INFO6205: Program Structures and Algorithms

Spring 2022 Project
The Menace

Team Members

Name: Jayanth Vakkalagadda
NUID: 002950342

Name: Sai Meghana Surapaneni
NUID: 002929424

1. Introduction

1.1 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.
- You will need to choose values for:

Parameter	Definition
alpha	the number of ”beads” of a colour in each “matchbox” at the start of the game, may be different for each move: first move, second move, etc.
beta	the number of ”beads” to add to the “matchbox” in the event of a win
gamma	the number of ”beads” to take to the “matchbox” in the event of a loss
delta	the number of ”beads” to add to the “matchbox” in the event of a draw

Human strategy

- Chooses optimal strategy with probability p*. In the “zone,” chooses random move.

Implement logging

- Log each training run with date/time, win/loss/draw, and p. If you vary alpha, beta, etc. then record these values also. (A suitable value of “p” might be 0.9)
- For your final match(es), log every move taken by the “Menace” and its opponent.

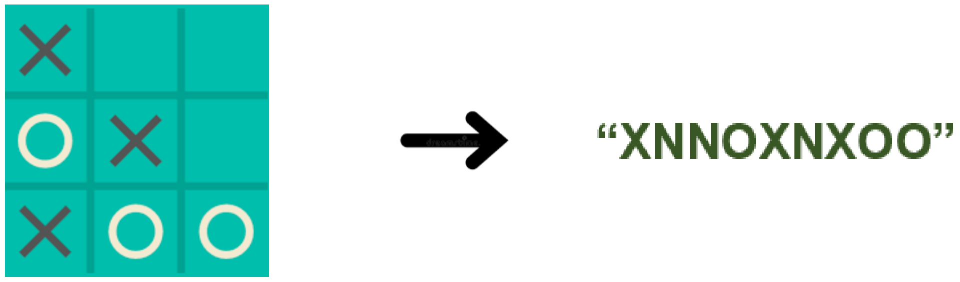
- Choose SLF4j as your logging framework. (if you're used to something else, or not using Java, then choose whatever logging framework you like)

Unit tests

- You must run your unit tests before you start training. Show the date/time of your most recent unit test run.
- Tests must have good coverage. (Each method must be covered)

1.2 Approach & Algorithm

- In this Tic-Tac-Toe game, we have 2 players - Menace & Human.
- In this project scope, Menace always starts the game.
- Menace gets 1st, 3rd, 5th, 7th, 9th turns, while Human gets 2nd, 4th, 6th, 8th turns or even less turns if the game gets over soon.
- In order to play the 9th move, we need not pick any bead as there is only one position that could be played on the board. Hence, its not necessary to store any pattern to support the 9th move of the Menace.
- All the possible patterns in the game are processed using a string format from a 2D matrix format.
- First 3 rows of the matrix are the first 3 characters in the string followed by the other rows.
- Empty boxes are replaced by the letter "N" while "X" & "O" remain the same.



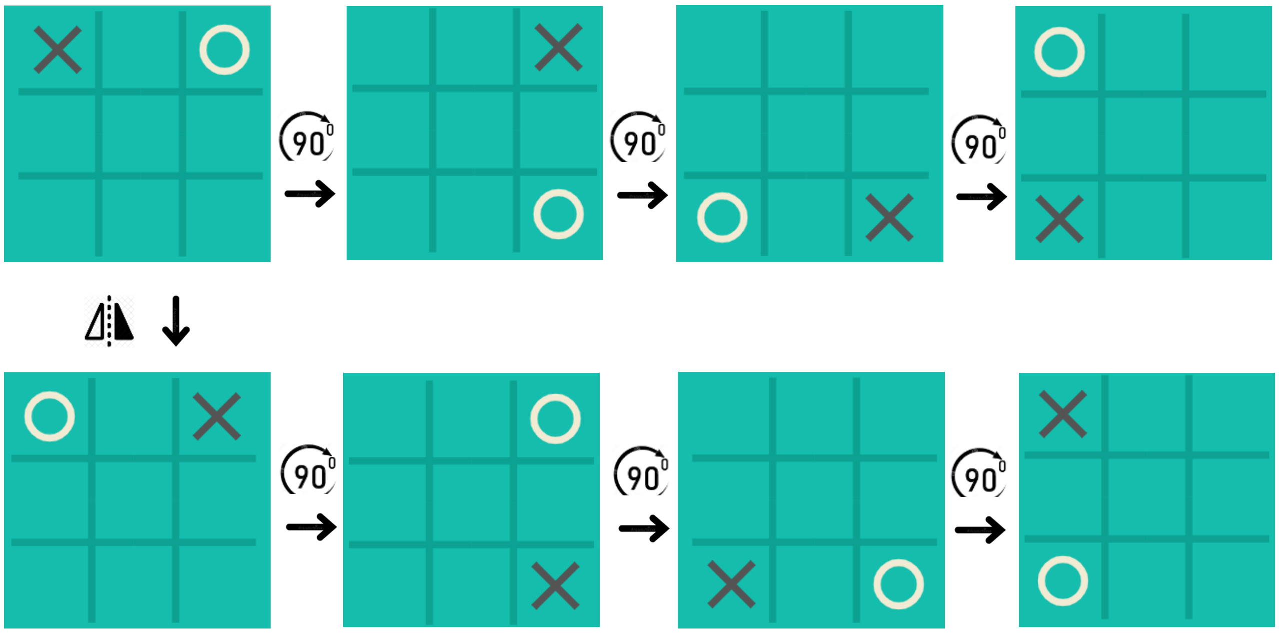
- We get the input parameters like alpha, beta, gamma, delta, and "p" from the user which is handled in the main method of Driver class.
- Based on the alpha (which is a list of 4 numbers), we create a csv file known as beads_file that has all the patterns where each pattern has 9 different colours of beads and their count, and here each colour of bead corresponds to a position.

Pattern	Blue	Red	Yellow	Black	White	Orange	Pink	Green	Brown
NNNNNNNNN	0	0	8	0	8	0	0	8	0

- The above-mentioned pattern denotes the start of the game as there are no markings by any player. As Menace plays first, it picks a bead among – 8 yellow, 8 white and 8 green beads randomly and plays the move.
- Various Moves are as follows

Colour	**Row**	**Column**
Blue	1	1
Red	1	2
Yellow	1	3
Black	2	1
White	2	2
Orange	2	3
Pink	3	1
Green	3	2
Brown	3	3

- Also, we made sure to eliminate all the similar patterns while creating the beads_file to fasten the training process of the Menace.



- Above is an example of the similar states, which are obtained just by 90-degree clockwise rotations and flip operation along vertical axis. So, from the above 8 patterns we store only one pattern in the beads_data file.
- Also, in most cases, the current state of the game will not be available in the beads_file. In those cases, we must try to find similar patterns in the file by rotating and flipping the current state of the game.

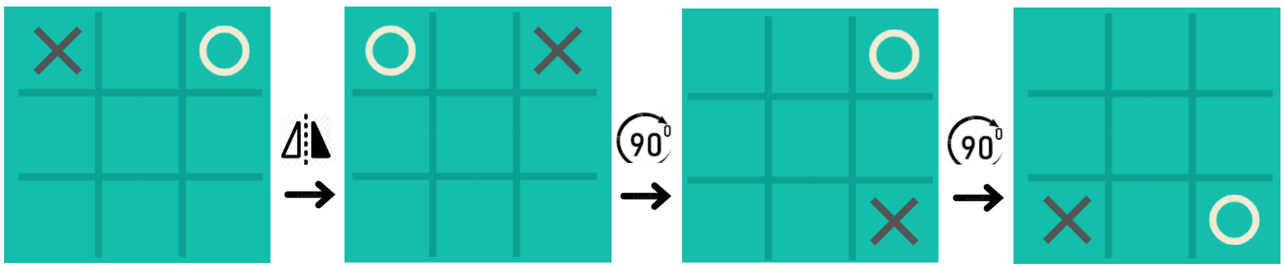
- In such cases where we try to look for similar patterns in the file and pick a bead, it's a bit complex to connect back to the actual game that is running. Below are the instructions for coming back to the actual game.

