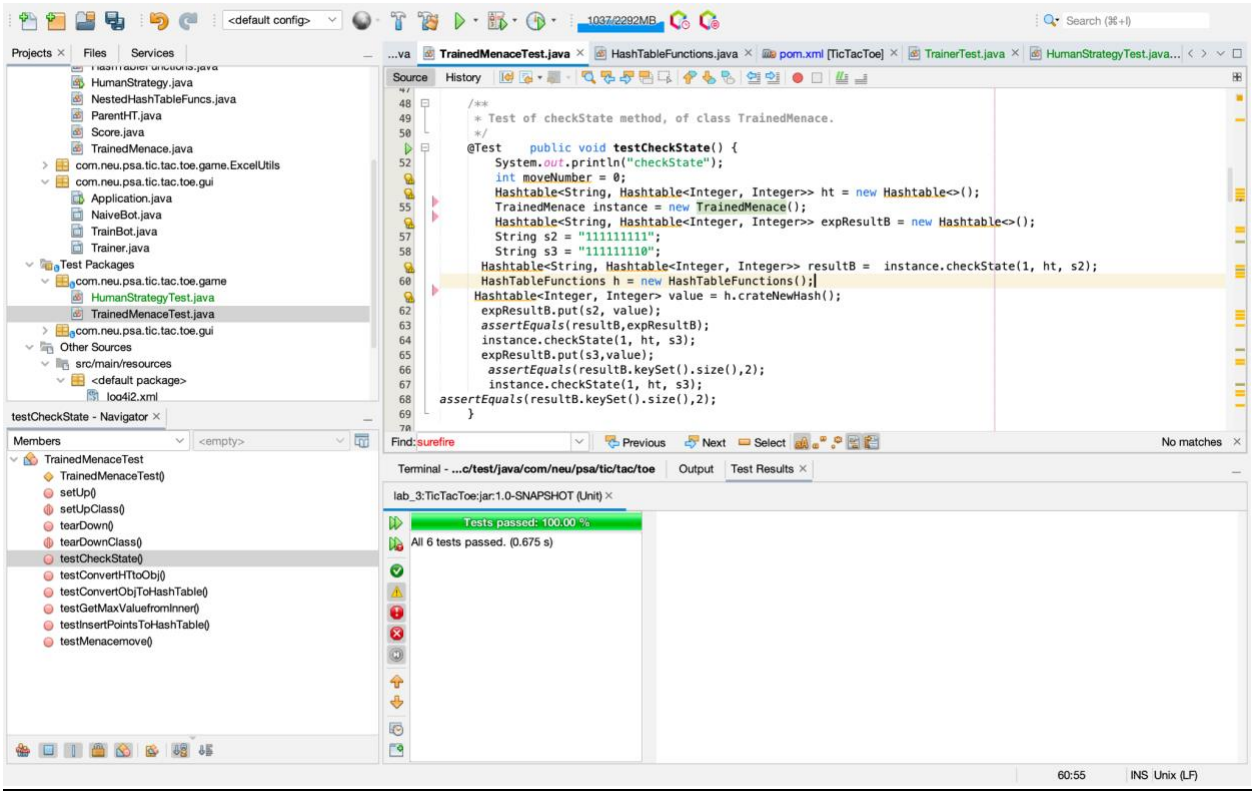
RESULTS AND MATHEMATICAL ANALYSIS:

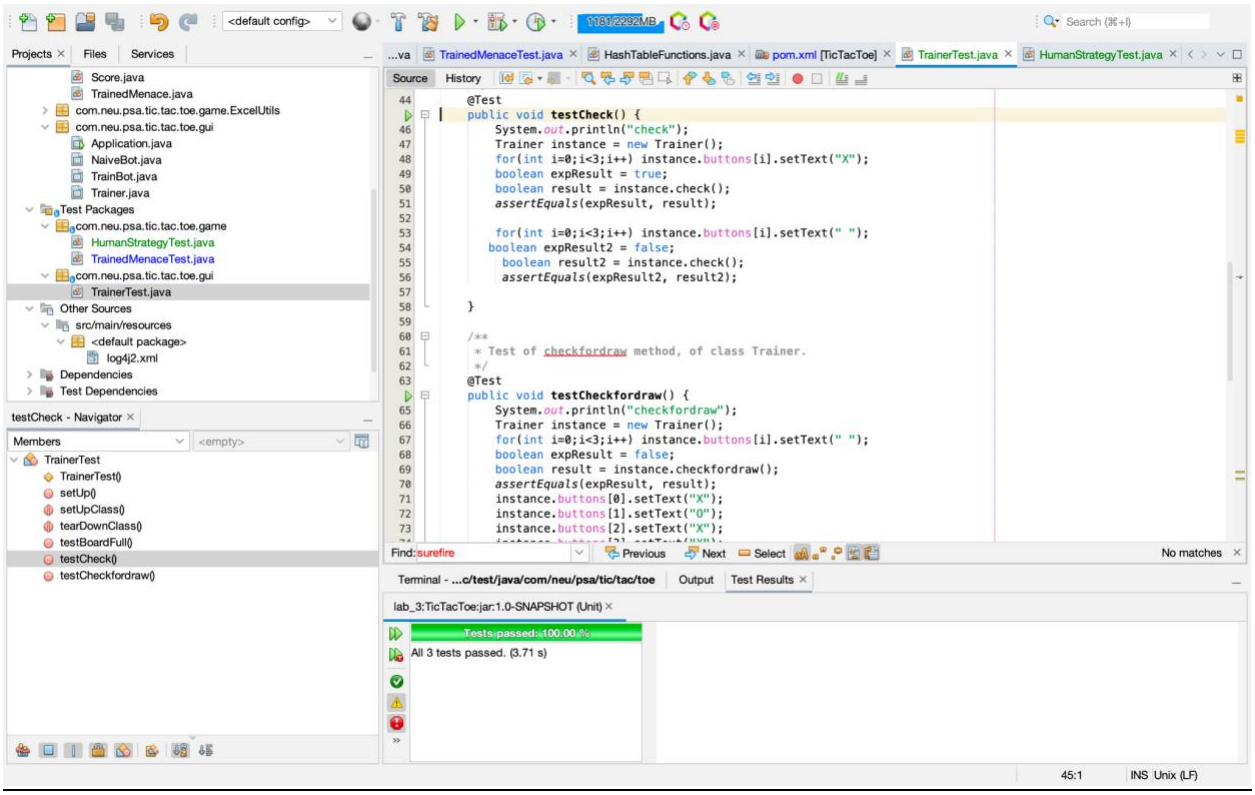
- We have checked for the first 1000 games and our Human Strategy was winning 85% of the games as per our p value.
- In the first 10000 games, the Human Strategy won ~8600 games which implies it had not learnt anything but it got better as the number of games played increased.
- The Human strategy winning percentage started dropping gradually. At the starting of the game it won ~85% of the games but at the end of our iterations it won only ~69% of the games.

Human Winning	Menace Winning	Draw	Total	Percentage of Human Winning
8652	817	532	10001	86.51134887
8842	804	352	9998	88.43768754
7880	1284	836	10000	78.8
8763	1021	216	10000	87.63
8015	1806	179	10000	80.15
8099	1720	181	10000	80.99
7725	1934	341	10000	77.25
7472	2070	458	10000	74.72
7374	2009	617	10000	73.74
7744	1595	661	10000	77.44
7787	1631	582	10000	77.87
7642	1721	637	10000	76.42
7084	1623	1293	10000	70.84
7085	1592	1323	10000	70.85
6946	1800	1254	10000	69.46

TESTCASES:

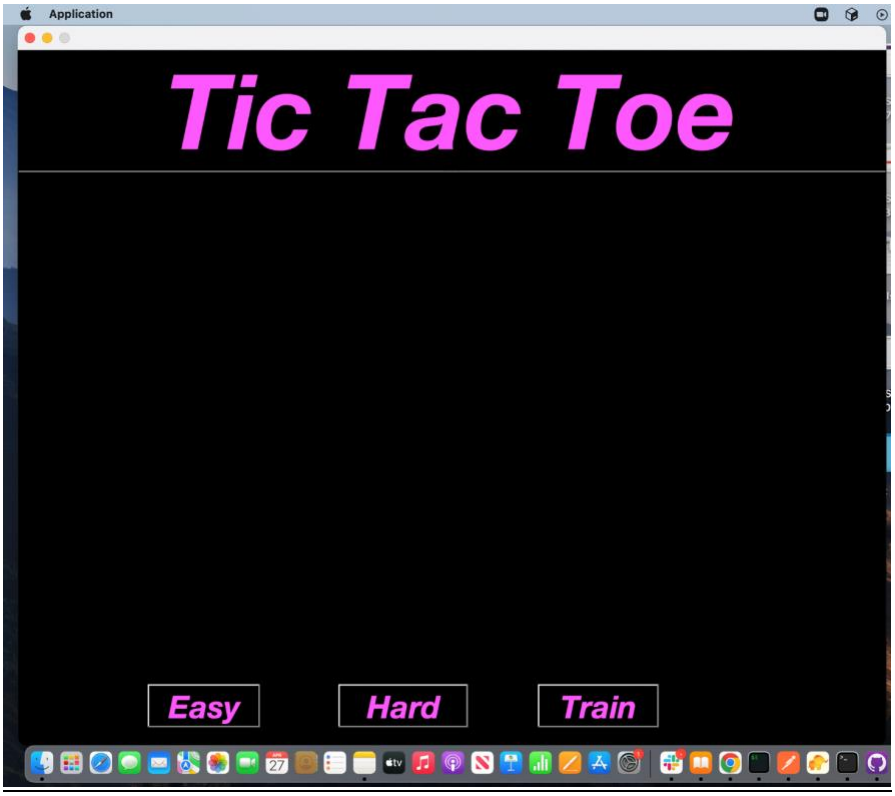

HumanStrategyTest.java
HumanStrategy.java
NestedHashTableFuncs.java
ParentHT.java
Score.java
TrainedMenace.java
Application.java
NaiveBot.java
TrainBot.java
Trainer.java
HumanStrategyTest.java
TrainedMenaceTest.java
HumanStrategyTest.java - Navigator
HumanStrategyTest()
setUp()
setUpClass()
tearDown()
tearDownClass()
testCheckCentre()
testCheckCorners()
testCheckEdges()
testGetAllEmptySpacesOnBoard()
testGetNewBoard()
testIsWinner()
Tests passed: 100.00 %
All 6 tests passed. (0.622 s)

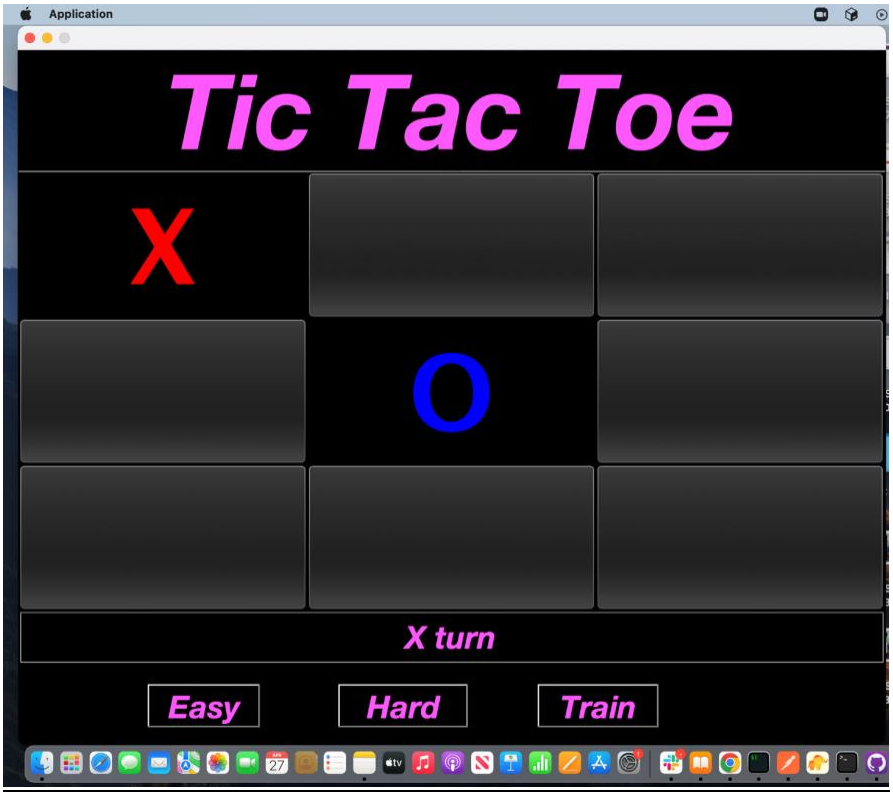

TrainedMenaceTest.java
HumanStrategy.java
NestedHashTableFuncs.java
ParentHT.java
Score.java
TrainedMenace.java
Application.java
NaiveBot.java
TrainBot.java
Trainer.java
HumanStrategyTest.java
TrainedMenaceTest.java
testCheckState - Navigator
TrainedMenaceTest()
setUp()
setUpClass()
tearDown()
tearDownClass()
testCheckState()
testConvertHTtoObj()
testConvertObjToHashTable()
testGetMaxValuefromInner()
testInsertPointsToHashTable()
testMenacemove()
Tests passed: 100.00 %
All 6 tests passed. (0.675 s)

