INFO 6205 Spring 2022 Project

Menace

Submitted By:

Manoj Reddy Amireddy (002196218)

Anshita Verma (001007320)

Varun Vupalla

Introduction

Aim:

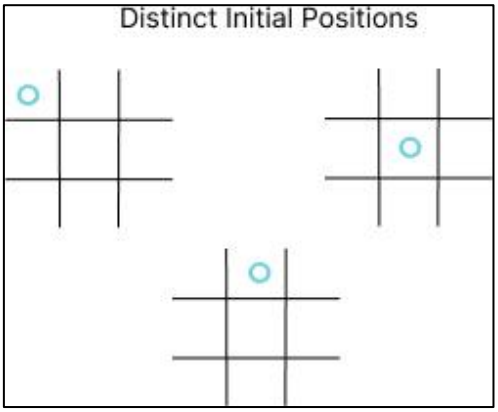
•Implement “The Menace” by replacing matchboxes with values in a hash table (key will be the state of the game).

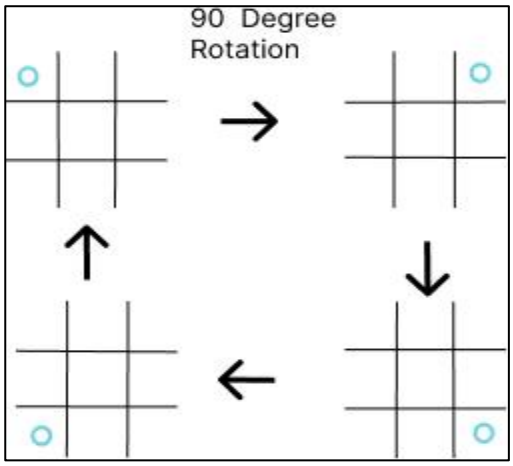
•Train the Menace by running games played against “human” strategy, which is based upon optimal strategy (see Tic-tac-toe).

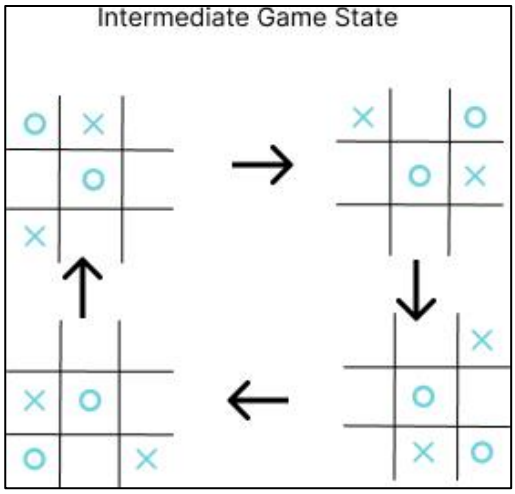
Approach:

1. We are training the system itself to get better as it plays the game. In this way it can improve itself in making the next move.
2. Initially we are generating all the possible stages of the game and storing them.

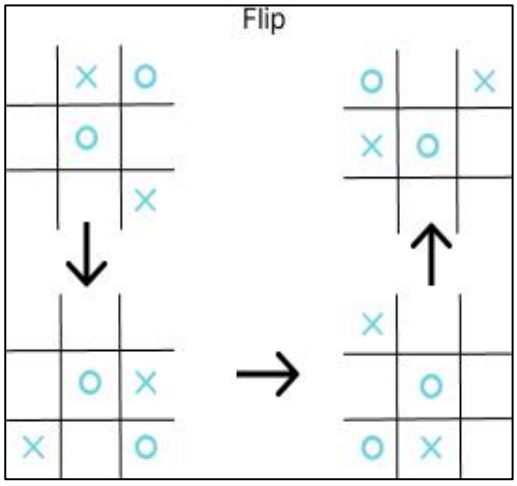
3. These stages may have duplicates, since rotating the board and flipping the board results in the same state.







.



4. So, we remove the duplicates by checking with the other states. Finally, we will be left with all the unique states that are possible to occur while playing the game.
5. Once we have the states, we finally use them to represent as a Matchbox in our system.
6. Now for each state we get the possible next moves that it can make and place that (Beads) in the Matchbox with an initial equal no of them for giving equal probabilities for each move.
7. To train the system, we will make the system to play against a Random gamer.
8. This we take as an input form the user to run the desired no of games to play for training.

9. The training games happen alternatively, so that Random gamer and System makes the 1st move.
10. The Random user makes a move by just randomly choosing the empty place.
11. The system makes the move by randomly picking one of the beads in the Matchbox. These beads have different probabilities for each possible move.
12. We keep track of these moves made by System and the state of the game that it referred to make that move.
13. If the game was Over and the System has WON, then we reward the system with adding extra beads to the move in the Matchbox that it made in the game. It improves the probability of the outcome of this move if it ever sees the same stage in later games.
14. If the game is LOST by System, then we remove the beads of that move it made through the game for each state. So that we are trying to reduce the probabilities of the move in later games it plays and sees the same state again.
15. If the game is DRAW. We just do nothing.
16. This way we train the system and finally flush the trained states and their beads to a CSV for later use in playing against the Real Human.
17. The CSV has the contents in the format of the state comma separated with the bead's values for those positions.
18. In order to play against the Real human, we use this CSV file and build the whole Trained menace game states by parsing the values line by line with the state and its beads.
19. After each game with the Real Human, we get the Game status and then update the Beads in the Matchbox to train the System again.
20. We print the probabilities at every stage of making a move by System in the console.

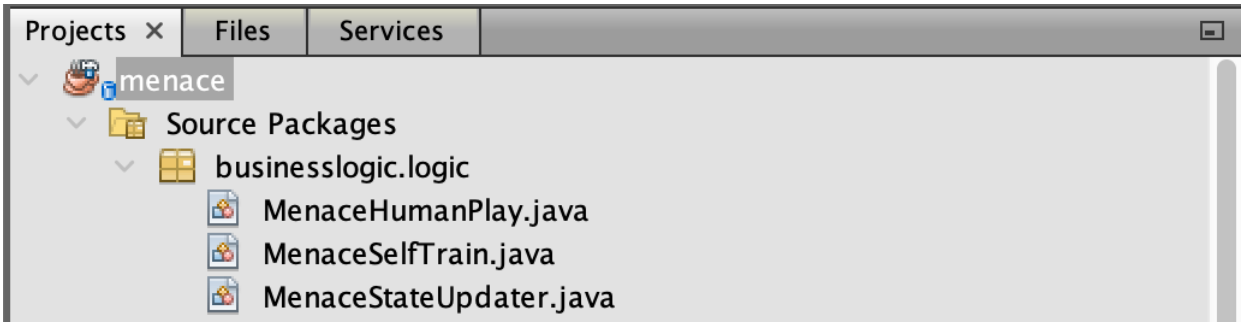
Program

Data Structures and Classes:

DataStructures

1. **Arrays** used to keep track of the state of the game.
2. **HashMap** is used to map the State of the game (Matchbox) to its corresponding Beads statuses.
3. **List** is used to keep the Beads counts in the Matchbox with its position and its counts.

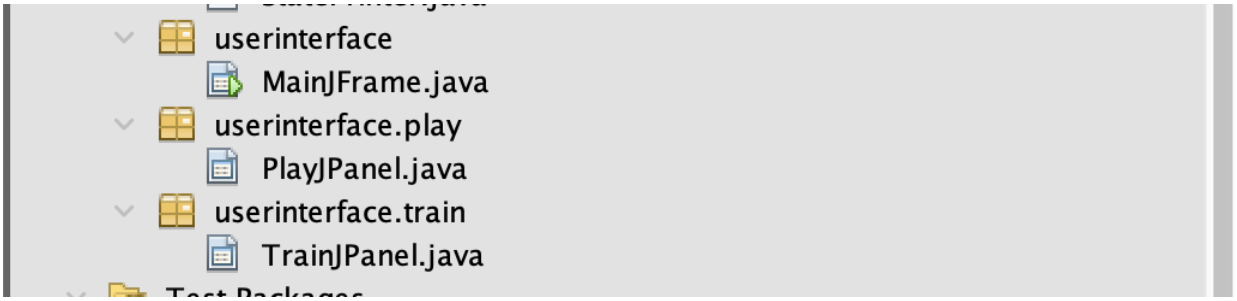
Logic Classes



Model Classes

Util Classes

UI Classes



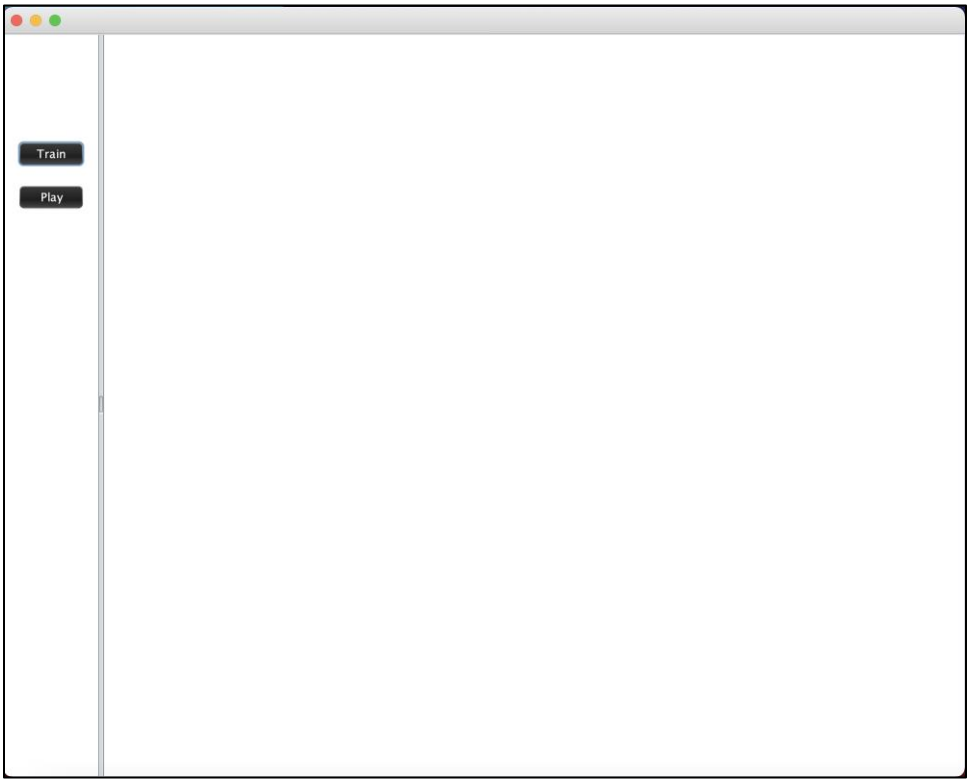
Algorithm:

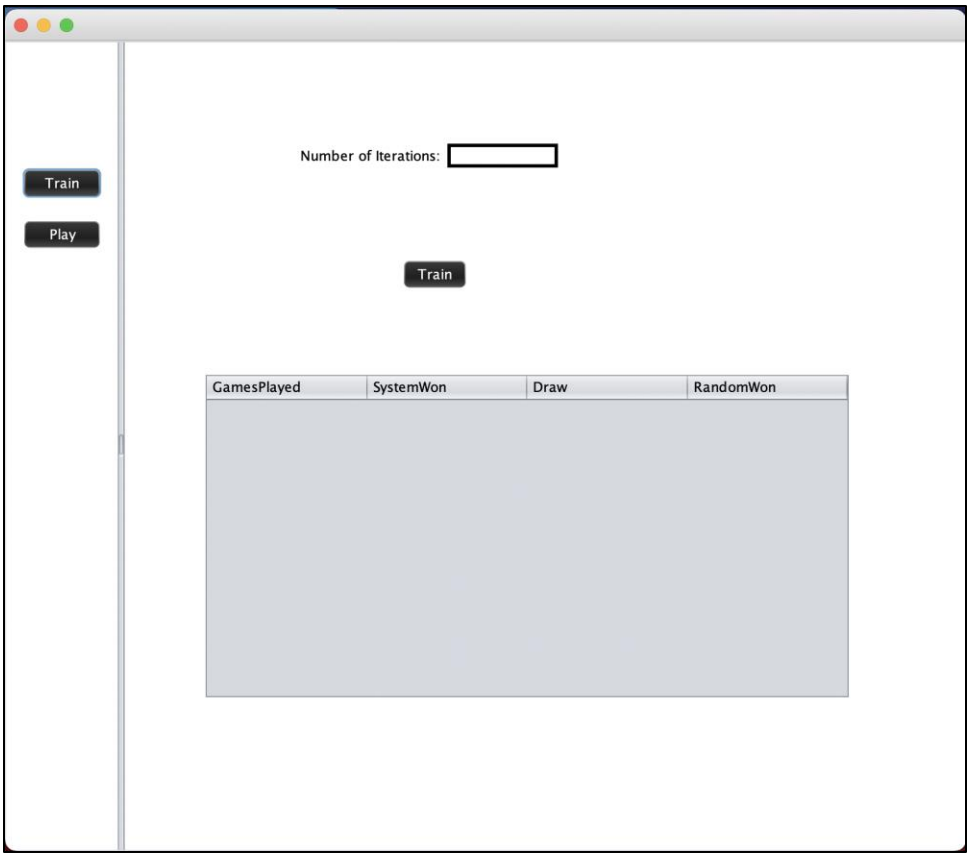
1. We have built a HashMap that maps each state of the Menace Game with the Matchbox. This Matchbox has the beads count for each viable move that it can make.
2. Every move the system makes refers to this Matchbox and gets a random shuffle of the beads. We keep track of this Matchbox and the move it made.
3. Once the game is over. Based on the status whether it's won by a System or not we reward the System by adding extra bead to the Matchbox or remove a bead by using the GAMMA, BETA, DELTA values.
4. After every game the Beads count in the Matchbox changes, and it makes the next move differently than completely random as the counts are different for each viable position.

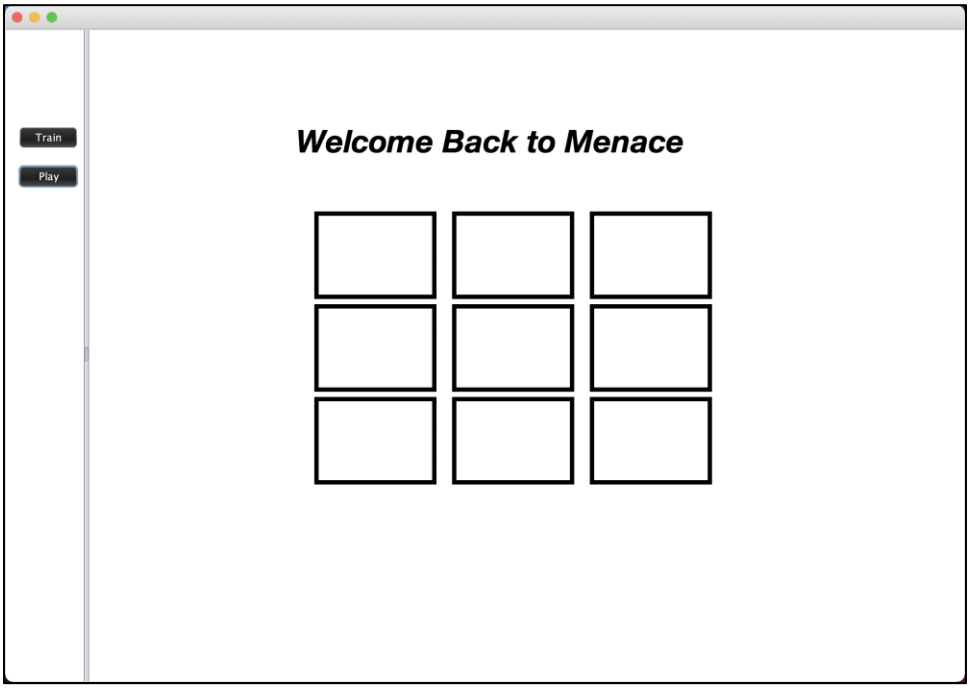
Invariants:

1. At any stage of the game if you flip the game and rotate it clockwise by 90,180,270 degrees. It always results in the same game.
2. Even rotating without flip by 90,180,270 degrees also results in the same game.
3. Also, if a game is either started by the user or the menace, they always refer to the same Matchbox for making the next move. This could remove the initial condition of System to start first always.
4. Whoever starts the game the state of the game will be same as the state is independent of the player.