Initial Operation		**Retrieval Operation**	
Vertical Flips (1st)	**Rotations (2nd)**	**Vertical Flips (1st)**	**Rotations (2nd)**
0	1	0	3
0	2	0	2
0	3	0	1
1	0	1	0
1	1	1	1
1	2	1	2
1	3	1	3

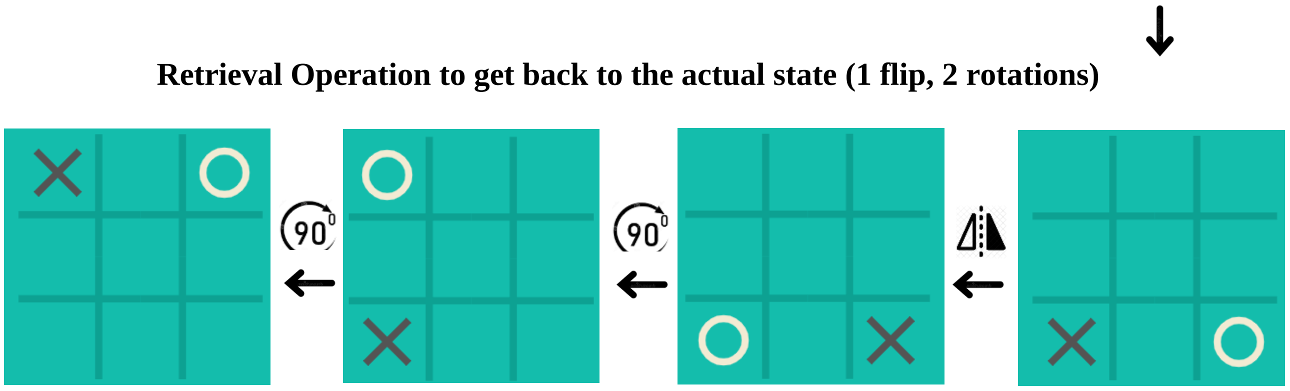
Note: *Whenever we talk about rotation, it's 90 degrees clockwise rotation & for flipping, its mirror image w.r.t vertical axis*

- Please find the below sample state retrieval process

Initial Operation to match the pattern in the beads_file (1 flip, 2 rotations)



Retrieval Operation to get back to the actual state (1 flip, 2 rotations)



- There are only 4 possible outcomes which are continuously monitored after each turn of Menace / Human and they are "*Menace Wins*", "*Human Wins*", "*Draw*" and "*In Progress*".
- The players keep playing as long as the state of the game is "In progress" or till all the positions get filled.
- A key aspect here is to store all the moves played by the Menace which would help us during the process of rewarding.
- There are 3 possible scenarios in the process of Rewarding -

Game Result	**Reward**
Menace Wins	Add more beads of the same colour which are picked at different states of the game
Draw	Add beads of the same colour which are picked at different states of the game, but not as much as in the case if it would have won
Human Wins	Remove beads of the same colour which are picked at different states of the game as part of penalization

- In a draw, we are rewarding the Menace even though it didn't win because it tried hard not losing the game and we know in a tic-tac-toe game, "***the best play from both parties leads to a draw***".
- In a win, we are rewarding the Menace with more beads than the beads we remove in a defeat because "***a win is a rare scenario which happens only when the opposite player makes a wrong move***". So, the magnitude of reward is considered more than the magnitude of penalization.
- Once the game is over, we update the beads_data file (this is kind of a database for our project) with new count of beads for all the patterns the Menace has referred during the gameplay.
- There are two ways of running the game,
    1) Train the Menace
    2) Watch the final game

- Both the options are almost the same, but the only difference is that we display all moves by the players on the console and store them in log file in a final game while there is no console output and only the game result is stored and not the moves when we choose the training mode.
- We implemented logging requirement using slf4j and logback.

2. Program

2.1 Data Structures

Below are the data structures used in the project -

HashSet - to hold all the possible patterns of the game
HashMap - to hold key-value pair of pattern-count of beads
ArrayList - to hold all the states of the menace machine till the end of game

2.2 Classes

Driver: This class has the main method for the project. All the User Inputs like parameter changes, number of iterations of the training/final games are read from the main method.

Game: It has methods to train games and watch final games.

OptimalStrategy: This class has methods to generate the ideal move to be followed by the human player based on the current state of the menace machine.

Human: It has methods to generate human player's next move based on the optimal strategy and "p" which is responsible for randomness in the moves of the human player.

Operations: This class is the powerhouse of the project. It has methods which could handle operations like array rotation, array mirror image, print array, string to array conversion, array to string conversion, picking random elements from a list, identifying similar arrays, and checking the state of the game.

Permutations: It contains the logic to find all the possible permutations of a string.

Rewards: This class deals with the rewarding process of the menace machine based on its actions and the result of the game.

BeadsFile: It deals with creating the database of the project which is a table of all possible patterns the menace machine could encounter during its game play along with their respective beads count. (We found that there are 304 possible patterns after eliminating the similar ones)

CSVreadWrite: This class contains methods to read and write csv files.

2.3 Invariants

Below are the invariants for this project scope

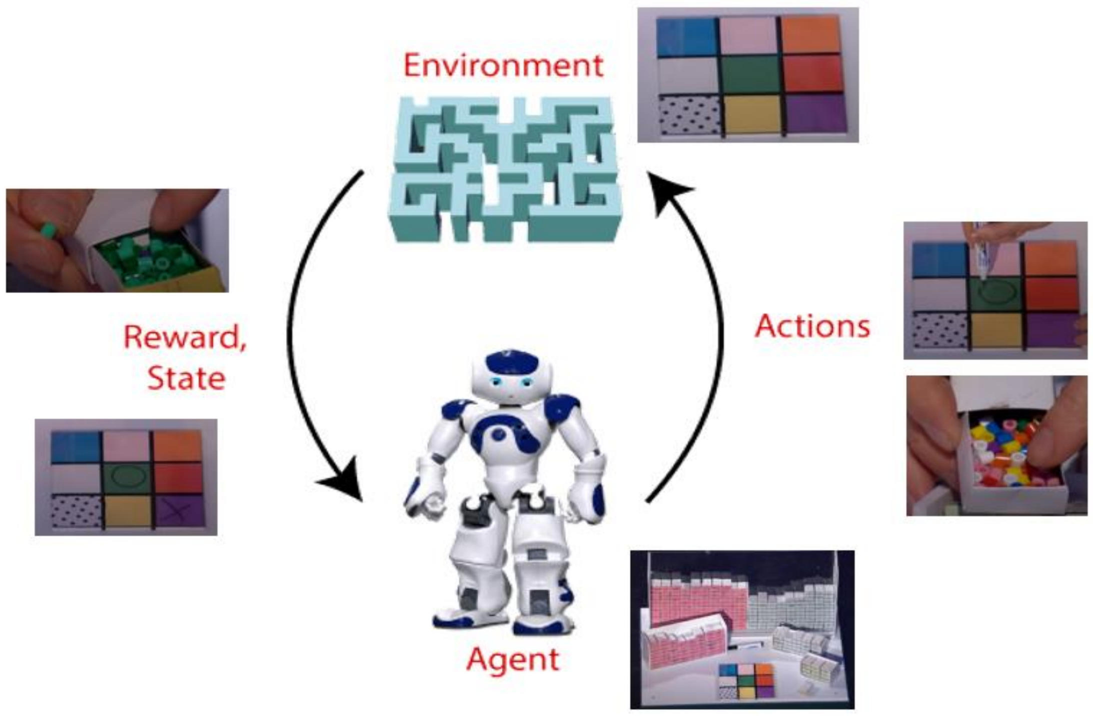
- Menace always plays first.
- Menace marks "X" as its symbol and Human uses "O".

- Both the players take alternate turns till the end of game or till all the positions gets filled.
- For every Menace move, program refers to beads file and pick a bead based on its current state pattern.
- At the end of game, beads of all the patterns that have been referred are either rewarded or penalized based on the game result.
- Irrespective of the user input of training the menace or watching the final game, the Menace gets trained in both the instances.

3. Flow Charts

The current problem statement involves reinforcement learning.

Elements of Reinforcement Learning



- Reinforcement learning is a machine learning training method based on rewarding desired behaviors and/or punishing undesired ones.
- In general, a reinforcement learning agent can perceive and interpret its environment, take actions, and learn through trial and error.
- Below are some of the elements of reinforcement learning and how our current problem statement can relate to these elements.

Agent: An entity that can perceive/explore the environment and act upon it. In this case, Menace machine is the agent.