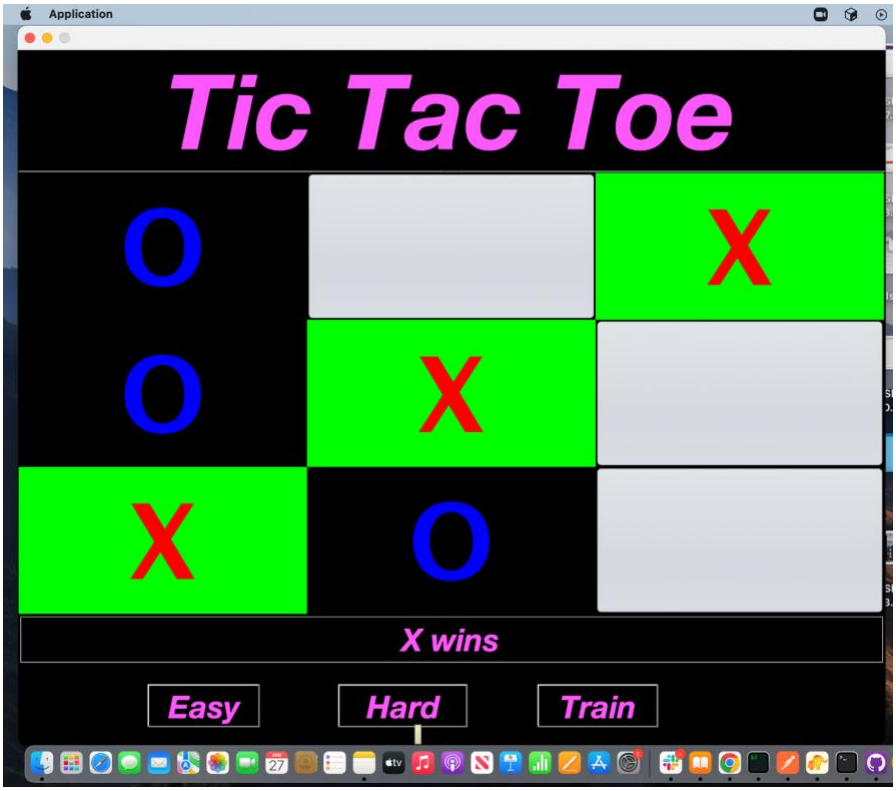

Projects
Files
Services
Score.java
TrainedMenace.java
com.neu.psa.tic.tac.toe.game.ExcelUtils
com.neu.psa.tic.tac.toe.gui
Application.java
NaiveBot.java
TrainBot.java
Trainer.java
Test Packages
com.neu.psa.tic.tac.toe.game
HumanStrategyTest.java
TrainedMenaceTest.java
com.neu.psa.tic.tac.toe.gui
TrainerTest.java
Other Sources
src/main/resources
<default package>
log4j2.xml
Dependencies
Test Dependencies
testCheck - Navigator
Members
<empty>
TrainerTest
TrainerTest()
setUp()
setUpClass()
tearDownClass()
testBoardFull()
testCheck()
testCheckfordraw()
TrainedMenaceTest.java
HashTableFunctions.java
pom.xml [TicTacToe]
TrainerTest.java
HumanStrategyTest.java
Source
History
@Test
public void testCheck() {
System.out.println("check");
Trainer instance = new Trainer();
for(int i=0;i<3;i++) instance.buttons[i].setText("X");
boolean expResult = true;
boolean result = instance.check();
assertEquals(expResult, result);
for(int i=0;i<3;i++) instance.buttons[i].setText(" ");
boolean expResult2 = false;
boolean result2 = instance.check();
assertEquals(expResult2, result2);
}
/**
* Test of checkfordraw method, of class Trainer.
*/
@Test
public void testCheckfordraw() {
System.out.println("checkfordraw");
Trainer instance = new Trainer();
for(int i=0;i<3;i++) instance.buttons[i].setText(" ");
boolean expResult = false;
boolean result = instance.checkfordraw();
assertEquals(expResult, result);
instance.buttons[0].setText("X");
instance.buttons[1].setText("O");
instance.buttons[2].setText("X");
Find: surefire
Previous
Next
Select
No matches
Terminal - ...c/test/java/com/neu/psa/tic/tac/toe
Output
Test Results
lab_3:TicTacToe:jar:1.0-SNAPSHOT (Unit)
Tests passed: 100.00 %
All 3 tests passed. (3.71 s)
45:1
INS
Unix (LF)

OUTPUT:

Training logs can be found here:
https://github.com/popatvidhi/PSA_Final_Project/blob/main/reports/data/application-20220427-logs.txt




Application
Tic Tac Toe
X
O
X turn
Easy
Hard
Train


Application
Tic Tac Toe
O
X
O
X
X
O
X wins
Easy
Hard
Train

CONCLUSION:

Every game menace plays four moves or less than four moves. For every 50,000 games the chance of the menace winning or tieing increases by 5% (i.e the menace learns)

REFERENCES:

- https://docs.oracle.com/en/java/
- https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html
- https://logging.apache.org/log4j/2.x/javadoc.html
- https://github.com/popatvidhi/PSA_Final_Project/blob/main/reports/data/INFO_6205_Menace_Data.xlsx
- https://github.com/popatvidhi/PSA_Final_Project/tree/main/src/test/java/com/neu/psa/tic/tac/toe