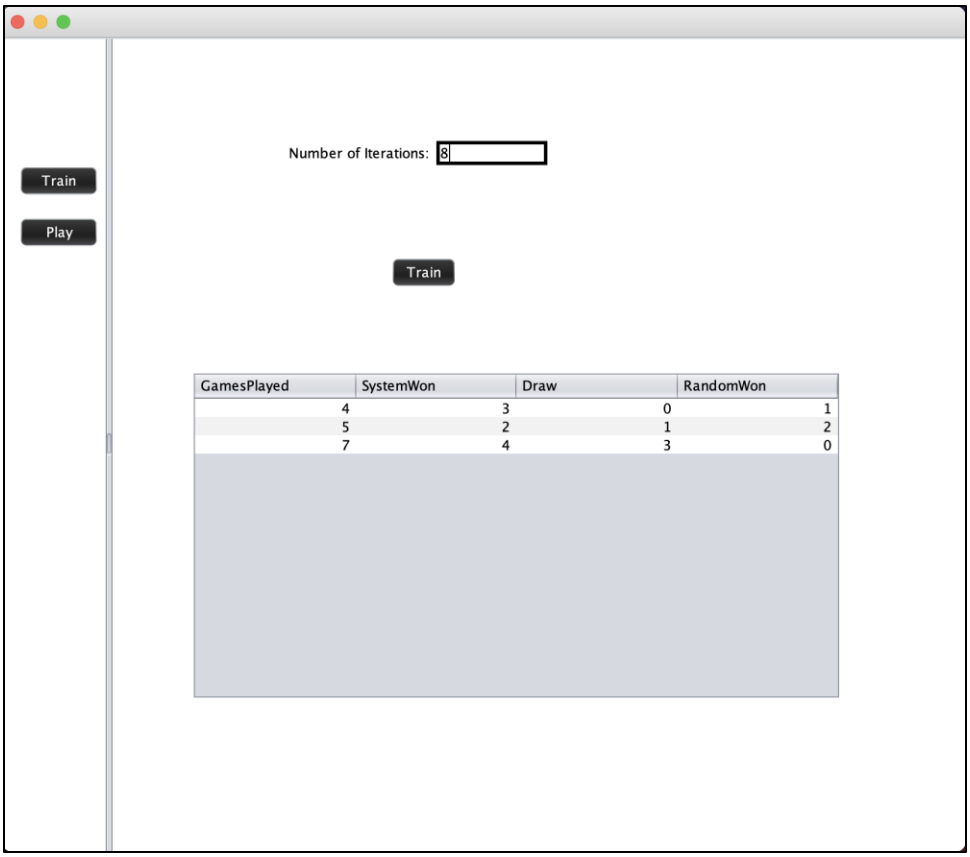
UI Design:



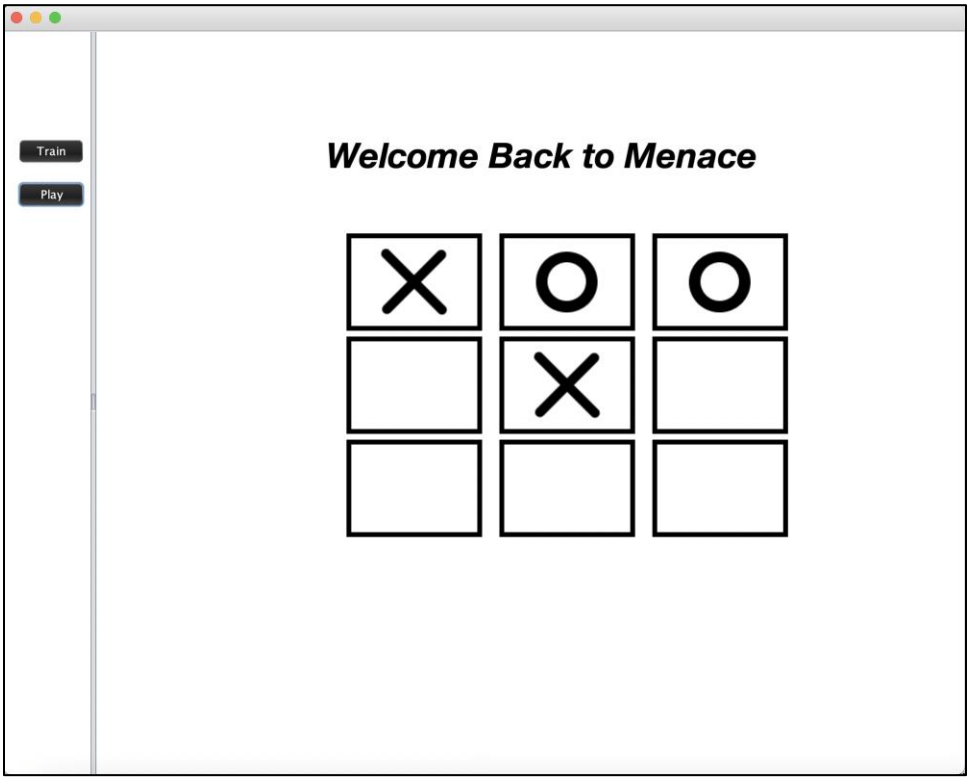




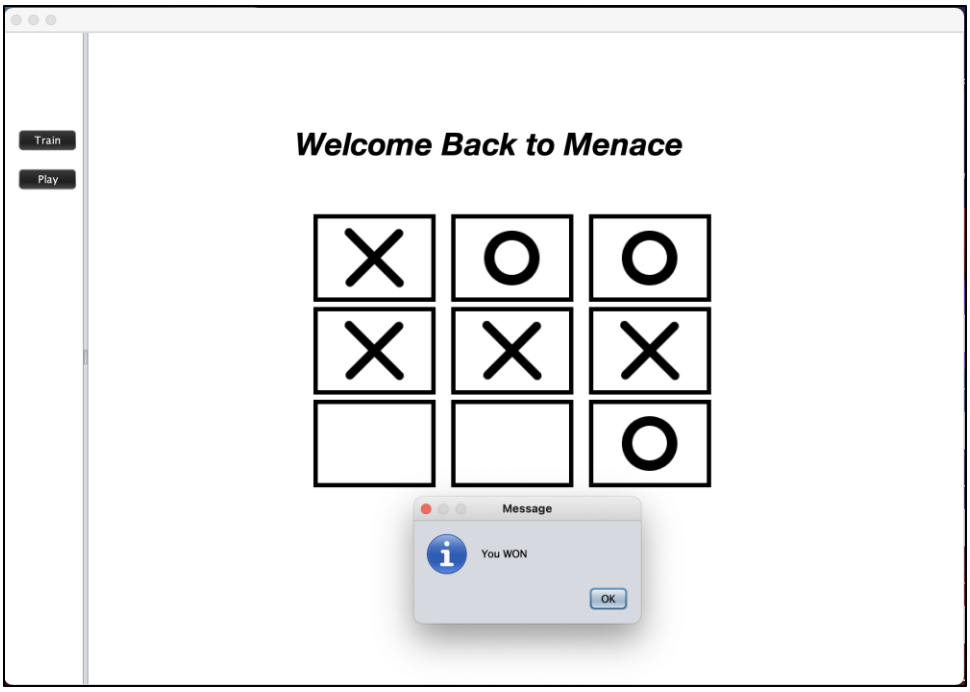
Application allows to train the machine:



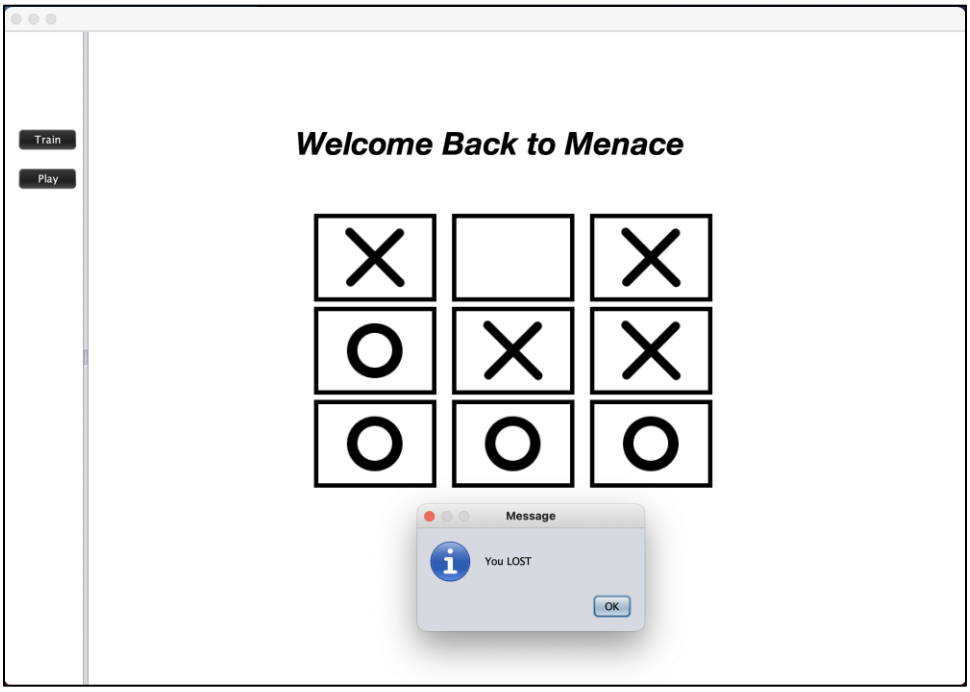
Application asks user to choose to play first move:



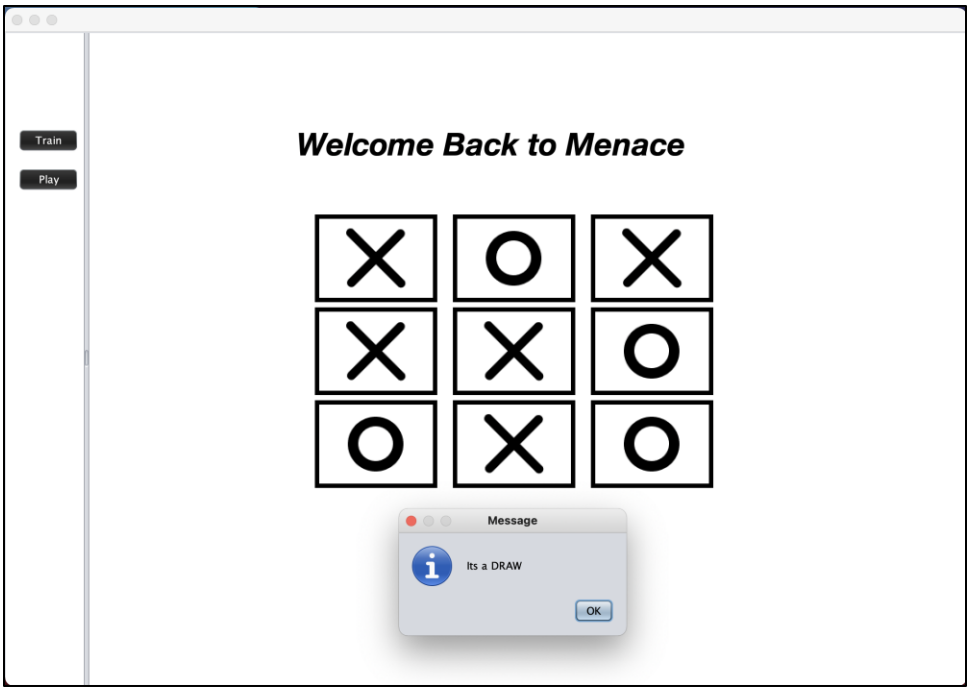
If the user wins:



If the machine wins:

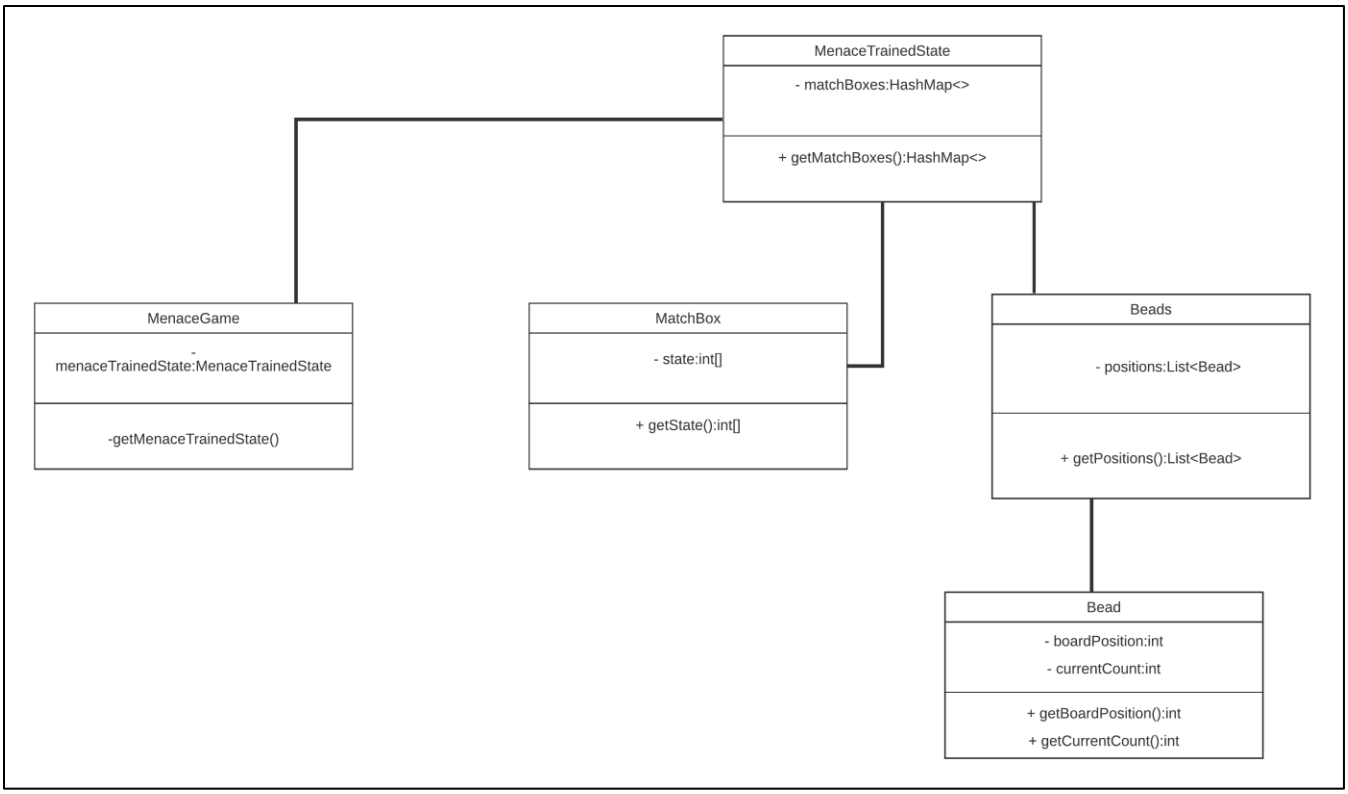


If game results in Draw:

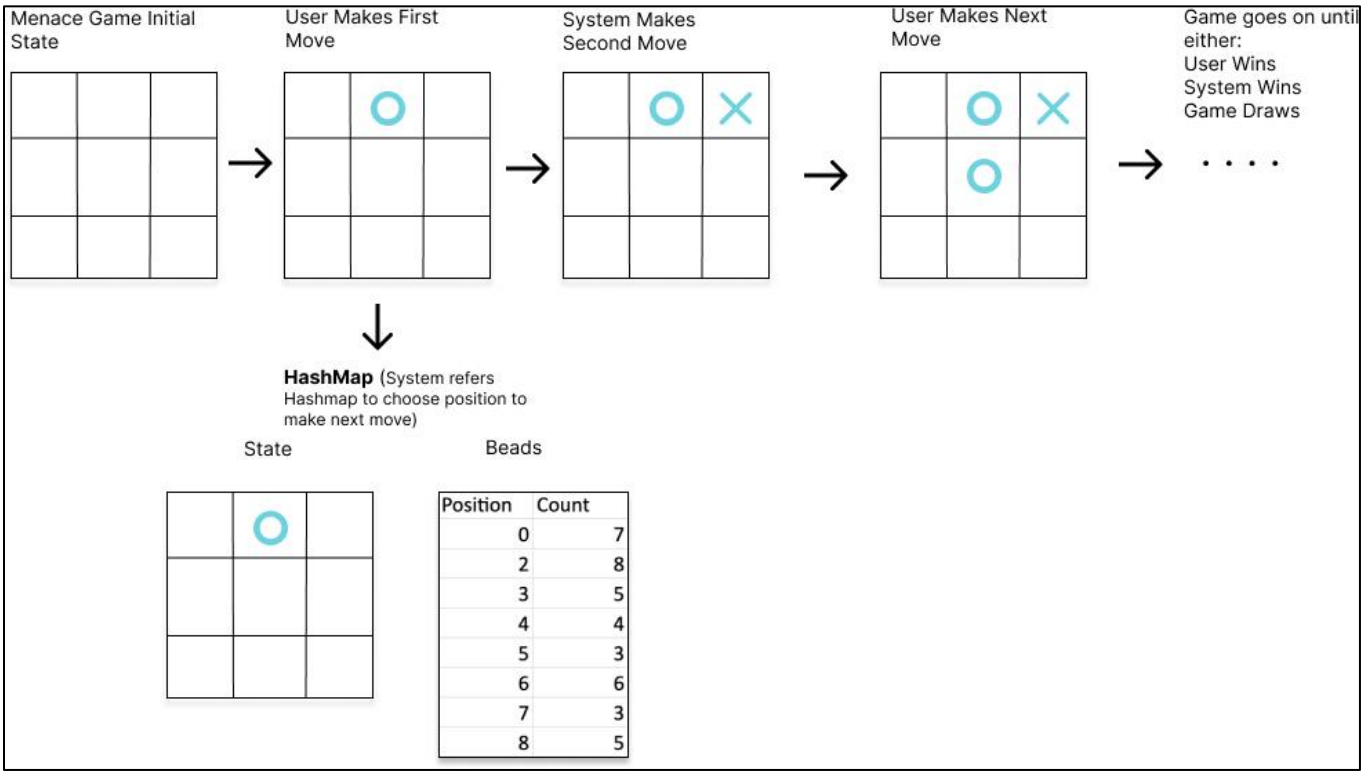


Flow Charts

UML Diagram


MenaceTrainedState
- matchBoxes:HashMap<>
+ getMatchBoxes():HashMap<>
MenaceGame
- menaceTrainedState:MenaceTrainedState
-getMenaceTrainedState()
MatchBox
- state:int[]
+ getState():int[]
Beads
- positions:List<Bead>
+ getPositions():List<Bead>
Bead
- boardPosition:int
- currentCount:int
+ getBoardPosition():int
+ getCurrentCount():int

UI Flow

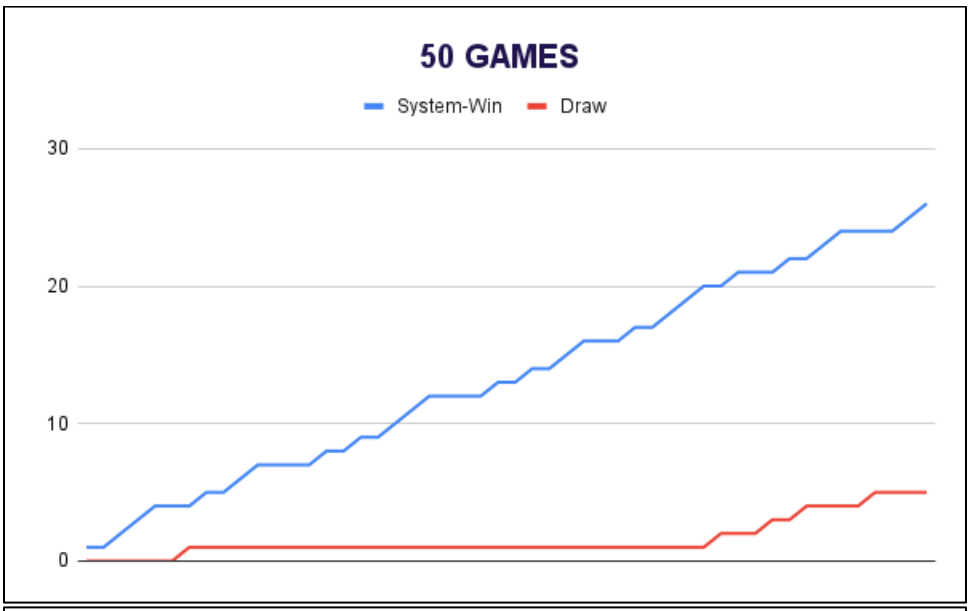

Menace Game Initial State
User Makes First Move
System Makes Second Move
User Makes Next Move
Game goes on until either:
User Wins
System Wins
Game Draws
HashMap (System refers Hashmap to choose position to make next move)
State
Beads
Position Count
0 7
2 8
3 5
4 4
5 3
6 6
7 3
8 5

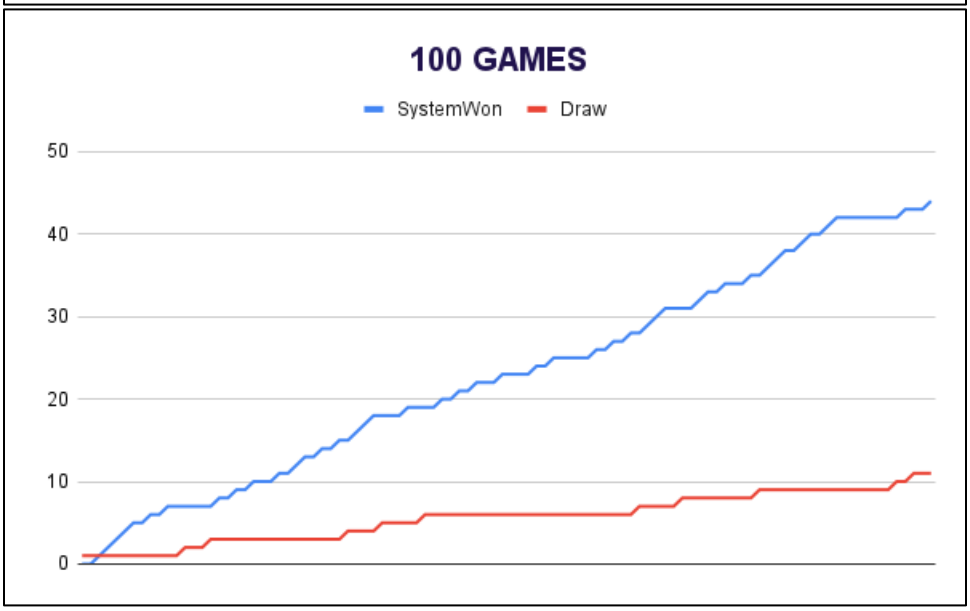
Observation And Graphical Analysis

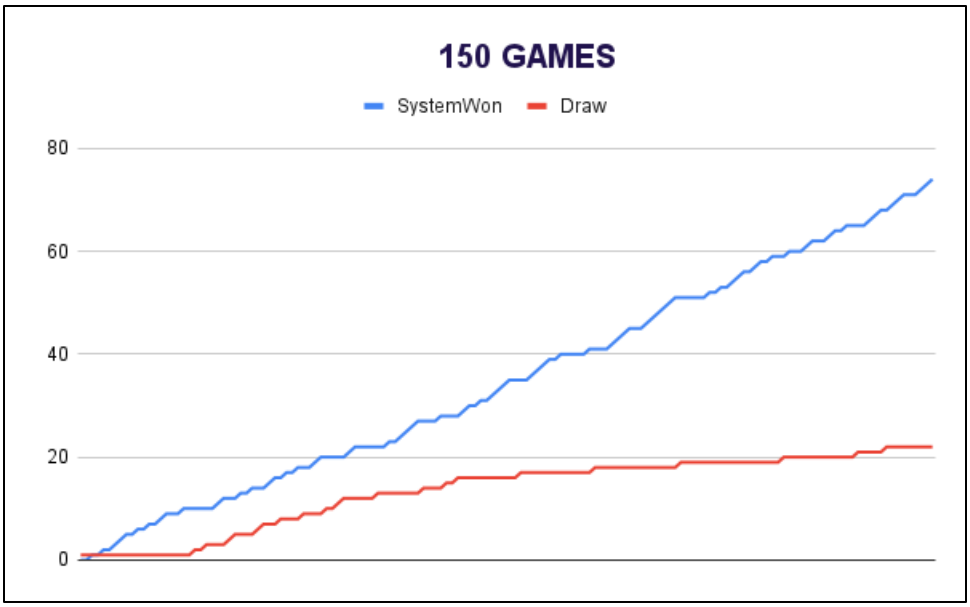
Performed menace game with different values of ALPHA, BETA, GAMMA, DELTA. Trained the machine by playing up to 50, 100 and 150 games.

Below is the graphical analysis for each experiment on Menace Game:

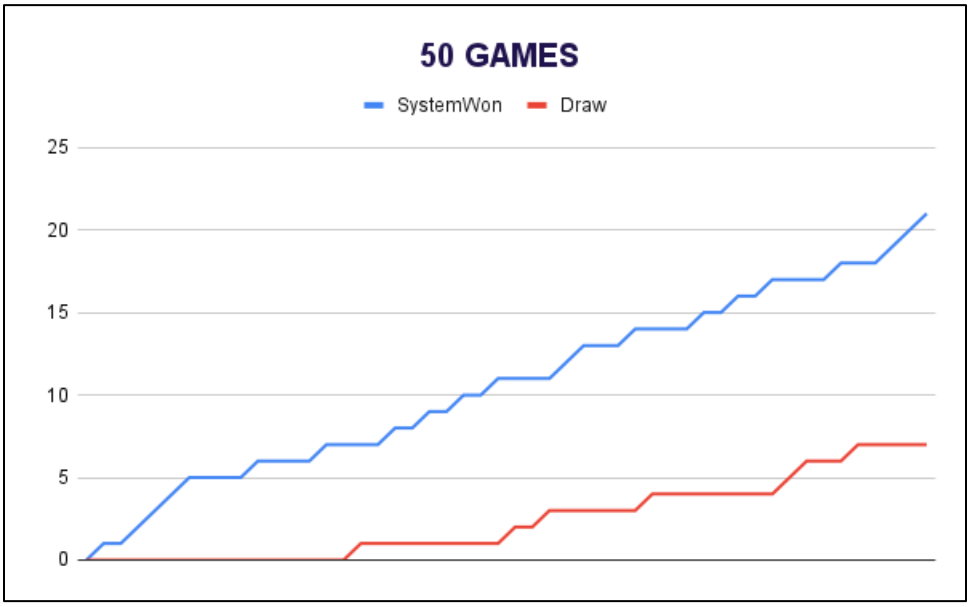
ALPHA: 10 BETA:1 GAMMA:1 DELTA:0

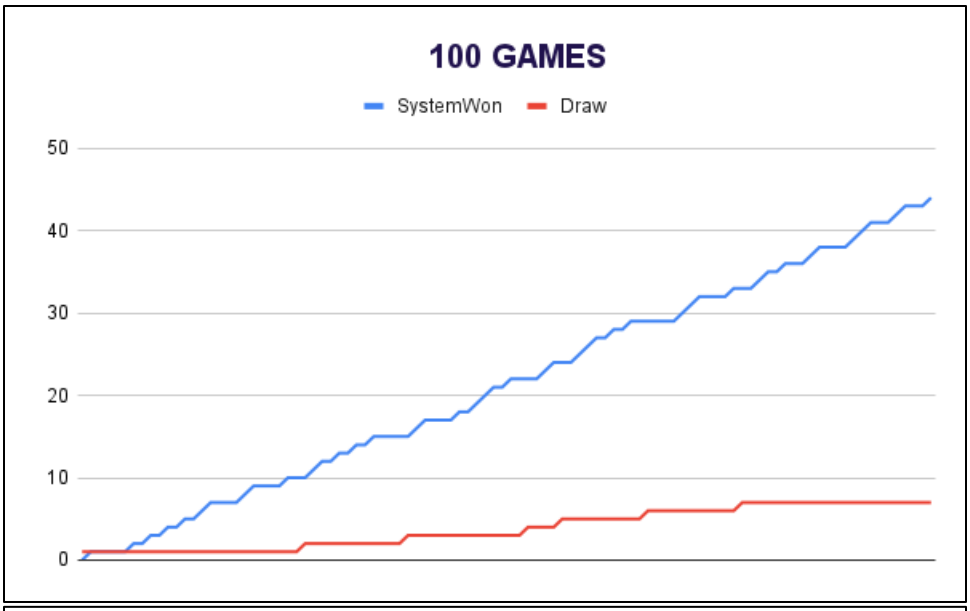


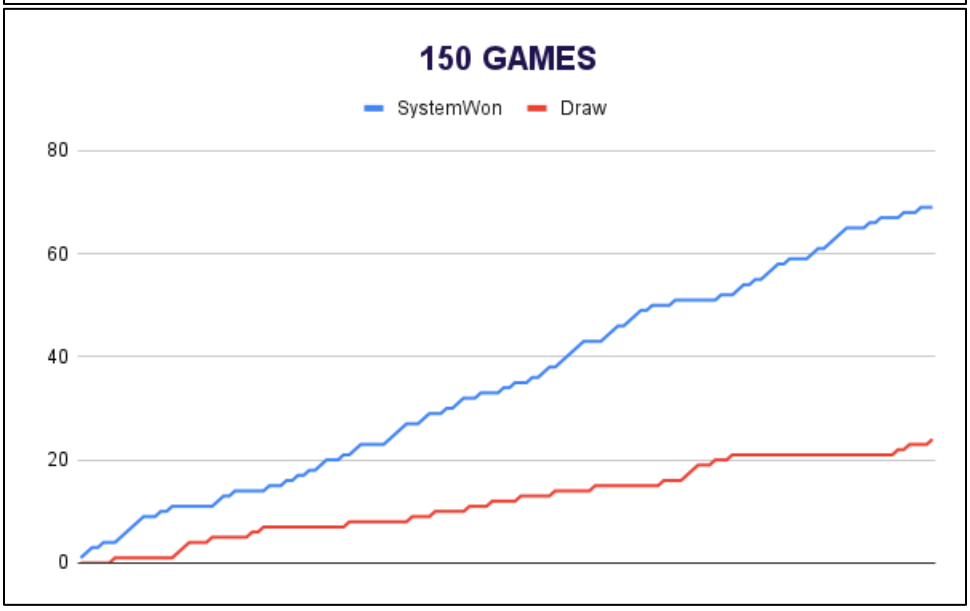




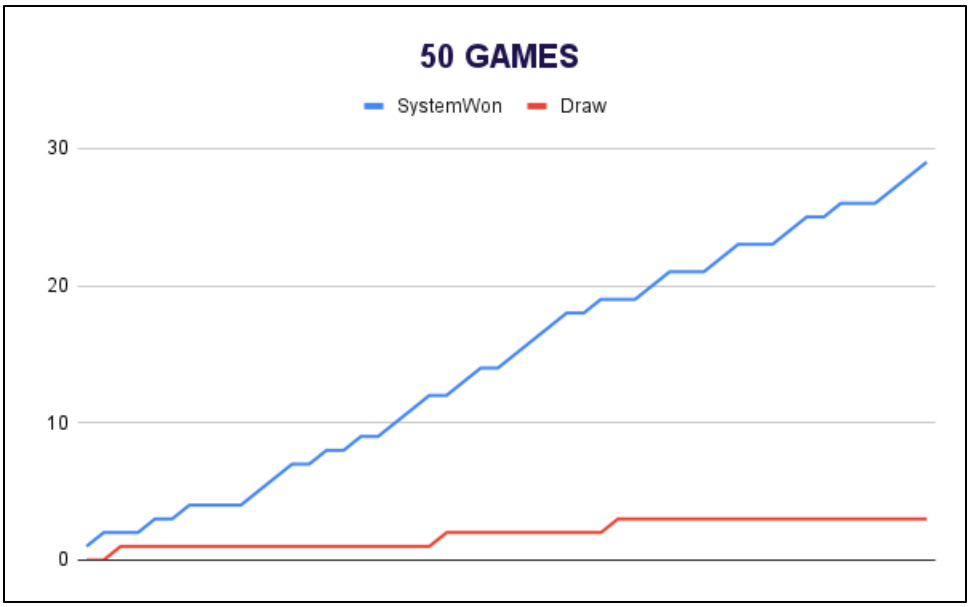
ALPHA: 10 BETA:1 GAMMA:1 DELTA:1

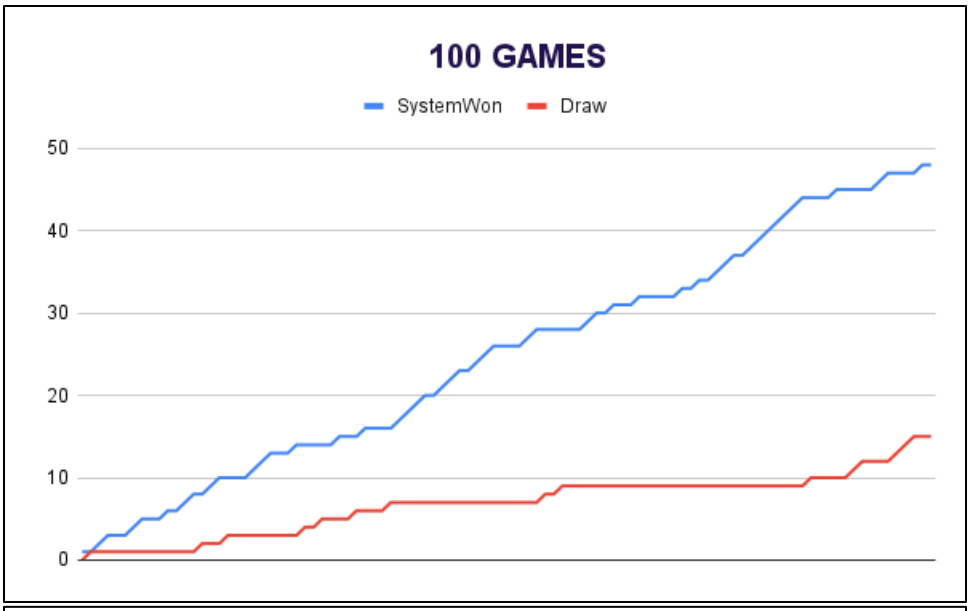


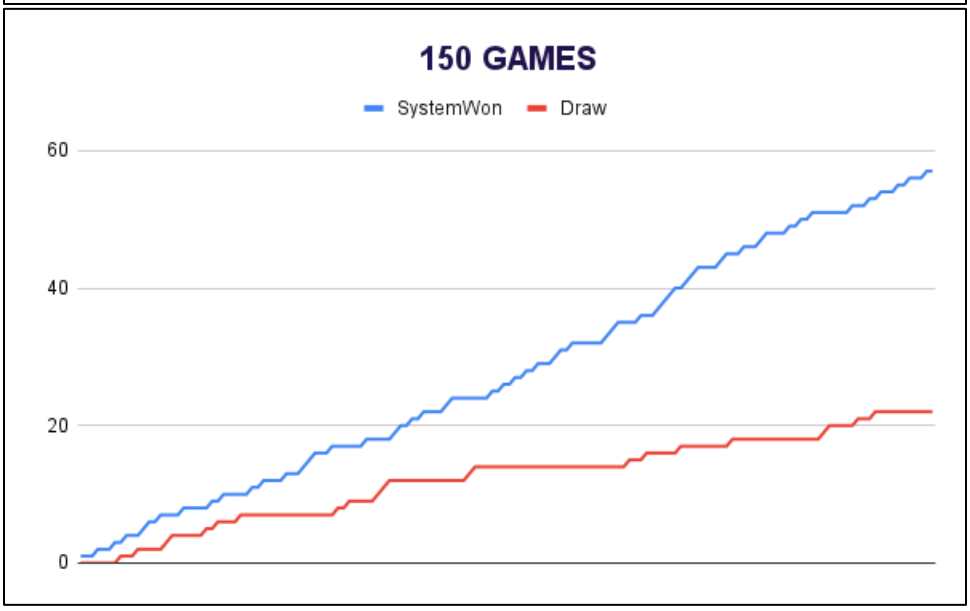




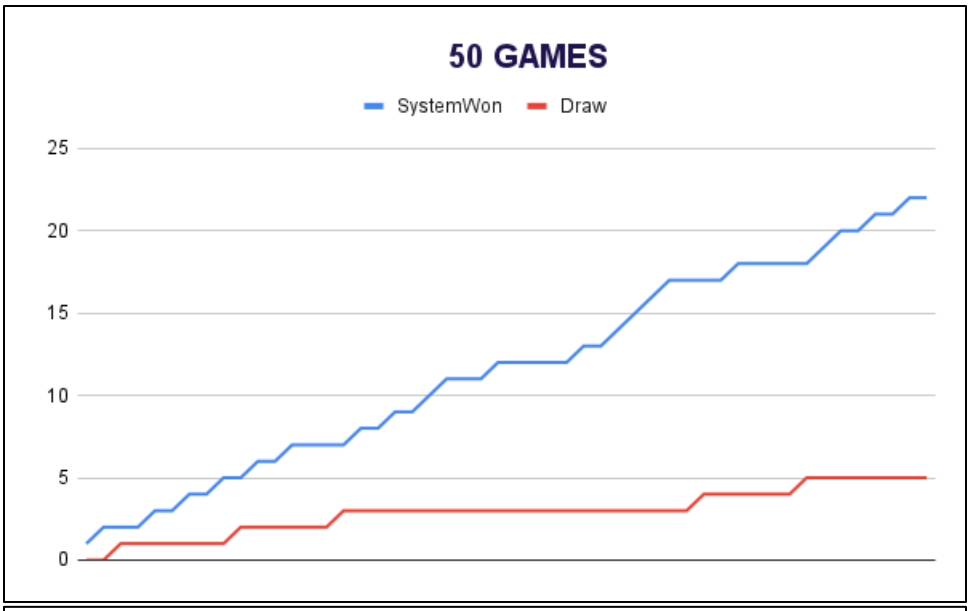
ALPHA: 10 BETA:1 GAMMA:2 DELTA:0

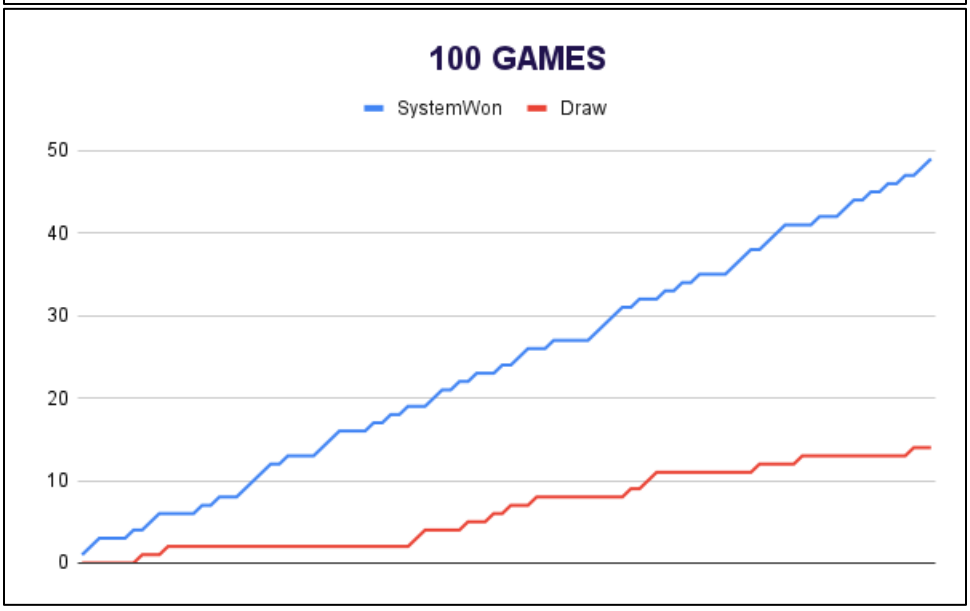

50 GAMES
SystemWon
Draw
30
20
10
0

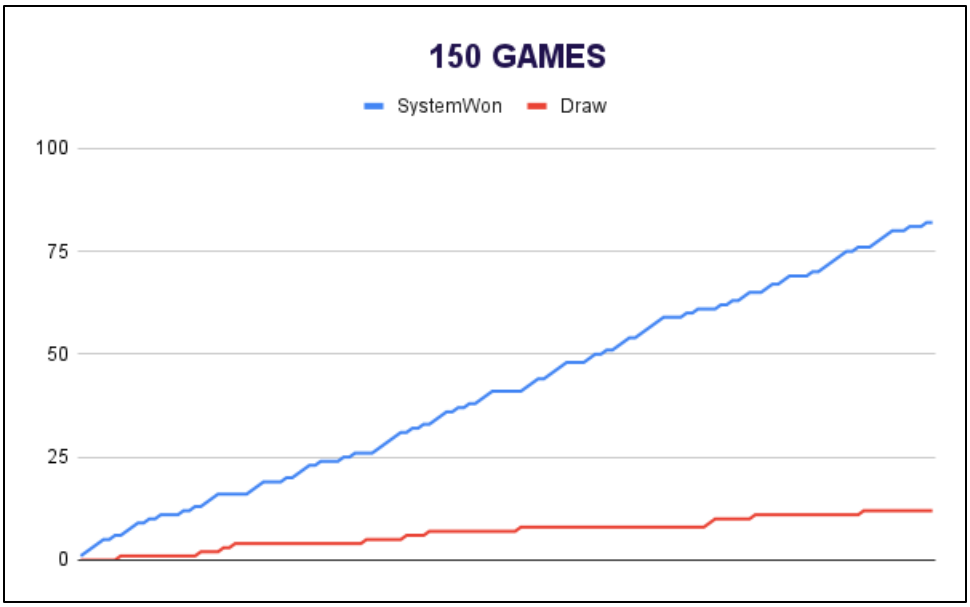




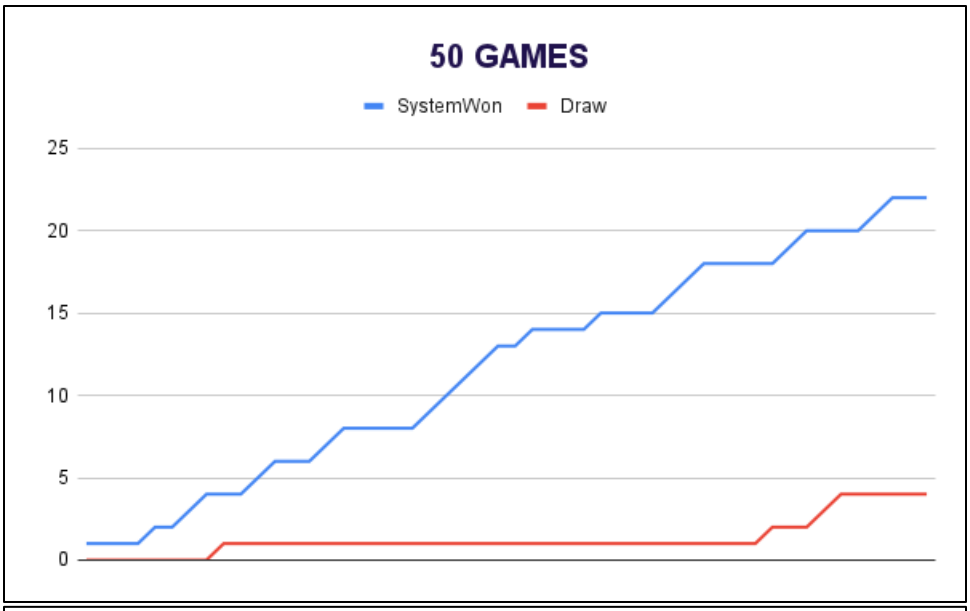
ALPHA: 10 BETA:2 GAMMA:1 DELTA:0

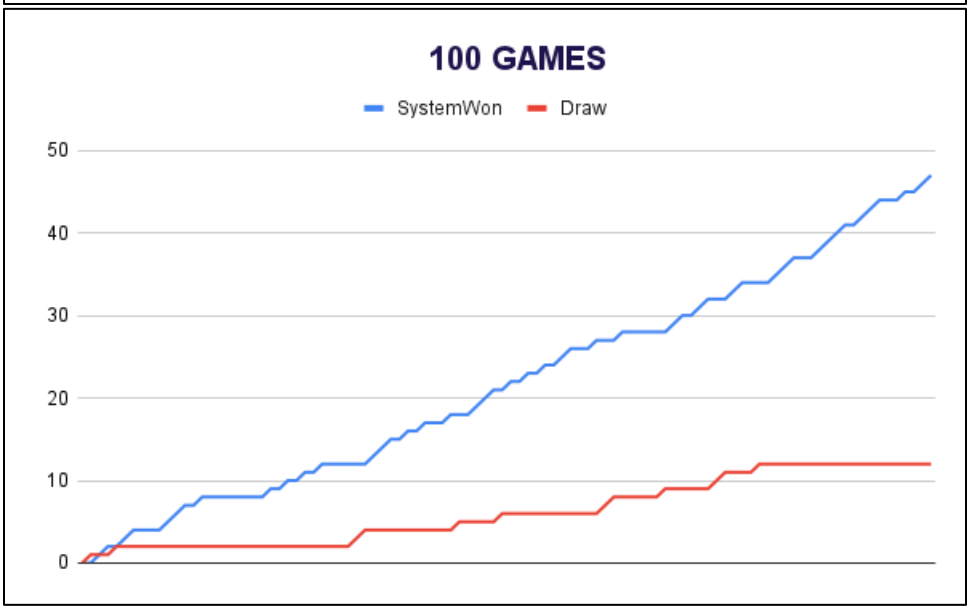

50 GAMES
SystemWon
Draw
25
20
15
10
5
0

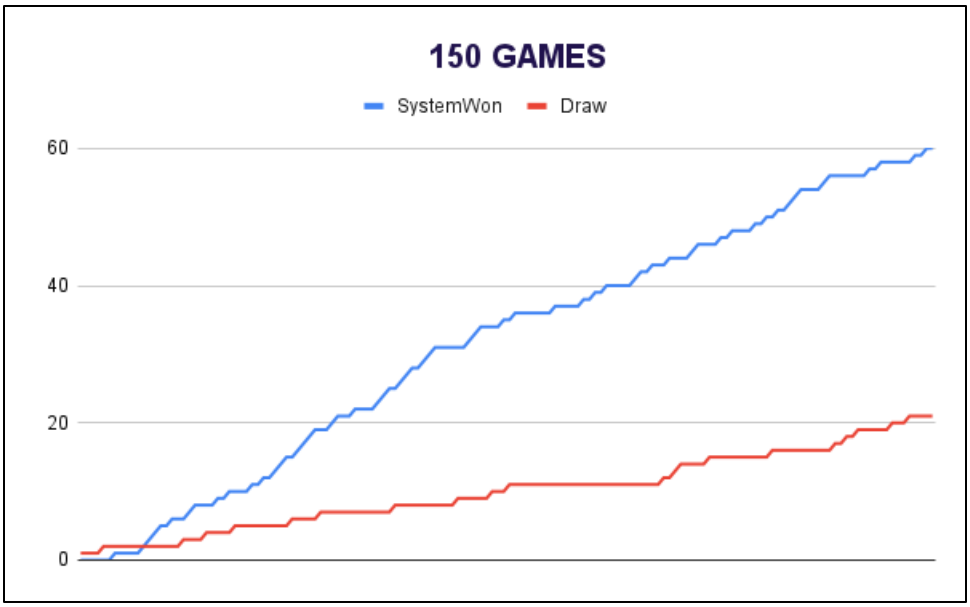

100 GAMES
SystemWon
Draw
50
40
30
20
10
0



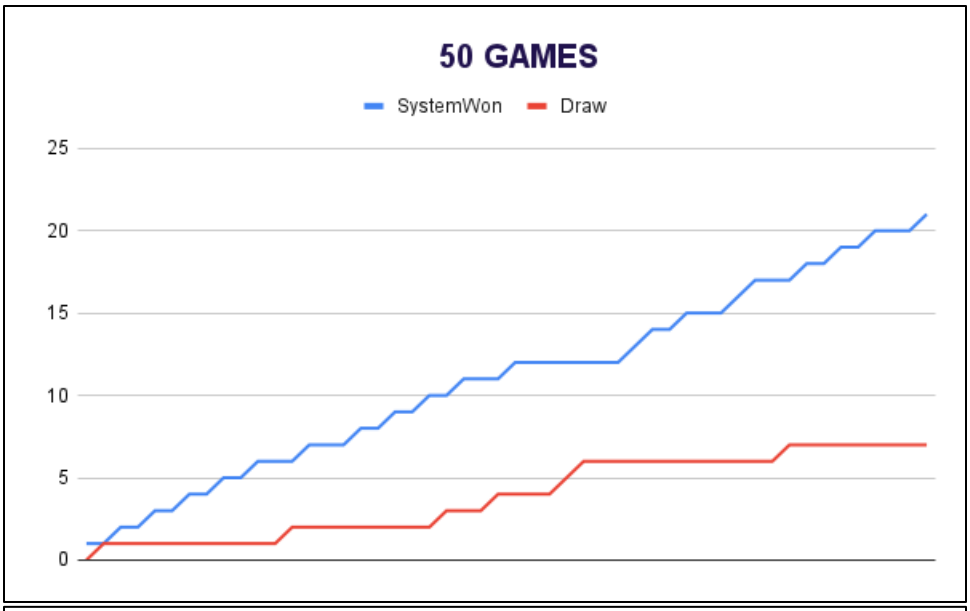
ALPHA: 30 BETA:1 GAMMA:1 DELTA:0

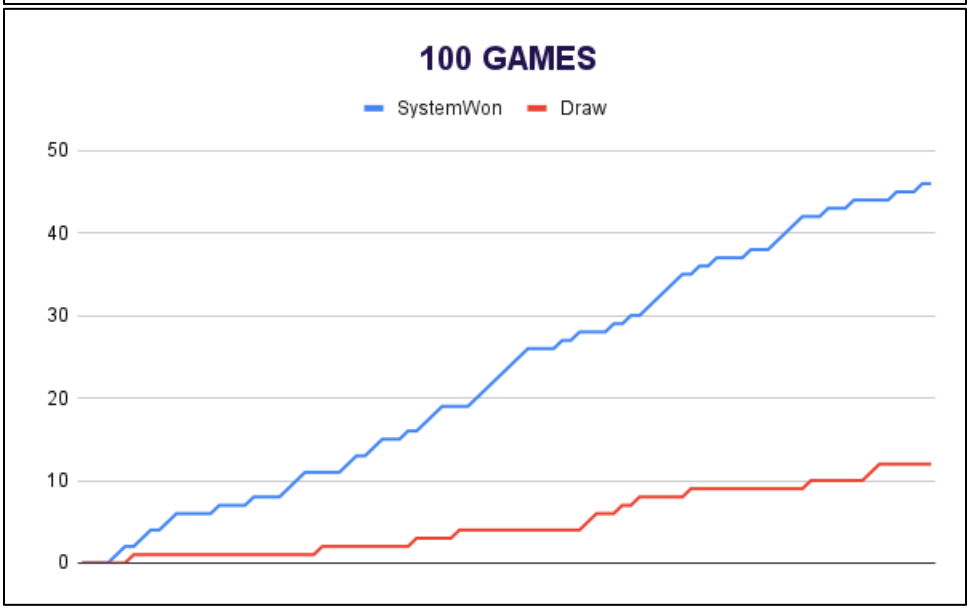

50 GAMES
SystemWon
Draw
25
20
15
10
5
0

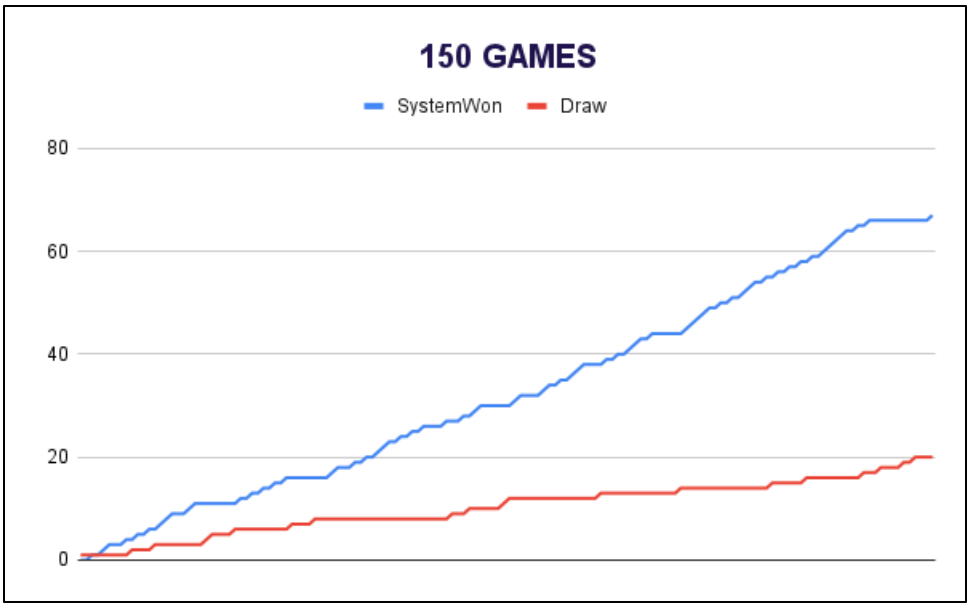

100 GAMES
SystemWon
Draw
50
40
30
20
10
0



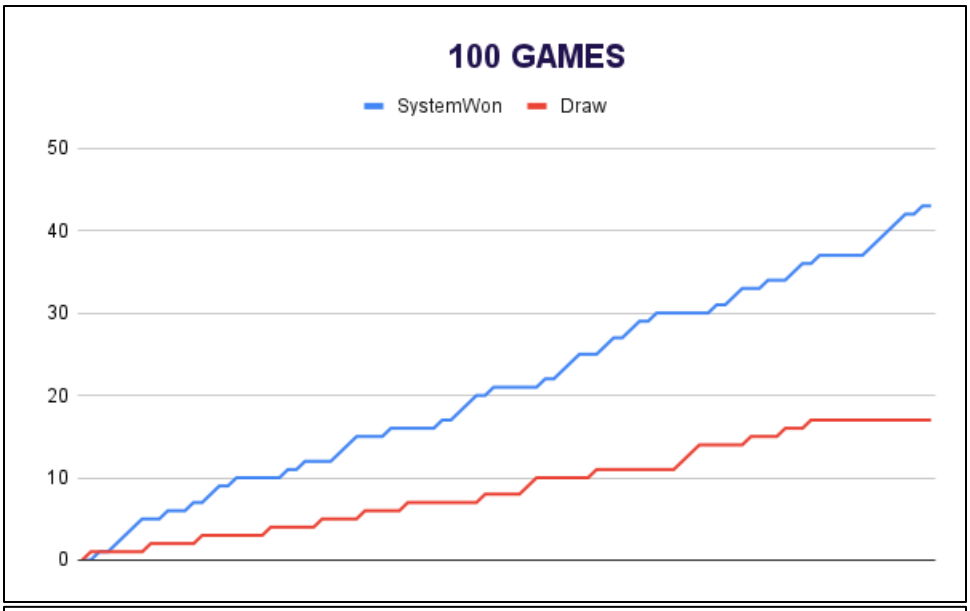
ALPHA: 30 BETA:1 GAMMA:2 DELTA:0

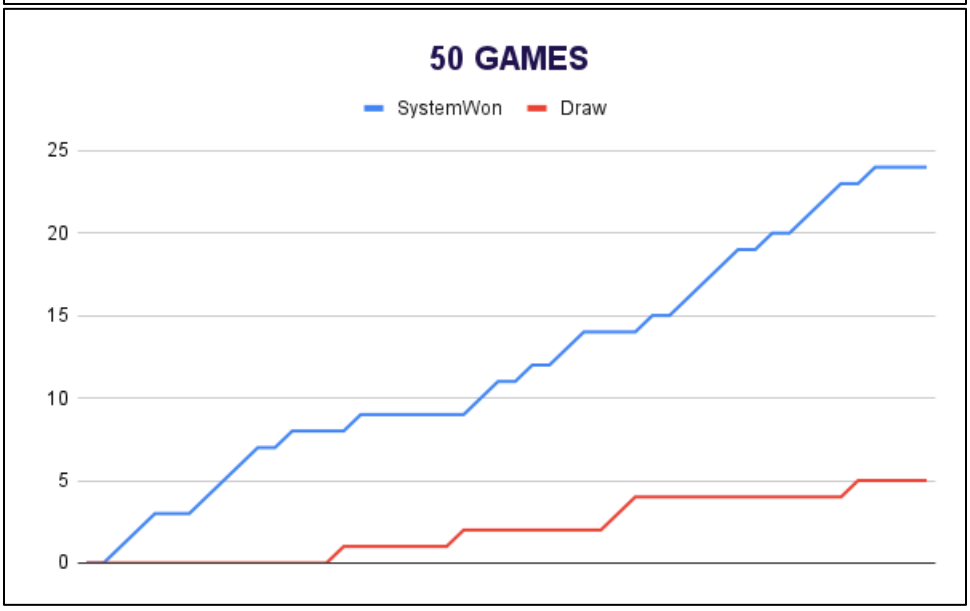

50 GAMES
SystemWon
Draw
25
20
15
10
5
0

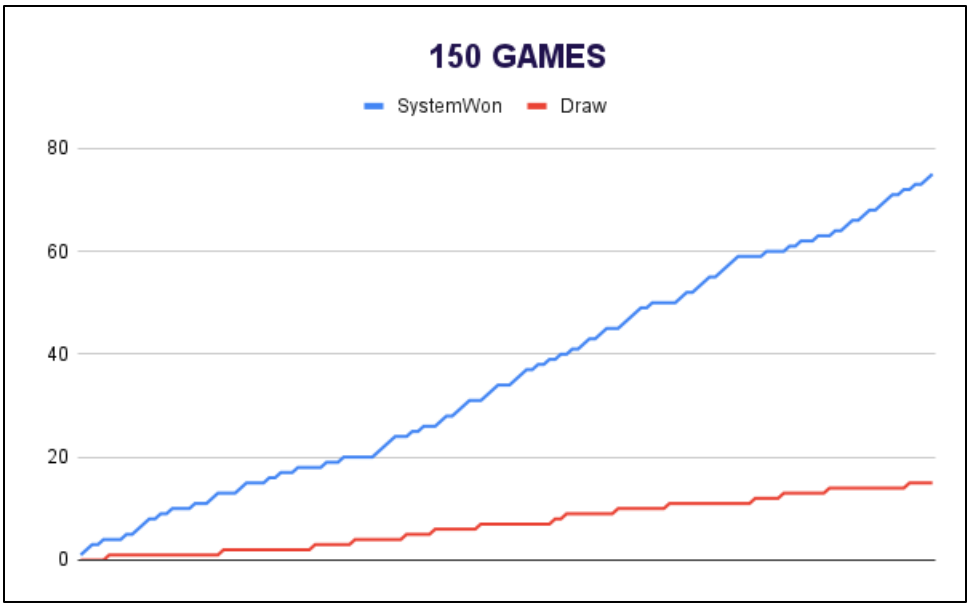

100 GAMES
SystemWon
Draw
50
40
30
20
10
0



ALPHA: 30 BETA:2 GAMMA:1 DELTA:1