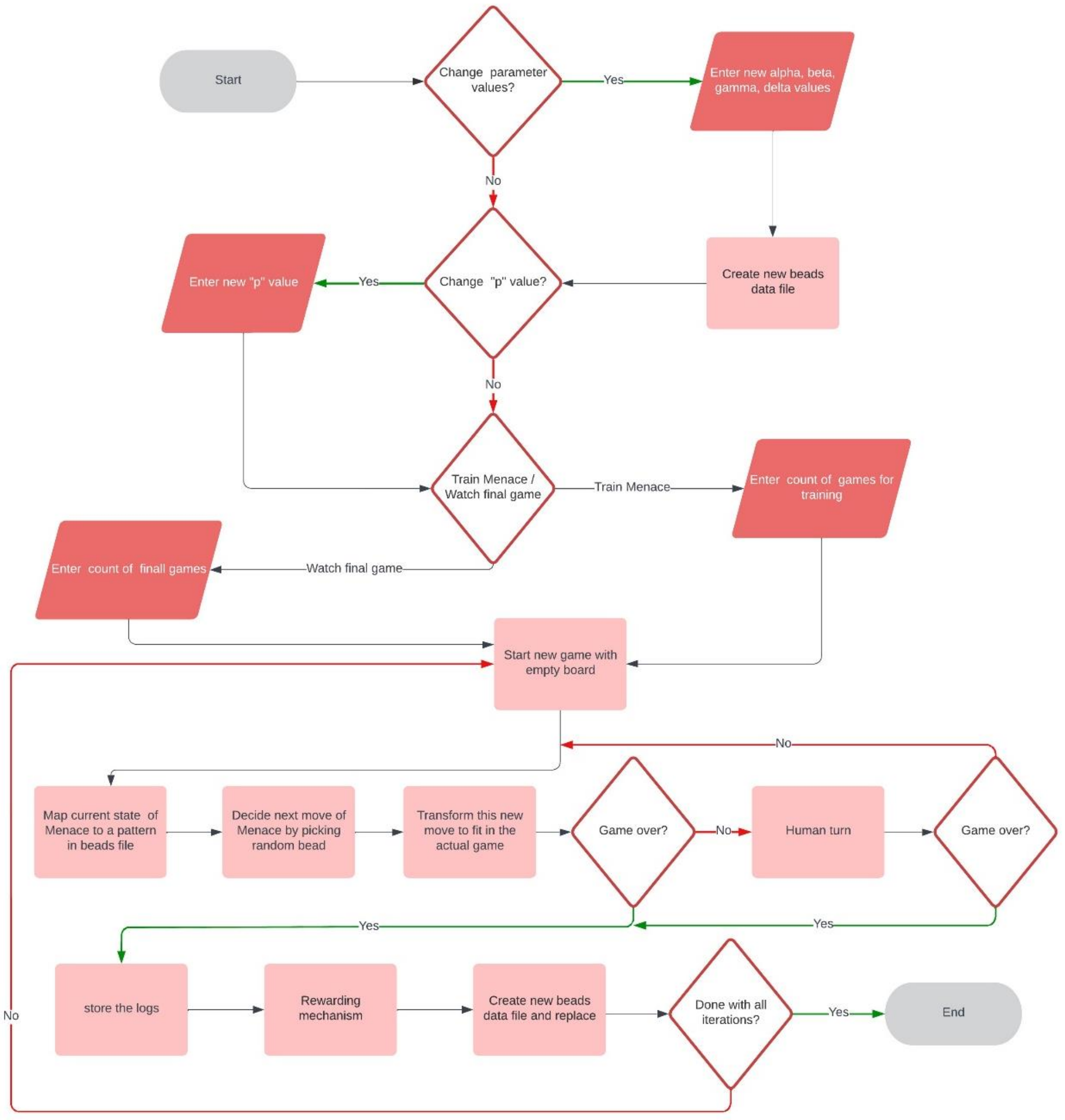
Environment: A situation in which an agent is present or surrounded by. The tic-tac-toe game is the environment.

Action: Actions are the moves taken by an agent within the environment. Picking beads and marking the position is the action.

State: State is a situation returned by the environment after each action taken by the agent. Menace changes its state by taking an action i.e., by marking on the board. In our case, board with all the markings on it denotes a state.

Reward: A feedback returned to the agent from the environment to evaluate the action of the agent. We are rewarding the Menace by altering the count of beads in the match boxes based on game result.

Overall flow of the program


Start
Change parameter values?
Yes
Enter new alpha, beta, gamma, delta values
No
Create new beads data file
Change "p" value?
Yes
Enter new "p" value
No
Train Menace / Watch final game
Train Menace
Enter count of games for training
Watch final game
Enter count of finall games
Start new game with empty board
No
Map current state of Menace to a pattern in beads file
Decide next move of Menace by picking random bead
Transform this new move to fit in the actual game
Game over?
No
Human turn
Game over?
Yes
Yes
store the logs
Rewarding mechanism
Create new beads data file and replace
Done with all iterations?
Yes
End
No

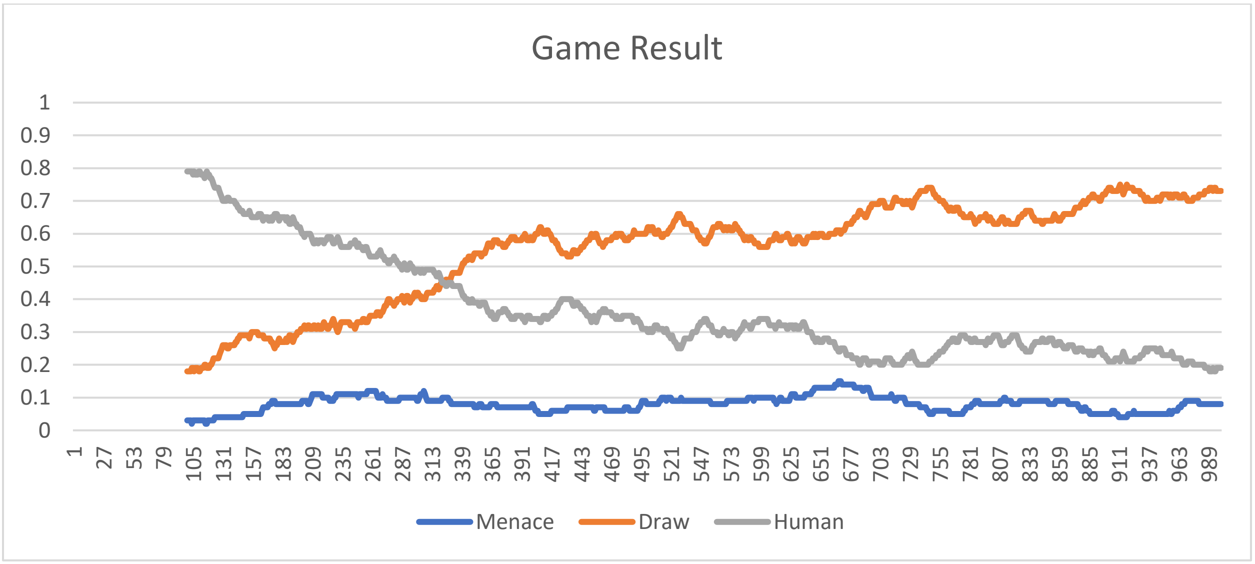
4. Mathematical Analysis & Graphical Analysis

Below is the analysis for selection of parameters for training the Menace.

4.1 Selection of alpha

- Various combinations of alpha have been analyzed based on the game result and the best value of alpha is [8,4,2,1].
- We have not gone anything below 4 for initial states of the Menace, because there is a chance of running out of beads while training.
- Below are some of the combinations of alpha which have been analyzed.

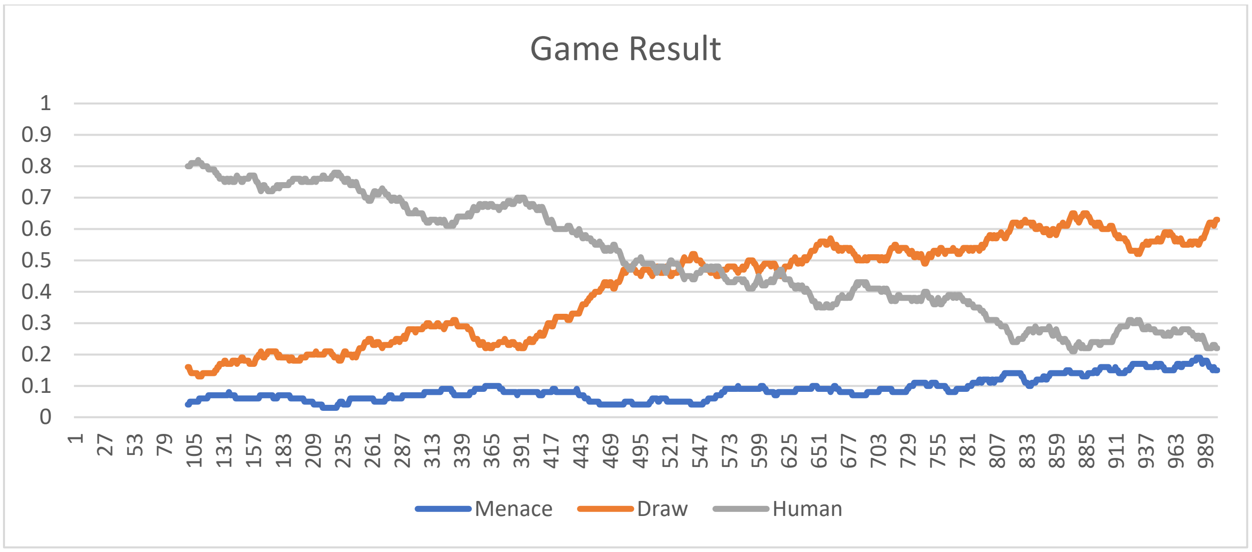
Case-1) alpha = [8,4,2,1], beta = 3, gamma = 1, delta =1, p = 0.9, games = 1000



Menace Wins	***Draws***	***Human Wins***
8%	73%	19%

Note: Percentages are calculated based on the recent 100 matches played

Case-2) alpha = [8,8,8,8], beta = 3, gamma = 1, delta =1, p = 0.9, games = 1000

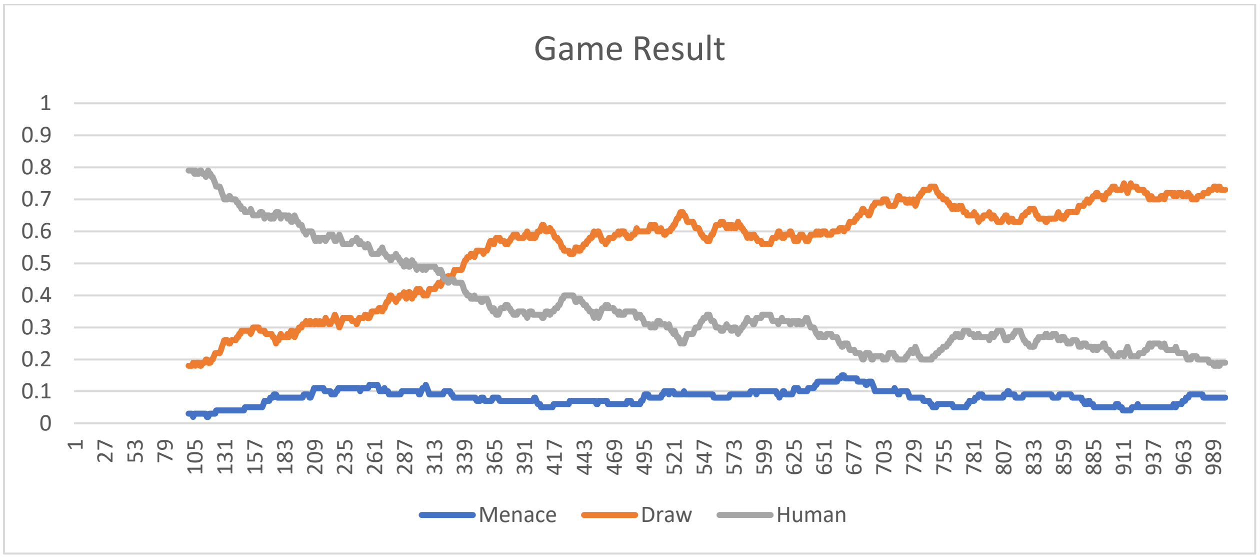


Menace Wins	***Draws***	***Human Wins***
15%	63%	22%

4.2 <u>Selection of beta, gamma & delta</u>

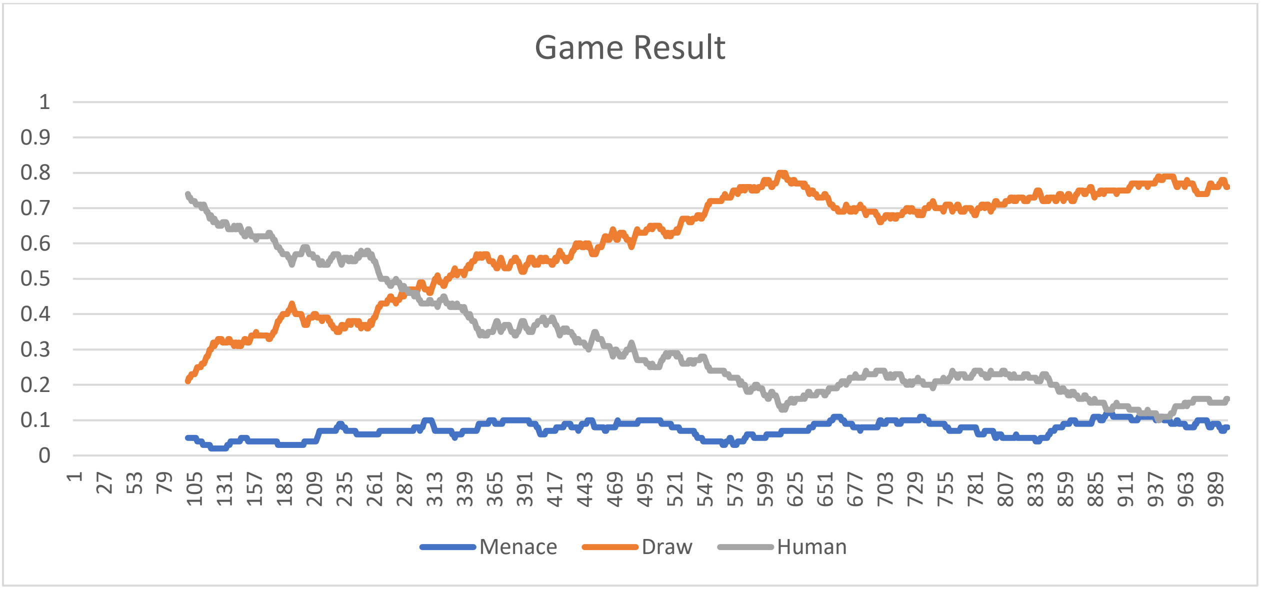
- Various combinations of beta, gamma and delta have been analyzed based on the game result and the best possible combination is beta = 2, gamma = 1 and delta = 1
- We have not gone beta value of anything below 2 because we want to keep the magnitude of reward more than penalization.
- Below are some of the combinations of beta, gamma and delta which have been analyzed.