100 GAMES
SystemWon
Draw
50
40
30
20
10
0


50 GAMES
SystemWon
Draw
25
20
15
10
5
0



It could be observed from the graphical representation that the number of games won by system and number of draws increases with the number of games played.

Results And Mathematical Analysis

GamesPlayed	SystemWon	Draw	RandomWon
500	251	58	191
500	261	64	175
500	290	55	155
500	297	53	150
500	331	38	131
500	319	42	139
500	312	48	140
500	347	26	127

As we see in the above images, we can clearly see that Menace is trying to learn against all the possible states and moves it can make and win the games with the more no of Games.

To explain it mathematically, as we play more games the probability (p) for the move is increased if the menace has won a game and its more likely to pick it again in future as we increment the beads count for that position in the Matchbox.

Alpha-10, Beta-1, Gamma-1, Delta-0

Intermediate state in the Move of a winning game of a menace

State - 0,1,2,1,0,0,1,2,2

Beads state -- 10,0,0,0,11,9,0,0,0 (Probabilities --- 0.33,0,0,0,0.36,0.3,0,0,0)

After adding Beads for winning move at position 5

Beads state -- 10,0,0,0,11,10,0,0,0 (Probabilities --- 0.32,0,0,0,0.35,0.32,0,0,0)

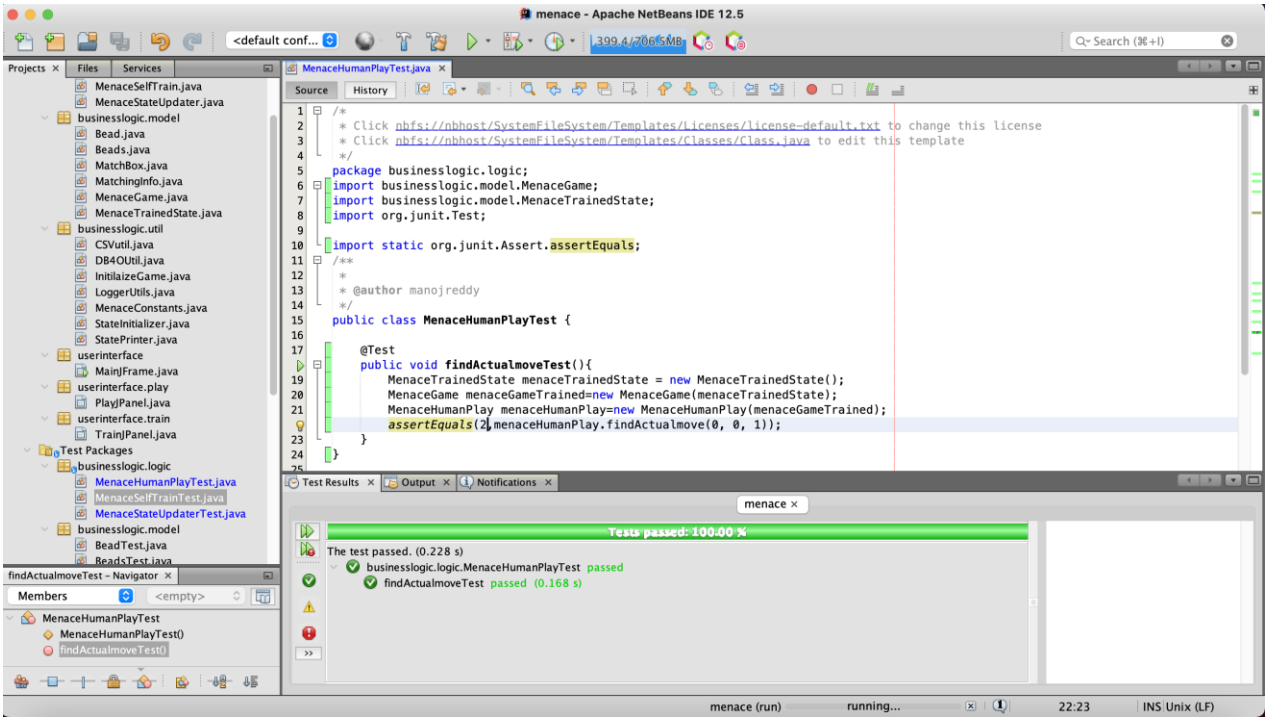
If the game is Lost by Menace for move at position 5