Case-1) alpha = [8,4,2,1], beta = 3, gamma = 1, delta =1, p = 0.9, games = 1000



Menace Wins	*Draws*	*Human Wins*
8%	73%	19%

Case-2) alpha = [8,4,2,1], beta = 2, gamma = 1, delta =1, p = 0.9, games = 1000

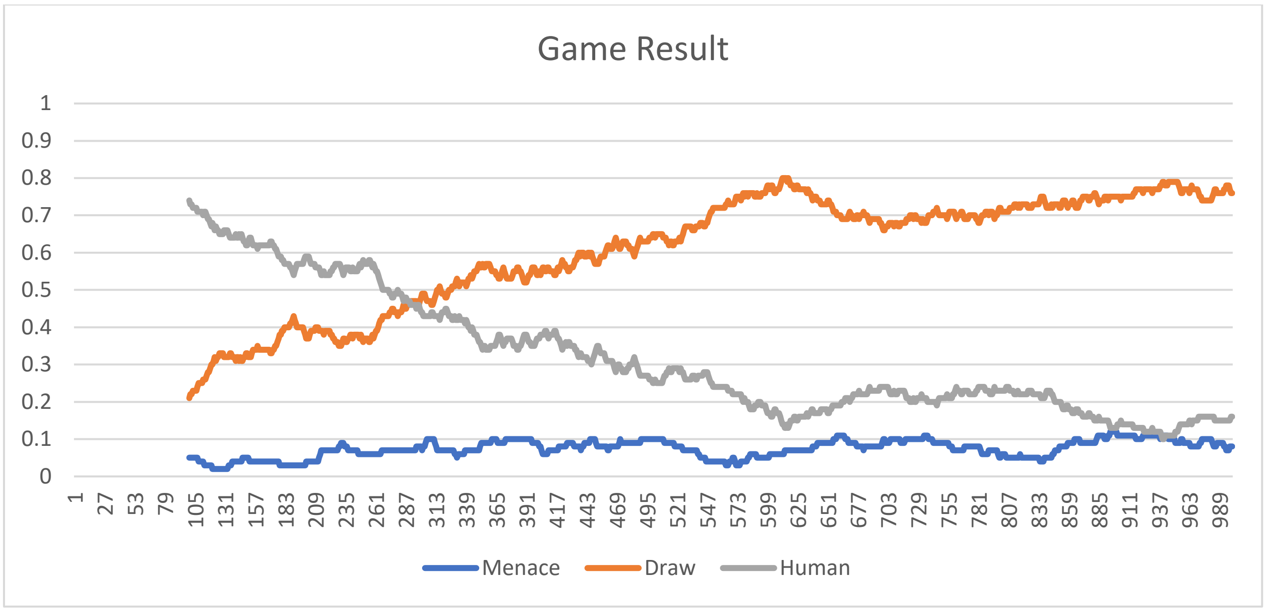


Menace Wins	*Draws*	*Human Wins*
8%	76%	16%

4.3 <u>Strategy for training (Altering "p" while training)</u>

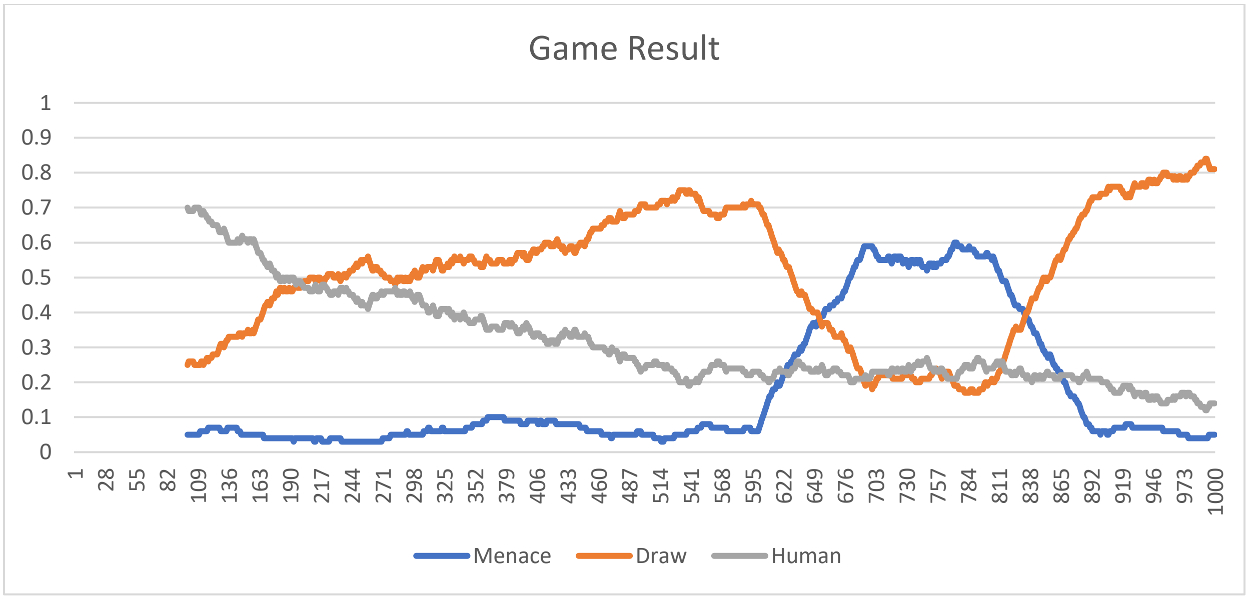
- We have tried altering the value of "p" while training the Menace machine to make sure it can play well against the Human not only when the human plays optimal strategy but also when the human plays random.
- In case 1, we trained the Menace against a "p" value of 0.9 for 1000 games. But, in case 2, we trained the menace against a "p" value of 0.9 for 800 games and against a "p" value of 0.1 for 200 games.
- We found that training the Menace against "p" value of 0.1 for few games is giving better results.

Case-1) alpha = [8,4,2,1], beta = 2, gamma = 1, delta =1, p = 0.9, games = 1000



Menace Wins	*Draws*	*Human Wins*
8%	76%	16%

Case-2)
alpha = [8,4,2,1], beta = 2, gamma = 1, delta =1, p = 0.9, games = 600
alpha = [8,4,2,1], beta = 2, gamma = 1, delta =1, p = 0.1, games = 200
alpha = [8,4,2,1], beta = 2, gamma = 1, delta =1, p = 0.9, games = 200



Menace Wins	*Draws*	*Human Wins*
5%	81%	14%

5. Conclusion

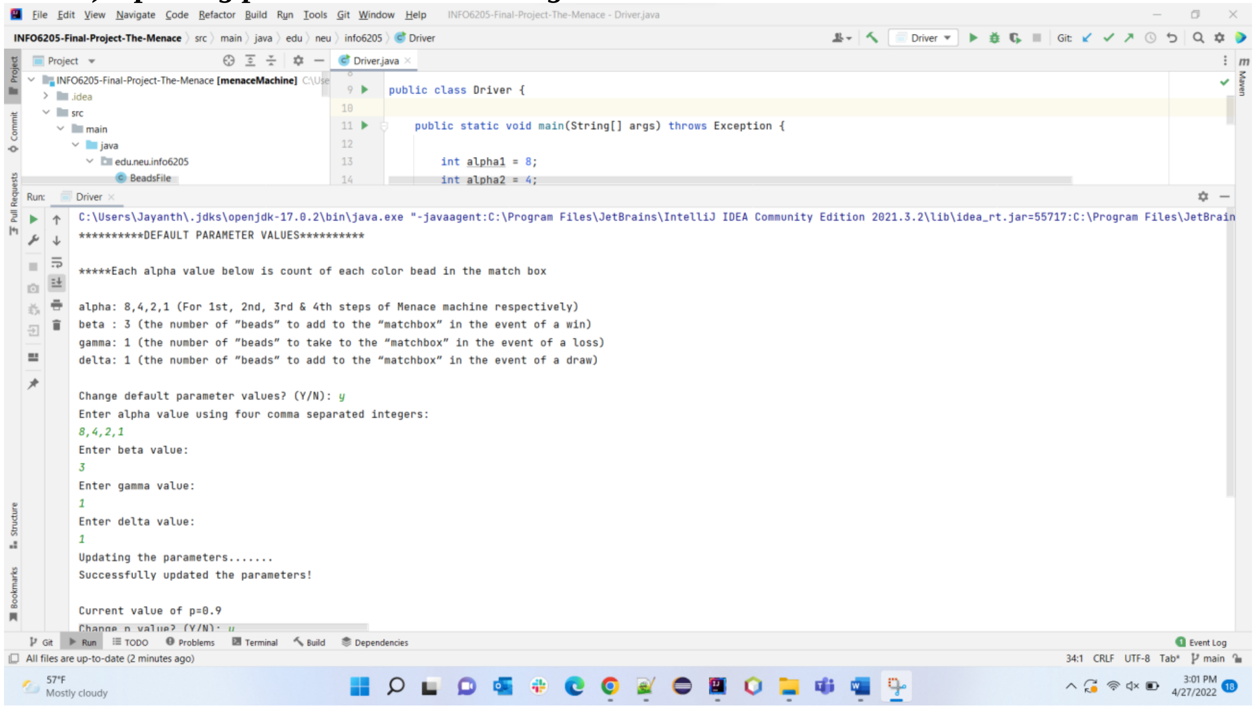
- The Menace machine was able to tackle the Human Strategy in most cases (loose percentage is less than 15%).
- Optimal set of parameters are alpha = [8,4,2,1], beta = 2, gamma = 1, delta = 1
- Also, we made sure to train the Menace play against low “p” value of 0.1 for few games to tackle the random play of the Human and it improved the results. (increased draw percentage from 76% to 81%)
- The reason for this improvement is because the Menace has been playing against Optimal strategy for most of the time and learnt to play well against the best moves, but the issue is that it is not effective against the random play because it never played much against this kind of play. And when we lower the “p” value and train the Menace for few more matches, it learns to play against random moves and it in turn improves the draw percentage even when the “p” value is set back to 0.9.

6. Future Enhancements

- Implementing GUI
- Allowing human player also to take the first chance
- Allow any player to take “X” or “O”