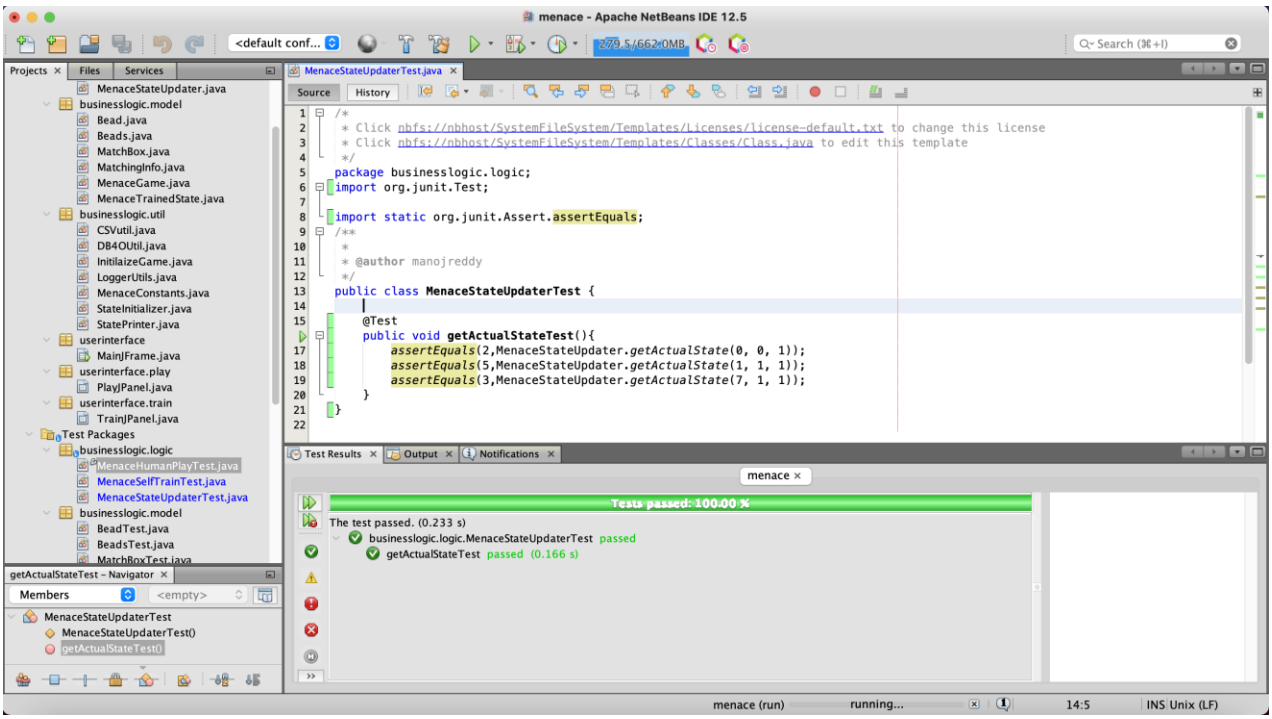
Beads state -- 10,0,0,0,11,8,0,0,0 (Probabilities --- 0.34,0,0,0,0.37,0.27,0,0,0)

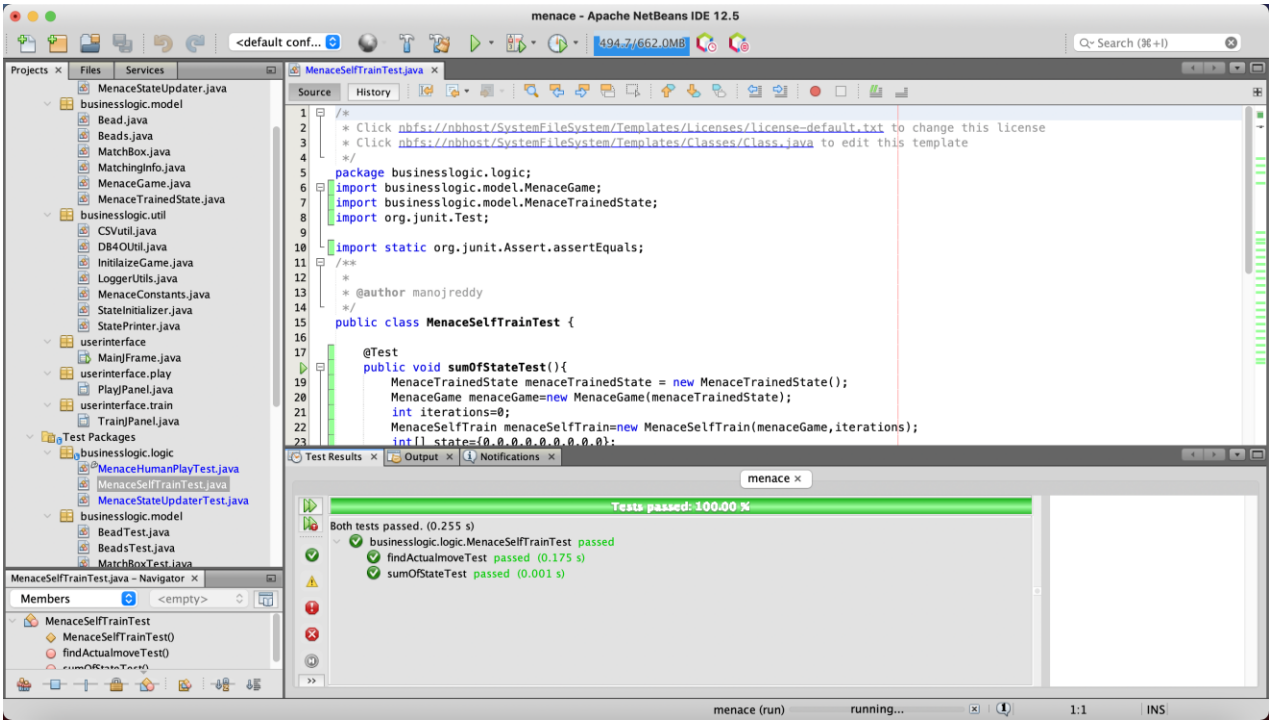
As we can see above we are increasing the probabilities for the menace to make a good move and learn from each viable move. So, increasing the games actually has a major impact on the probability distributions on these moves which helps in making a More viable move in long run.

Test Cases

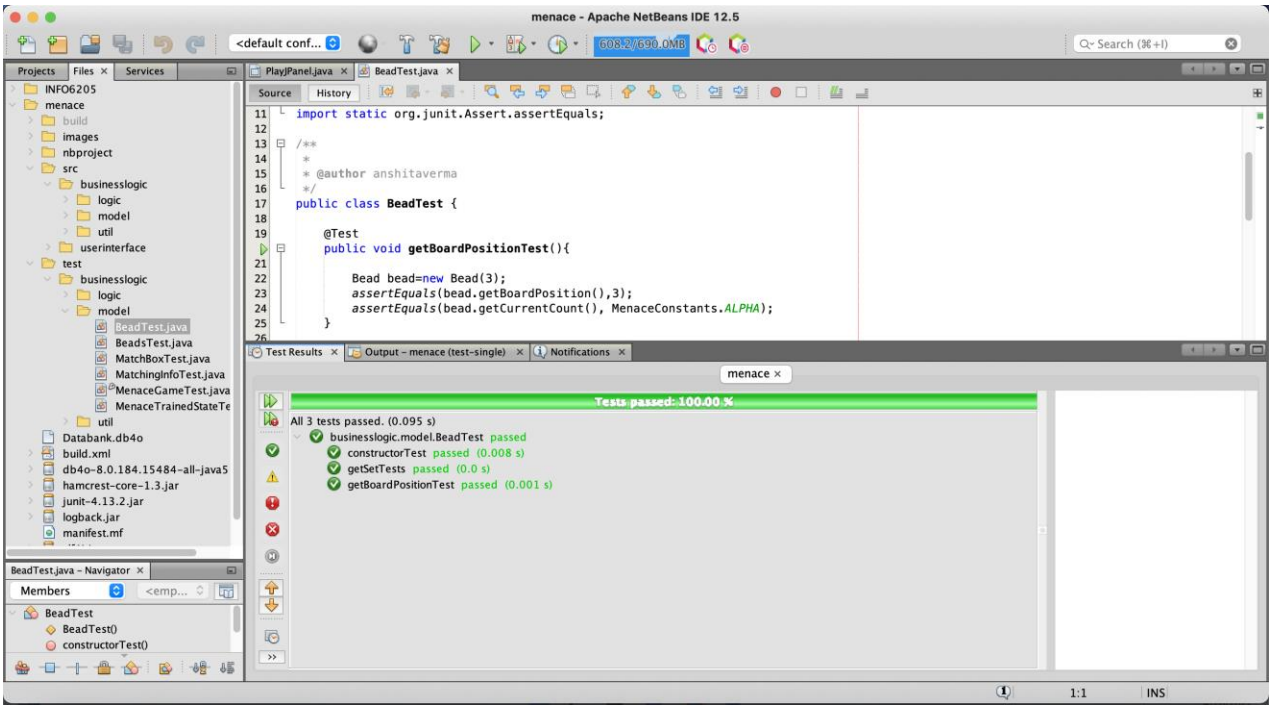
Test cases for logic classes:

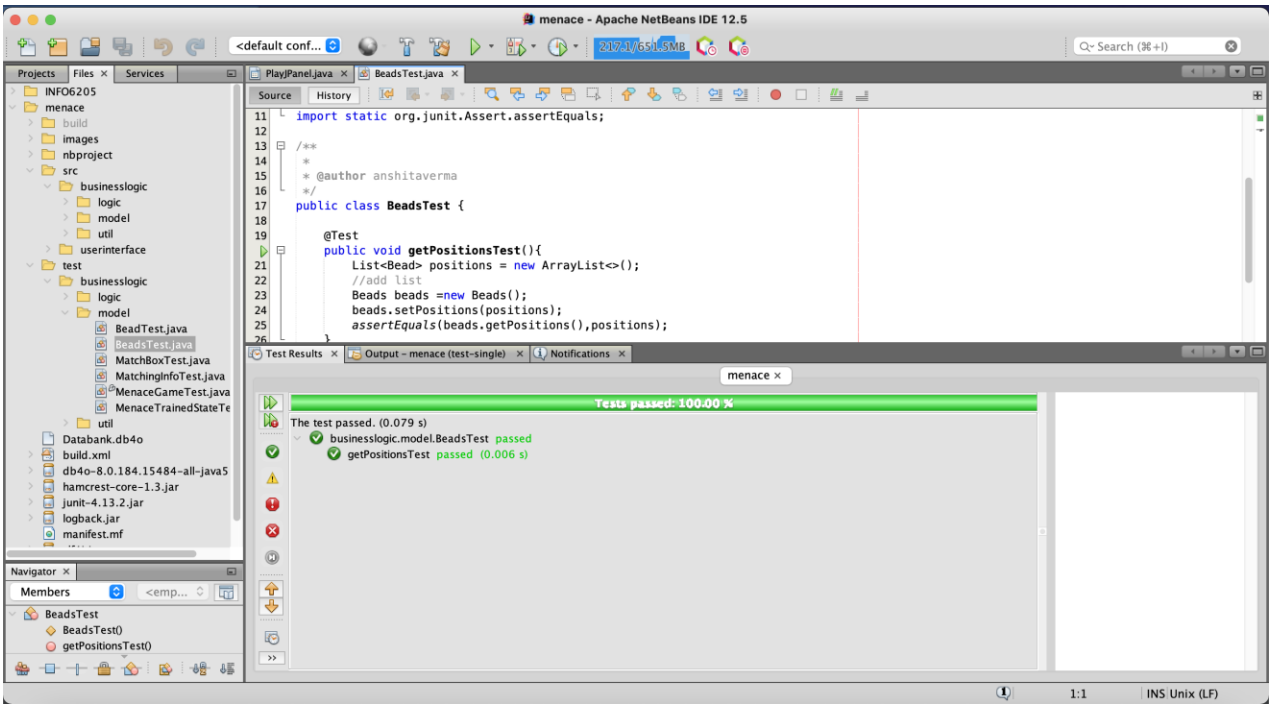

menace - Apache NetBeans IDE 12.5
MenaceHumanPlayTest.java
Source
History
/*
* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
*/
package businesslogic.logic;
import businesslogic.model.MenaceGame;
import businesslogic.model.MenaceTrainedState;
import org.junit.Test;
import static org.junit.Assert.assertEquals;
/**
*
* @author manojreddy
*/
public class MenaceHumanPlayTest {
@Test
public void findActualmoveTest(){
MenaceTrainedState menaceTrainedState = new MenaceTrainedState();
MenaceGame menaceGameTrained=new MenaceGame(menaceTrainedState);
MenaceHumanPlay menaceHumanPlay=new MenaceHumanPlay(menaceGameTrained);
assertEquals(2,menaceHumanPlay.findActualmove(0, 0, 1));
}
}
Test Results
Output
Notifications
menace
Tests passed: 100.00 %
The test passed. (0.228 s)
businesslogic.logic.MenaceHumanPlayTest passed
findActualmoveTest passed (0.168 s)
menace (run) running...
22:23
INS Unix (LF)

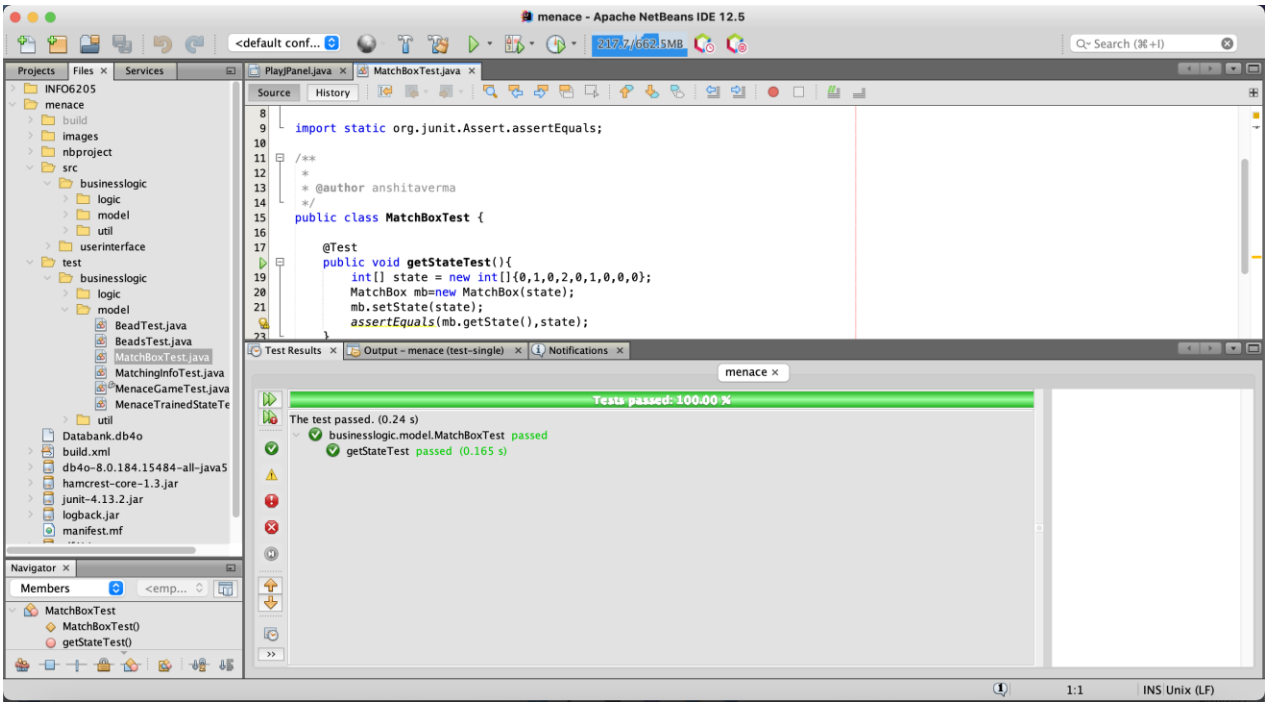

menace - Apache NetBeans IDE 12.5
MenaceStateUpdaterTest.java
Source
History
/*
* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
*/
package businesslogic.logic;
import org.junit.Test;
import static org.junit.Assert.assertEquals;
/**
*
* @author manojreddy
*/
public class MenaceStateUpdaterTest {
@Test
public void getActualStateTest(){
assertEquals(2,MenaceStateUpdater.getActualState(0, 0, 1));
assertEquals(5,MenaceStateUpdater.getActualState(1, 1, 1));
assertEquals(3,MenaceStateUpdater.getActualState(7, 1, 1));
}
}
Test Results
Output
Notifications
menace
Tests passed: 100.00 %
The test passed. (0.233 s)
businesslogic.logic.MenaceStateUpdaterTest passed
getActualStateTest passed (0.166 s)
menace (run) running...
14:5
INS Unix (LF)