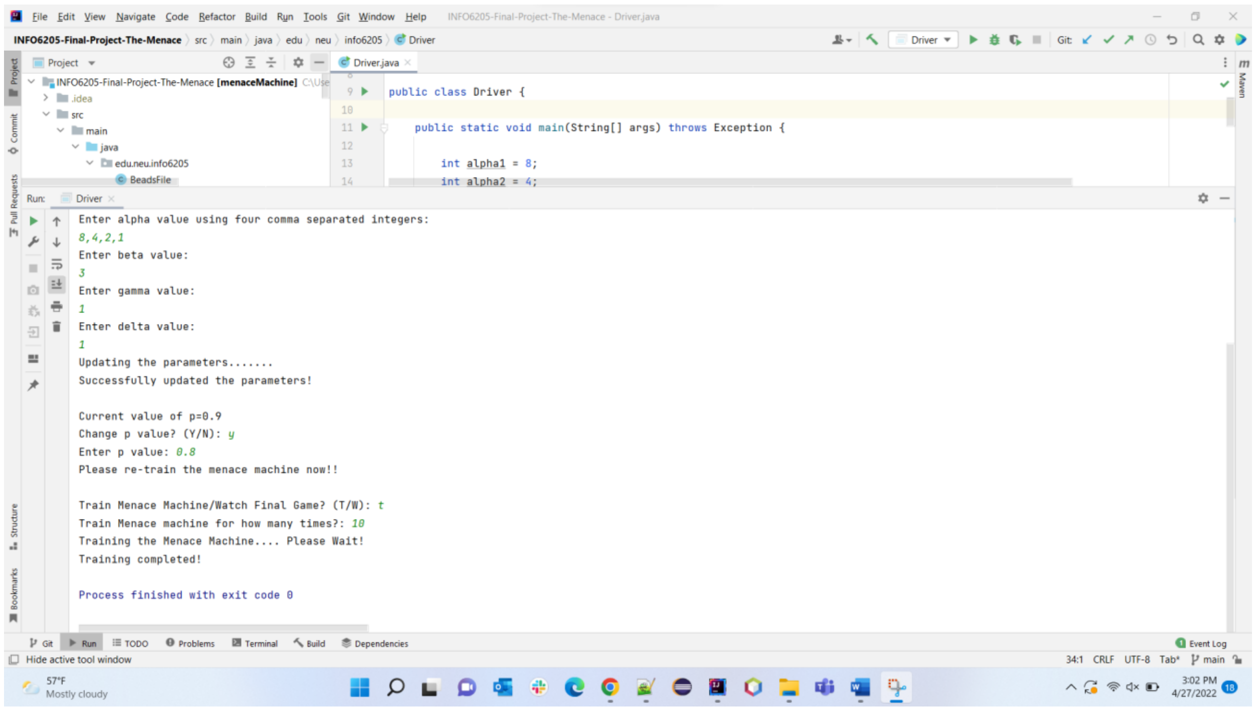
7. Console output screenshots

Case-1) Updating parameters and training the Menace



```java
public class Driver {

    public static void main(String[] args) throws Exception {

        int alpha1 = 8;
        int alpha2 = 4;
```

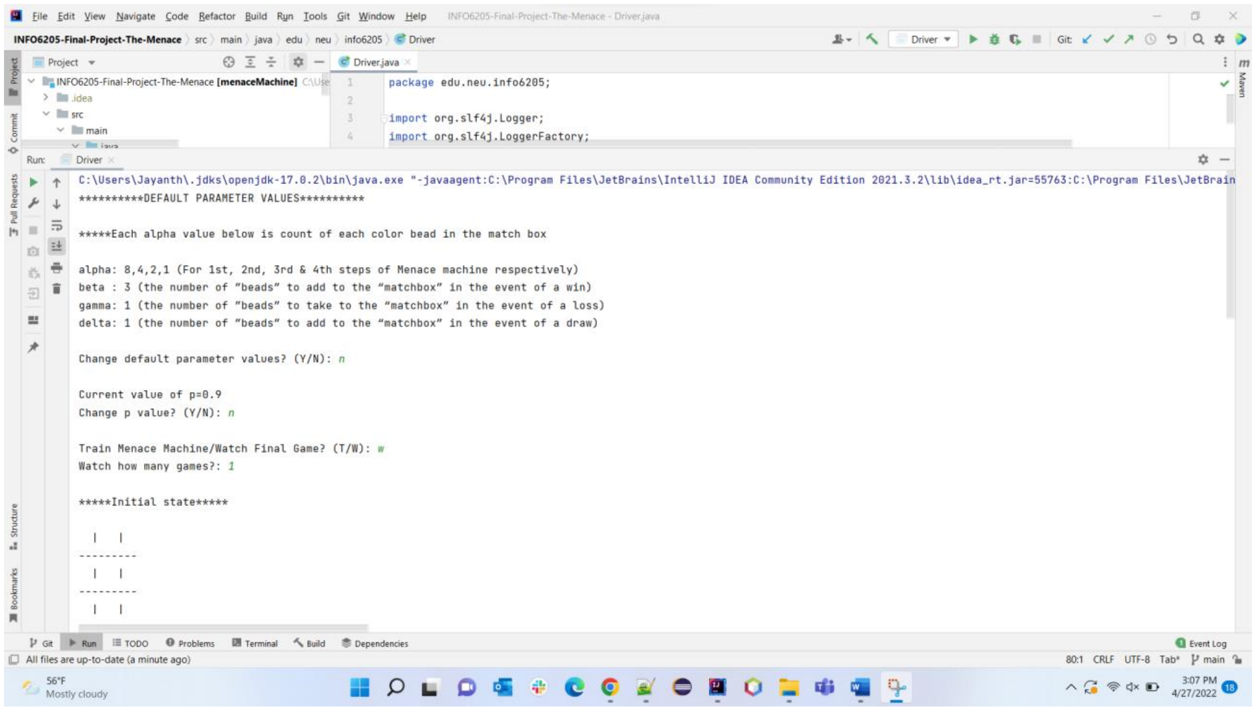
```
C:\Users\Jayanth\.jdks\openjdk-17.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2021.3.2\lib\idea_rt.jar=55717:C:\Program Files\JetBrain
**********DEFAULT PARAMETER VALUES**********

*****Each alpha value below is count of each color bead in the match box

alpha: 8,4,2,1 (For 1st, 2nd, 3rd & 4th steps of Menace machine respectively)
beta : 3 (the number of "beads" to add to the "matchbox" in the event of a win)
gamma: 1 (the number of "beads" to take to the "matchbox" in the event of a loss)
delta: 1 (the number of "beads" to add to the "matchbox" in the event of a draw)

Change default parameter values? (Y/N): y
Enter alpha value using four comma separated integers:
8,4,2,1
Enter beta value:
3
Enter gamma value:
1
Enter delta value:
1
Updating the parameters.......
Successfully updated the parameters!

Current value of p=0.9
Change p value? (Y/N): y
```



```
public class Driver {

    public static void main(String[] args) throws Exception {

        int alpha1 = 8;
        int alpha2 = 4;
```

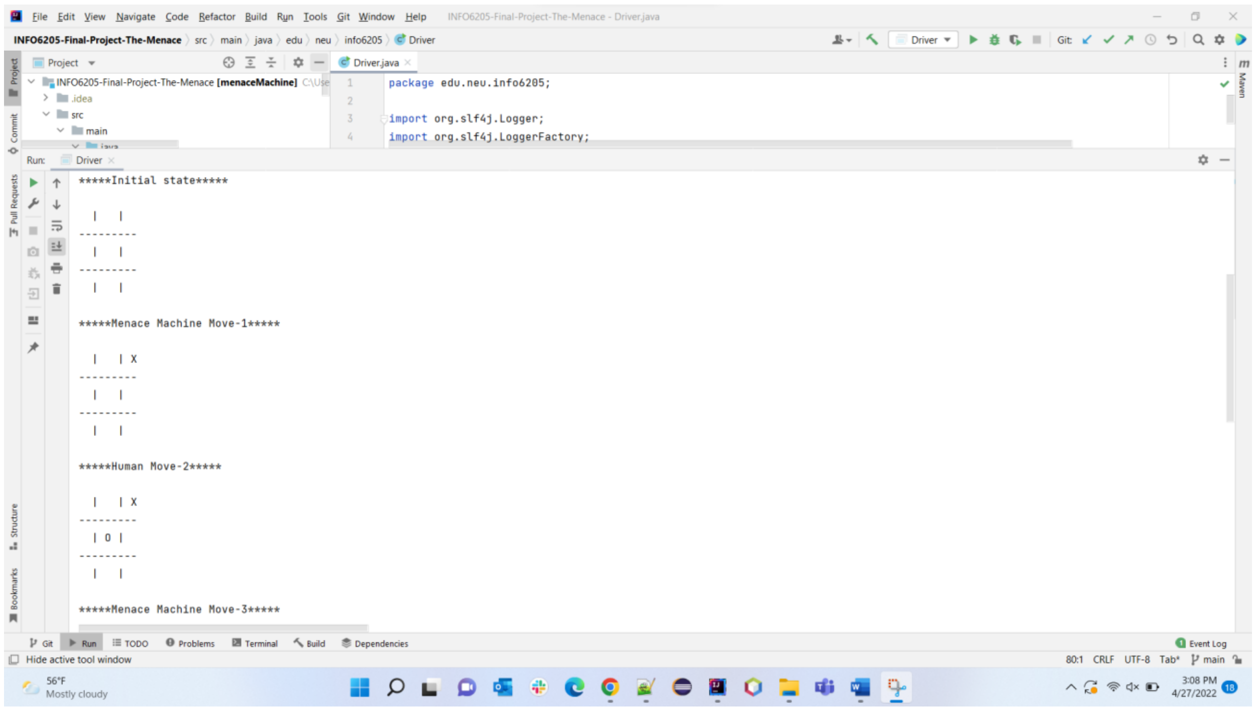
```
Enter alpha value using four comma separated integers:
8,4,2,1
Enter beta value:
3
Enter gamma value:
1
Enter delta value:
1
Updating the parameters.......
Successfully updated the parameters!

Current value of p=0.9
Change p value? (Y/N): y
Enter p value: 0.8
Please re-train the menace machine now!!

Train Menace Machine/Watch Final Game? (T/W): t
Train Menace machine for how many times?: 10
Training the Menace Machine.... Please Wait!
Training completed!

Process finished with exit code 0
```

Case-2) not updating parameters and watching the final game



```
package edu.neu.info6205;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

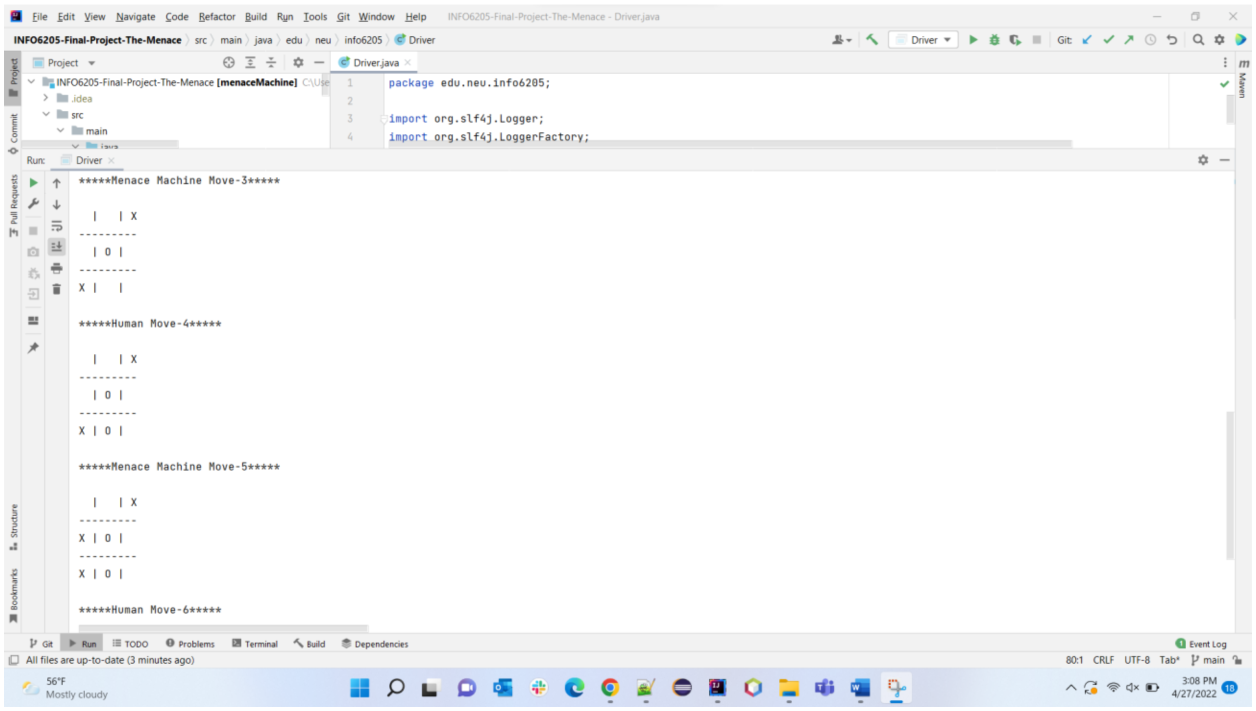
```
C:\Users\Jayanth\.jdks\openjdk-17.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2021.3.2\lib\idea_rt.jar=55763:C:\Program Files\JetBrain
**********DEFAULT PARAMETER VALUES**********

*****Each alpha value below is count of each color bead in the match box

alpha: 8,4,2,1 (For 1st, 2nd, 3rd & 4th steps of Menace machine respectively)
beta : 3 (the number of "beads" to add to the "matchbox" in the event of a win)
gamma: 1 (the number of "beads" to take to the "matchbox" in the event of a loss)
delta: 1 (the number of "beads" to add to the "matchbox" in the event of a draw)

Change default parameter values? (Y/N): n

Current value of p=0.9
Change p value? (Y/N): n

Train Menace Machine/Watch Final Game? (T/W): w
Watch how many games?: 1

*****Initial state*****

   |   |
---------
   |   |
---------
   |   |
```



```java
package edu.neu.info6205;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```
*****Initial state*****

  |   |
---------
  |   |
---------
  |   |

*****Menace Machine Move-1*****

  |   | X
---------
  |   |
---------
  |   |

*****Human Move-2*****

  |   | X
---------
  | 0 |
---------
  |   |

*****Menace Machine Move-3*****
```

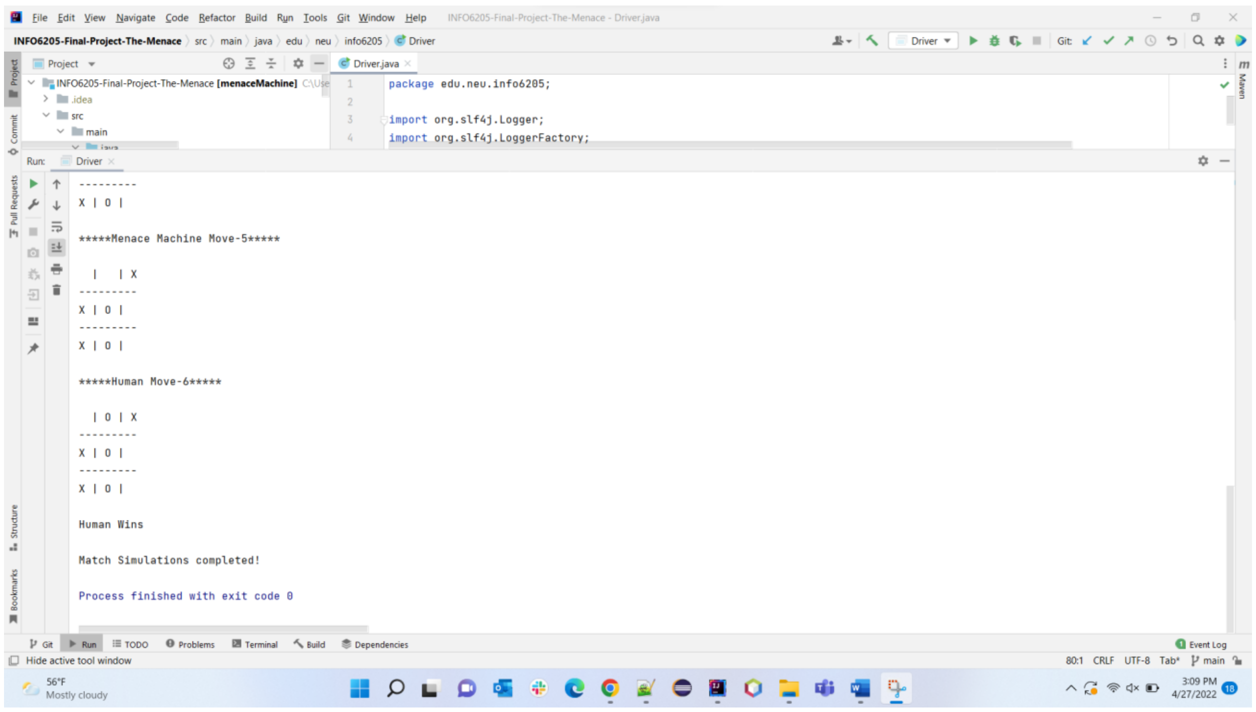
Hide active tool window



```java
package edu.neu.info6205;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```
*****Menace Machine Move-3*****

  |   | X
---------
  | 0 |
---------
X |   |

*****Human Move-4*****

  |   | X
---------
  | 0 |
---------
X | 0 |

*****Menace Machine Move-5*****

  |   | X
---------
X | 0 |
---------
X | 0 |

*****Human Move-6*****
```

All files are up-to-date (3 minutes ago)



```
package edu.neu.info6205;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```
---------
X | O |

*****Menace Machine Move-5*****

  |   | X
---------
X | O |
---------
X | O |

*****Human Move-6*****

  | O | X
---------
X | O |
---------
X | O |

Human Wins

Match Simulations completed!

Process finished with exit code 0
```

8. Console output

Case-1) Updating parameters and training the Menace

```
**********DEFAULT PARAMETER VALUES**********

*****Each alpha value below is count of each color bead in the match box

alpha: 8,4,2,1 (For 1st, 2nd, 3rd & 4th steps of Menace machine respectively)
beta : 3 (the number of ”beads” to add to the “matchbox” in the event of a win)
gamma: 1 (the number of ”beads” to take to the “matchbox” in the event of a loss)
delta: 1 (the number of ”beads” to add to the “matchbox” in the event of a draw)

Change default parameter values? (Y/N): y
Enter alpha value using four comma separated integers:
8,4,2,1
Enter beta value:
3
Enter gamma value:
1
Enter delta value:
1
Updating the parameters.......
Successfully updated the parameters!