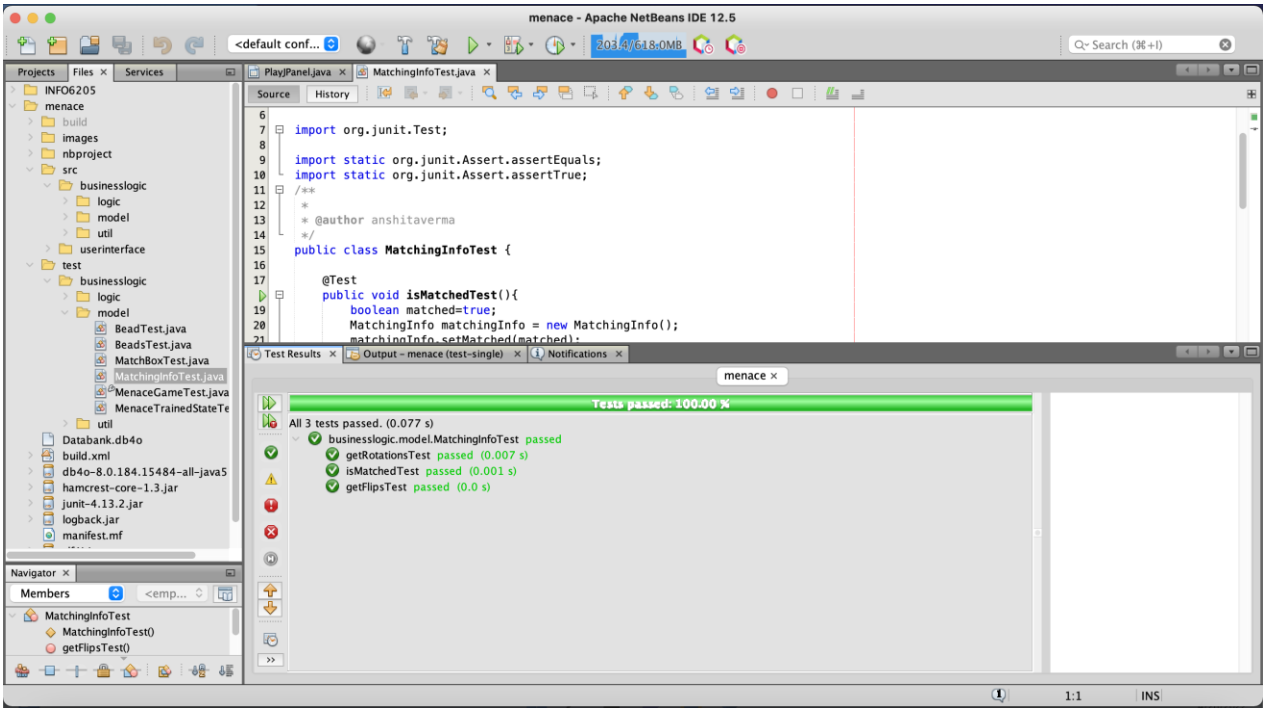


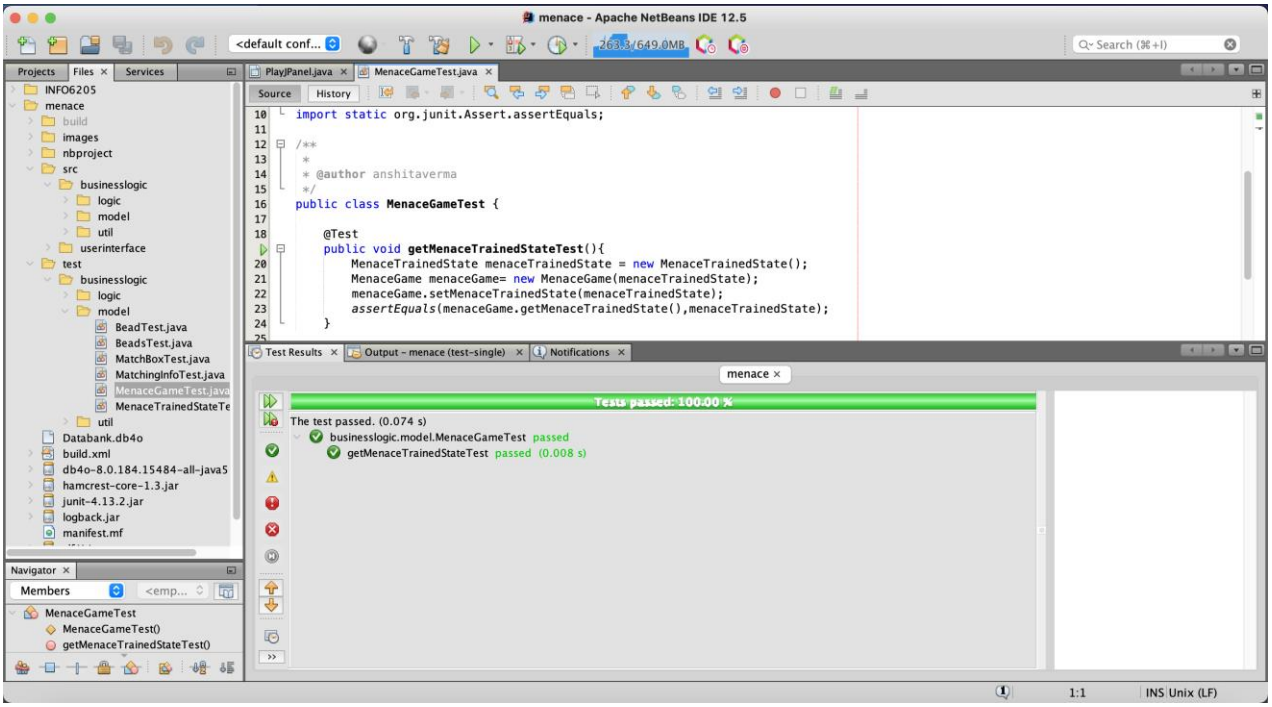
Test cases for model classes:

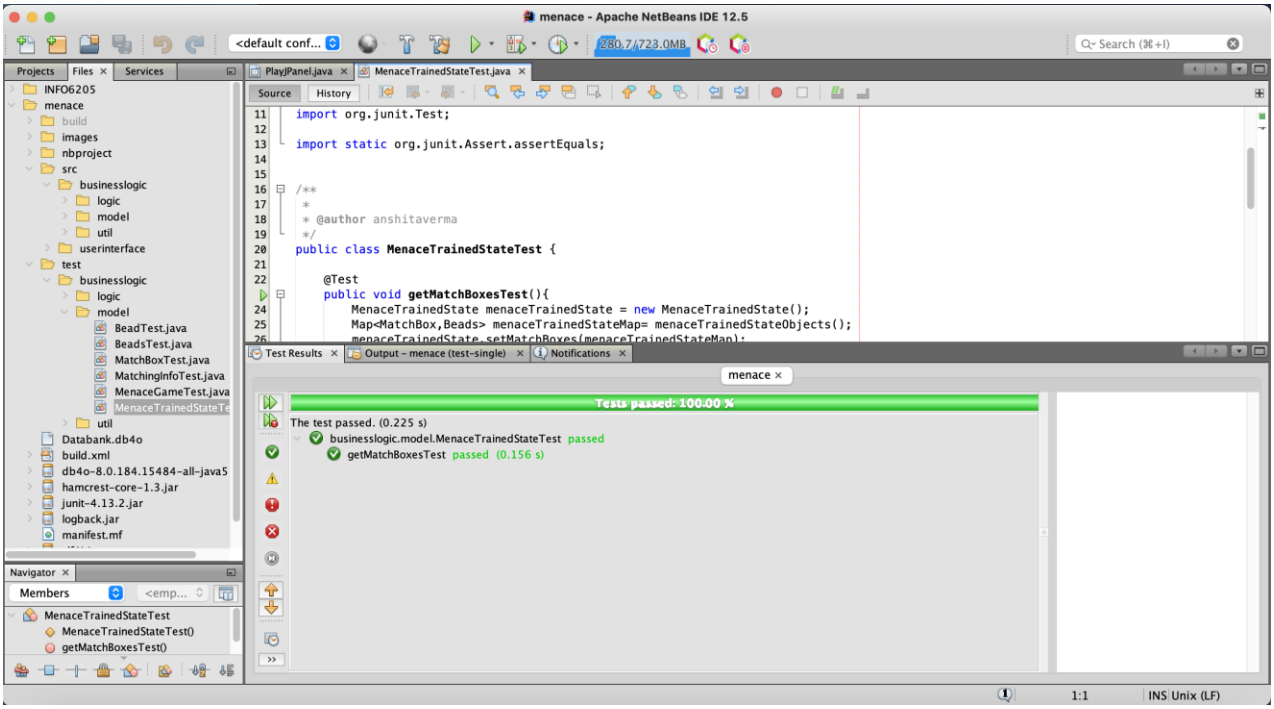



menace - Apache NetBeans IDE 12.5
<default conf...
Search (⌘+I)
Projects
Files
Services
INFO6205
menace
build
images
nbproject
src
businesslogic
logic
model
util
userinterface
test
businesslogic
logic
model
BeadTest.java
BeadsTest.java
MatchBoxTest.java
MatchingInfoTest.java
MenaceGameTest.java
MenaceTrainedStateTe
util
Databank.db4o
build.xml
db4o-8.0.184.15484-all-java5
hamcrest-core-1.3.jar
junit-4.13.2.jar
logback.jar
manifest.mf
Navigator
Members
BeadsTest
BeadsTest()
getPositionsTest()
PlayJPanel.java
BeadsTest.java
Source
History
import static org.junit.Assert.assertEquals;
/**
* @author anshitaverma
*/
public class BeadsTest {
@Test
public void getPositionsTest(){
List<Bead> positions = new ArrayList<>();
//add list
Beads beads =new Beads();
beads.setPositions(positions);
assertEquals(beads.getPositions(),positions);
}
Test Results
Output - menace (test-single)
Notifications
menace
Tests passed: 100.00 %
The test passed. (0.079 s)
businesslogic.model.BeadsTest passed
getPositionsTest passed (0.006 s)
1:1
INS Unix (LF)

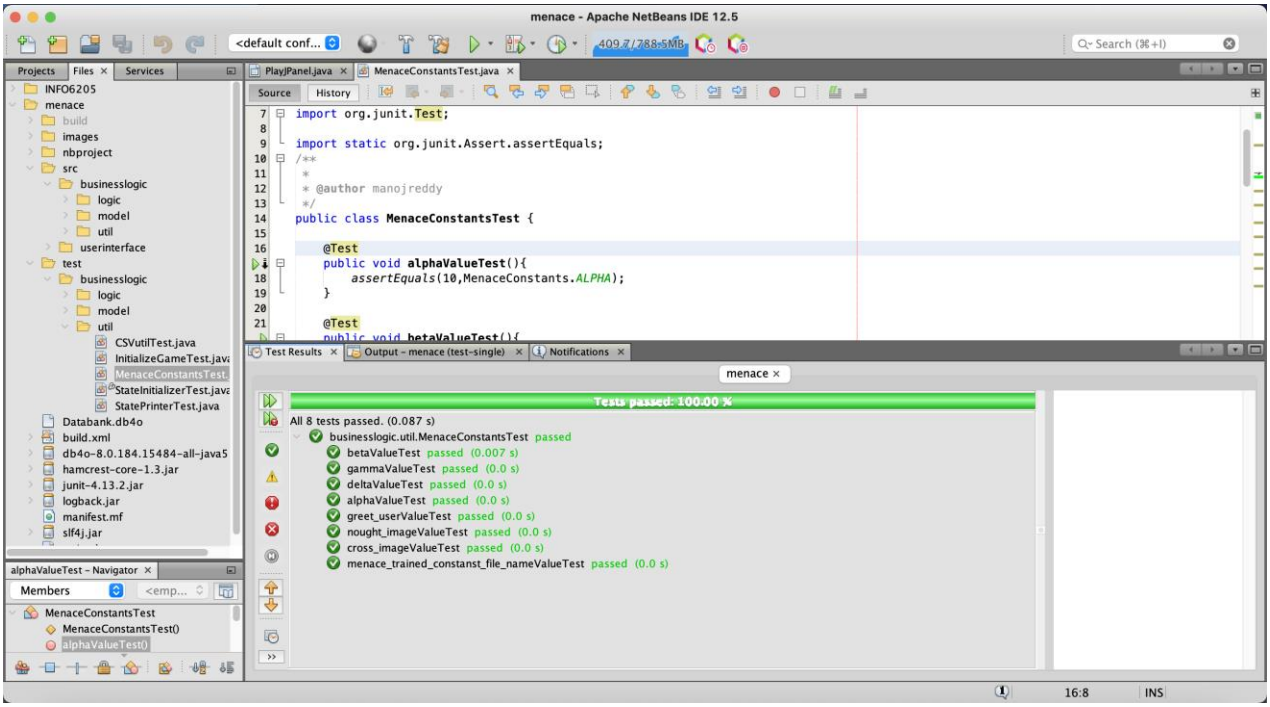

menace - Apache NetBeans IDE 12.5
<default conf...
Search (⌘+I)
Projects
Files
Services
INFO6205
menace
build
images
nbproject
src
businesslogic
logic
model
util
userinterface
test
businesslogic
logic
model
BeadTest.java
BeadsTest.java
MatchBoxTest.java
MatchingInfoTest.java
MenaceGameTest.java
MenaceTrainedStateTe
util
Databank.db4o
build.xml
db4o-8.0.184.15484-all-java5
hamcrest-core-1.3.jar
junit-4.13.2.jar
logback.jar
manifest.mf
Navigator
Members
MatchBoxTest
MatchBoxTest()
getStateTest()
PlayJPanel.java
MatchBoxTest.java
Source
History
import static org.junit.Assert.assertEquals;
/**
* @author anshitaverma
*/
public class MatchBoxTest {
@Test
public void getStateTest(){
int[] state = new int[]{0,1,0,2,0,1,0,0,0};
MatchBox mb=new MatchBox(state);
mb.setState(state);
assertEquals(mb.getState(),state);
}
Test Results
Output - menace (test-single)
Notifications
menace
Tests passed: 100.00 %
The test passed. (0.24 s)
businesslogic.model.MatchBoxTest passed
getStateTest passed (0.165 s)
1:1
INS Unix (LF)

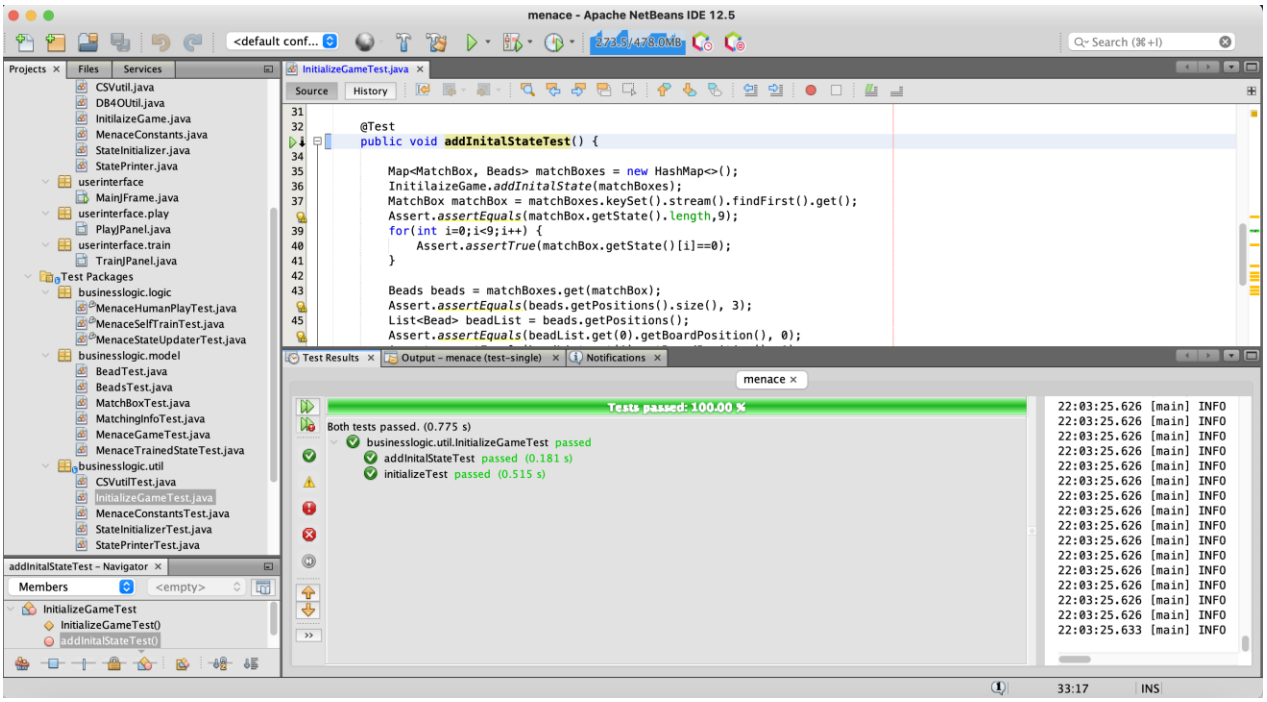

menace - Apache NetBeans IDE 12.5
<default conf...
Search (⌘+I)
Projects
Files
Services
INFO6205
menace
build
images
nbproject
src
businesslogic
logic
model
util
userinterface
test
businesslogic
logic
model
BeadTest.java
BeadsTest.java
MatchBoxTest.java
MatchingInfoTest.java
MenaceGameTest.java
MenaceTrainedStateTe
util
Databank.db4o
build.xml
db4o-8.0.184.15484-all-java5
hamcrest-core-1.3.jar
junit-4.13.2.jar
logback.jar
manifest.mf
Navigator
Members
MatchingInfoTest
MatchingInfoTest()
getFlipsTest()
PlayJPanel.java
MatchingInfoTest.java
Source
History
6 import org.junit.Test;
7
8 import static org.junit.Assert.assertEquals;
9 import static org.junit.Assert.assertTrue;
10 /**
11 *
12 * @author anshitaverma
13 */
14 public class MatchingInfoTest {
15
16 @Test
17 public void isMatchedTest(){
18 boolean matched=true;
19 MatchingInfo matchingInfo = new MatchingInfo();
20 matchingInfo.setMatched(matched);
Test Results
Output - menace (test-single)
Notifications
menace
Tests passed: 100.00 %
All 3 tests passed. (0.077 s)
businesslogic.model.MatchingInfoTest passed
getRotationsTest passed (0.007 s)
isMatchedTest passed (0.001 s)
getFlipsTest passed (0.0 s)
1:1
INS

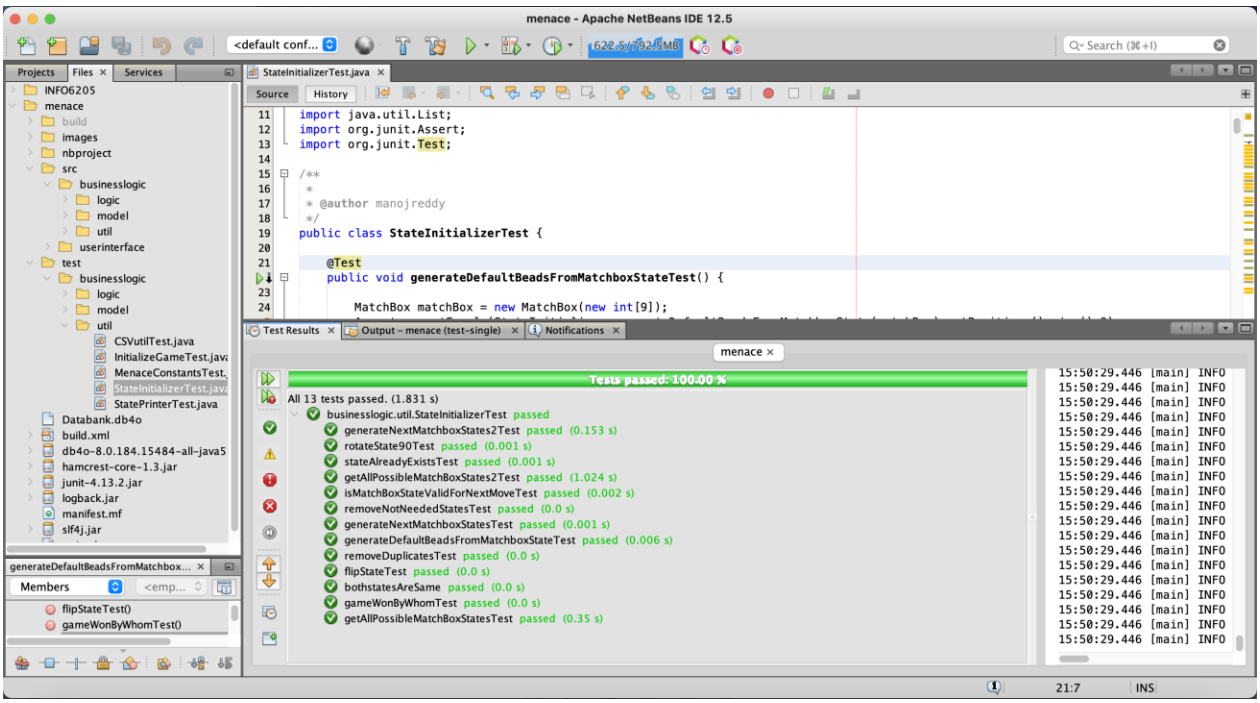

menace - Apache NetBeans IDE 12.5
<default conf...
Search (⌘+I)
Projects
Files
Services
INFO6205
menace
build
images
nbproject
src
businesslogic
logic
model
util
userinterface
test
businesslogic
logic
model
BeadTest.java
BeadsTest.java
MatchBoxTest.java
MatchingInfoTest.java
MenaceGameTest.java
MenaceTrainedStateTe
util
Databank.db4o
build.xml
db4o-8.0.184.15484-all-java5
hamcrest-core-1.3.jar
junit-4.13.2.jar
logback.jar
manifest.mf
Navigator
Members
MenaceGameTest
MenaceGameTest()
getMenaceTrainedStateTest()
PlayJPanel.java
MenaceGameTest.java
Source
History
10 import static org.junit.Assert.assertEquals;
11
12 /**
13 *
14 * @author anshitaverma
15 */
16 public class MenaceGameTest {
17
18 @Test
19 public void getMenaceTrainedStateTest(){
20 MenaceTrainedState menaceTrainedState = new MenaceTrainedState();
21 MenaceGame menaceGame= new MenaceGame(menaceTrainedState);
22 menaceGame.setMenaceTrainedState(menaceTrainedState);
23 assertEquals(menaceGame.getMenaceTrainedState(),menaceTrainedState);
24 }
Test Results
Output - menace (test-single)
Notifications
menace
Tests passed: 100.00 %
The test passed. (0.074 s)
businesslogic.model.MenaceGameTest passed
getMenaceTrainedStateTest passed (0.008 s)
1:1
INS Unix (LF)

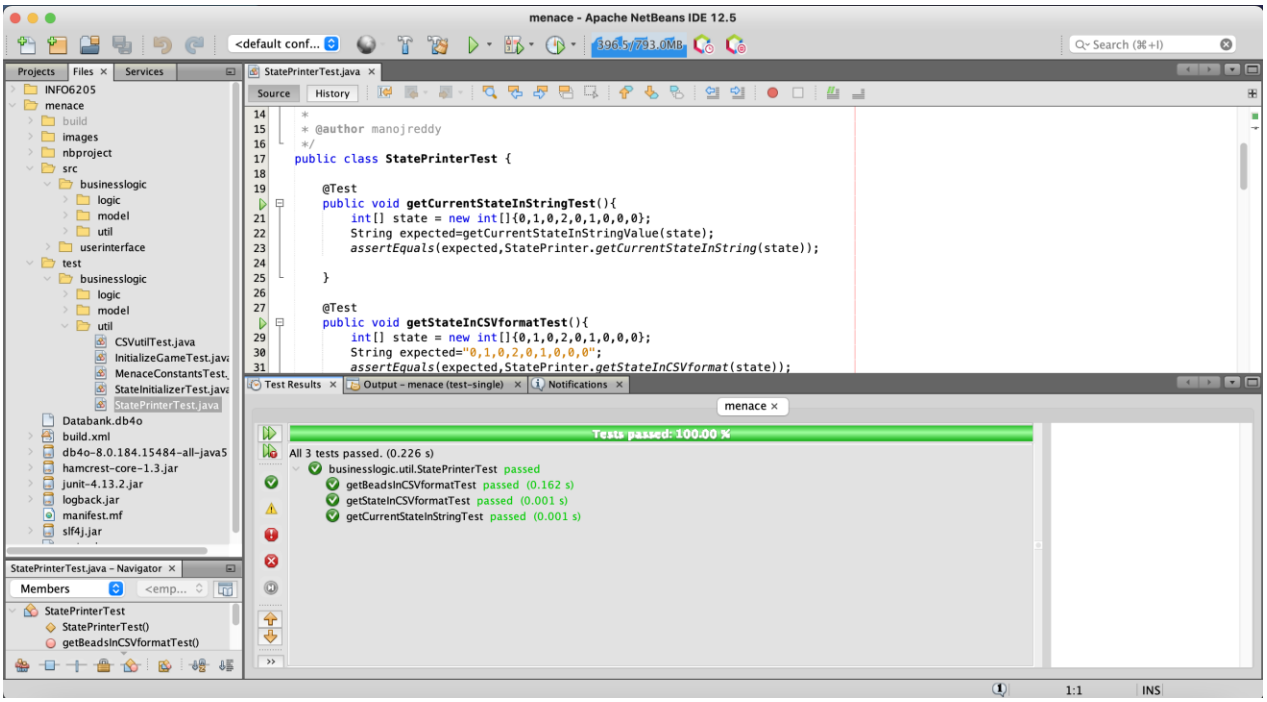


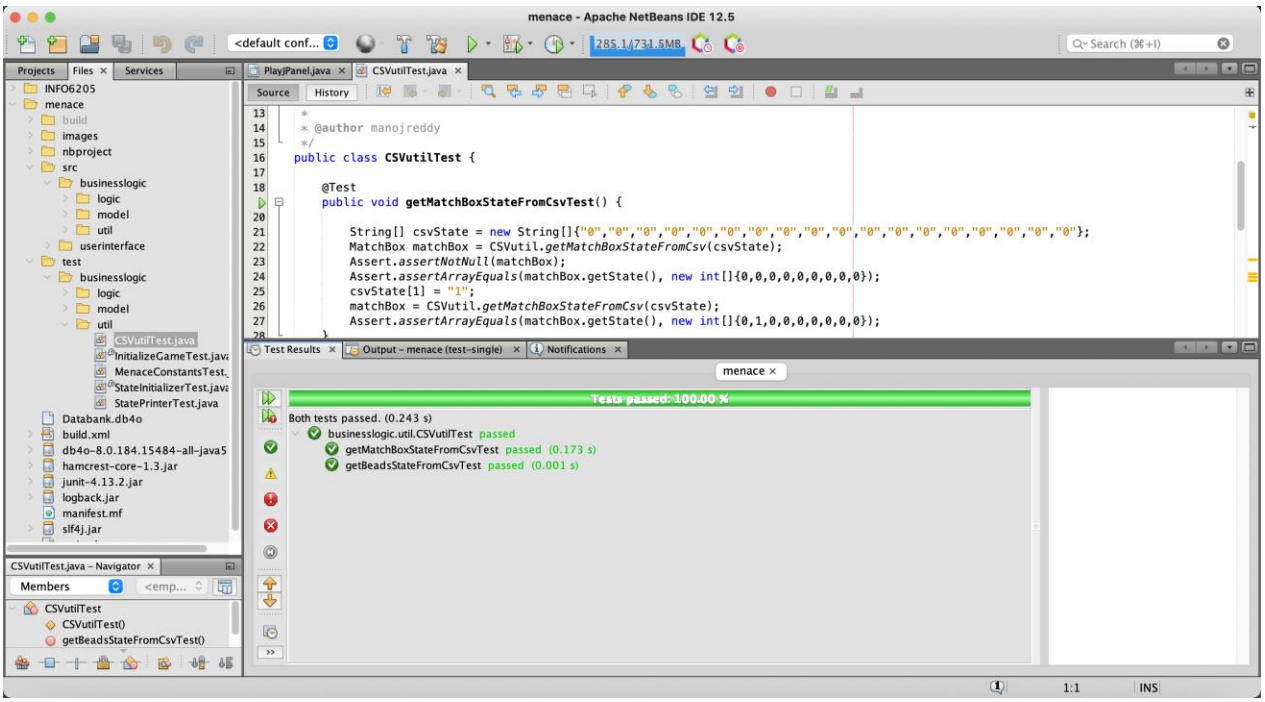
Test cases for util classes:




menace - Apache NetBeans IDE 12.5
InitializeGameTest.java
@Test
public void addInitalStateTest() {
Map<MatchBox, Beads> matchBoxes = new HashMap<>();
InitilaizeGame.addInitalState(matchBoxes);
MatchBox matchBox = matchBoxes.keySet().stream().findFirst().get();
Assert.assertEquals(matchBox.getState().length,9);
for(int i=0;i<9;i++) {
Assert.assertTrue(matchBox.getState()[i]==0);
}
Beads beads = matchBoxes.get(matchBox);
Assert.assertEquals(beads.getPositions().size(), 3);
List<Bead> beadList = beads.getPositions();
Assert.assertEquals(beadList.get(0).getBoardPosition(), 0);
Test Results
Output - menace (test-single)
Notifications
Tests passed: 100.00 %
Both tests passed. (0.775 s)
businesslogic.util.InitializeGameTest passed
addInitalStateTest passed (0.181 s)
initializeTest passed (0.515 s)


menace - Apache NetBeans IDE 12.5
StateInitializerTest.java
import java.util.List;
import org.junit.Assert;
import org.junit.Test;
/**
*
* @author manojreddy
*/
public class StateInitializerTest {
@Test
public void generateDefaultBeadsFromMatchboxStateTest() {
MatchBox matchBox = new MatchBox(new int[9]);
Test Results
Output - menace (test-single)
Notifications
Tests passed: 100.00 %
All 13 tests passed. (1.831 s)
businesslogic.util.StateInitializerTest passed
generateNextMatchboxStates2Test passed (0.153 s)
rotateState90Test passed (0.001 s)
stateAlreadyExistsTest passed (0.001 s)
getAllPossibleMatchBoxStates2Test passed (1.024 s)
isMatchBoxStateValidForNextMoveTest passed (0.002 s)
removeNotNeededStatesTest passed (0.0 s)
generateNextMatchboxStatesTest passed (0.001 s)
generateDefaultBeadsFromMatchboxStateTest passed (0.006 s)
removeDuplicatesTest passed (0.0 s)
flipStateTest passed (0.0 s)
bothstatesAreSame passed (0.0 s)
gameWonByWhomTest passed (0.0 s)
getAllPossibleMatchBoxStatesTest passed (0.35 s)


menace - Apache NetBeans IDE 12.5
StatePrinterTest.java
@author manojreddy
public class StatePrinterTest {
@Test
public void getCurrentStateInStringTest(){
int[] state = new int[]{0,1,0,2,0,1,0,0,0};
String expected=getCurrentStateInStringValue(state);
assertEquals(expected,StatePrinter.getCurrentStateInString(state));
@Test
public void getStateInCSVformatTest(){
int[] state = new int[]{0,1,0,2,0,1,0,0,0};
String expected="0,1,0,2,0,1,0,0,0";
assertEquals(expected,StatePrinter.getStateInCSVformat(state));
Test Results
Output - menace (test-single)
Notifications
Tests passed: 100.00 %
All 3 tests passed. (0.226 s)
businesslogic.util.StatePrinterTest passed
getBeadsInCSVformatTest passed (0.162 s)
getStateInCSVformatTest passed (0.001 s)
getCurrentStateInStringTest passed (0.001 s)


menace - Apache NetBeans IDE 12.5
CSVutilTest.java
@author manojreddy
public class CSVutilTest {
@Test
public void getMatchBoxStateFromCsvTest() {
String[] csvState = new String[]{"0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0"};
MatchBox matchBox = CSVutil.getMatchBoxStateFromCsv(csvState);
Assert.assertNotNull(matchBox);
Assert.assertArrayEquals(matchBox.getState(), new int[]{0,0,0,0,0,0,0,0,0,0});
csvState[1] = "1";
matchBox = CSVutil.getMatchBoxStateFromCsv(csvState);
Assert.assertArrayEquals(matchBox.getState(), new int[]{0,1,0,0,0,0,0,0,0,0});
Test Results
Output - menace (test-single)
Notifications
Tests passed: 100.00 %
Both tests passed. (0.243 s)
businesslogic.util.CSVutilTest passed
getMatchBoxStateFromCsvTest passed (0.173 s)
getBeadsStateFromCsvTest passed (0.001 s)

Conclusion

Since we are training the system with a Random Mover it trains the menace at all possible moves and helps in learning it better for the long term. So, we can expect the Menace to train better as we increase the number of games. But in order to train the system better we can further enhance the algorithm by dynamically changing the Beta, Gamma and Delta values based on a cost function. Also, by making some intelligent moves by human data set.

References

- https://en.wikipedia.org/wiki/Tic-tac-toe
- https://www.youtube.com/watch?v=TtisQ9yZ2zo&t=2210s&ab_channel=TheRoyalInstitution
- https://www.mscroggs.co.uk/blog/19
- https://people.csail.mit.edu/brooks/idocs/matchbox.pdf