Current value of p=0.9
Change p value? (Y/N): y
Enter p value: 0.8
```

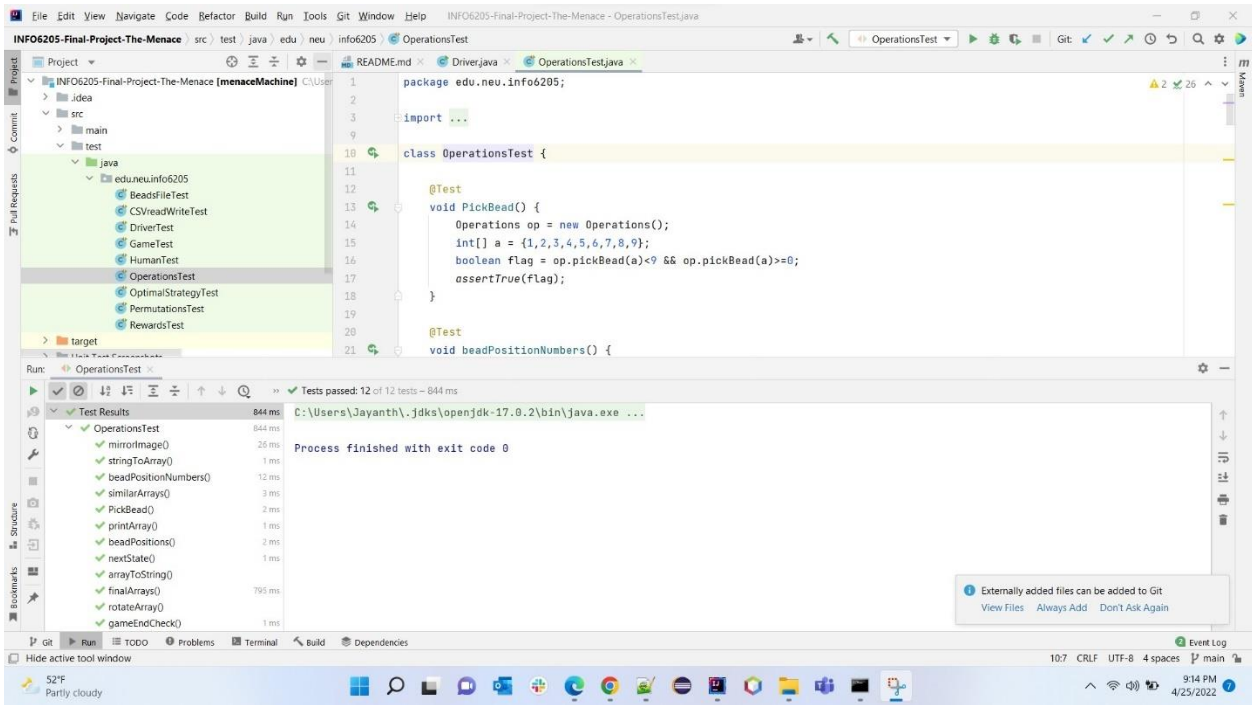
```
Please re-train the menace machine now!!

Train Menace Machine/Watch Final Game? (T/W): t
Train Menace machine for how many times?: 10
Training the Menace Machine.... Please Wait!
Training completed!

Process finished with exit code 0
```

Case-2) not updating parameters and watching the final game

```
**********DEFAULT PARAMETER VALUES**********

*****Each alpha value below is count of each color bead in the match box

alpha: 8,4,2,1 (For 1st, 2nd, 3rd & 4th steps of Menace machine respectively)
beta : 3 (the number of ”beads” to add to the “matchbox” in the event of a win)
gamma: 1 (the number of ”beads” to take to the “matchbox” in the event of a loss)
delta: 1 (the number of ”beads” to add to the “matchbox” in the event of a draw)

Change default parameter values? (Y/N): n

Current value of p=0.9
Change p value? (Y/N): n

Train Menace Machine/Watch Final Game? (T/W): w
Watch how many games?: 1

*****Initial state*****

  |  |
---------
  |  |
---------
  |  |

*****Menace Machine Move-1*****

  |  | X
---------
  |  |
---------
  |  |

*****Human Move-2*****

  |  | X
---------
  | O |
---------
  |  |

*****Menace Machine Move-3*****

  |  | X
---------
  | O |
---------
X |  |

*****Human Move-4*****

  |  | X
---------
  | O |
---------
X | O |

*****Menace Machine Move-5*****

  |  | X
---------
X | O |
---------
X | O |

*****Human Move-6*****

  | O | X
---------
X | O |
---------
X | O |

Human Wins

Match Simulations completed!

Process finished with exit code 0
```

9. Unit tests screenshots

OperationsTest.java



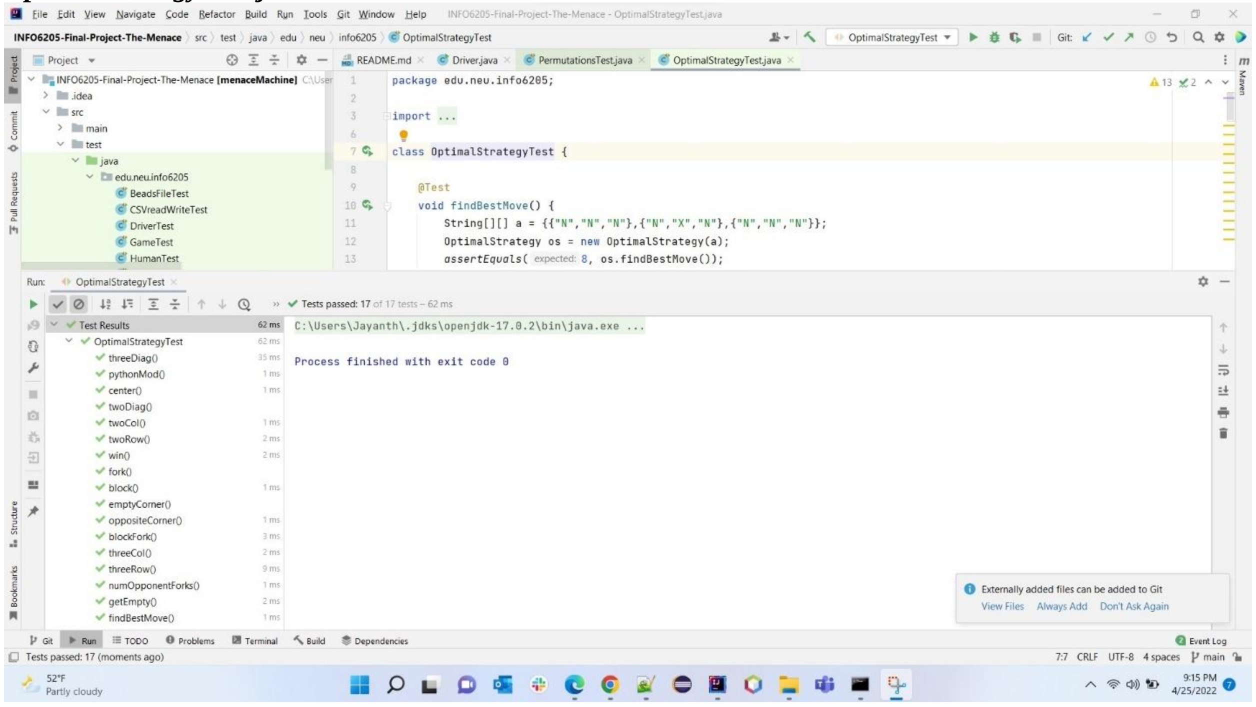
```
package edu.neu.info6205;

import ...

class OperationsTest {

    @Test
    void PickBead() {
        Operations op = new Operations();
        int[] a = {1,2,3,4,5,6,7,8,9};
        boolean flag = op.pickBead(a)<9 && op.pickBead(a)>=0;
        assertTrue(flag);
    }

    @Test
    void beadPositionNumbers() {
```

Tests passed: 12 of 12 tests – 844 ms

```
Test Results            844 ms
  OperationsTest        844 ms
    mirrorImage()        26 ms
    stringToArray()       1 ms
    beadPositionNumbers() 12 ms
    similarArrays()       3 ms
    PickBead()            2 ms
    printArray()          1 ms
    beadPositions()       2 ms
    nextState()           1 ms
    arrayToString()
    finalArrays()       795 ms
    rotateArray()
    gameEndCheck()        1 ms
```

```
C:\Users\Jayanth\.jdks\openjdk-17.0.2\bin\java.exe ...

Process finished with exit code 0
```

OptimalStrategyTest.java



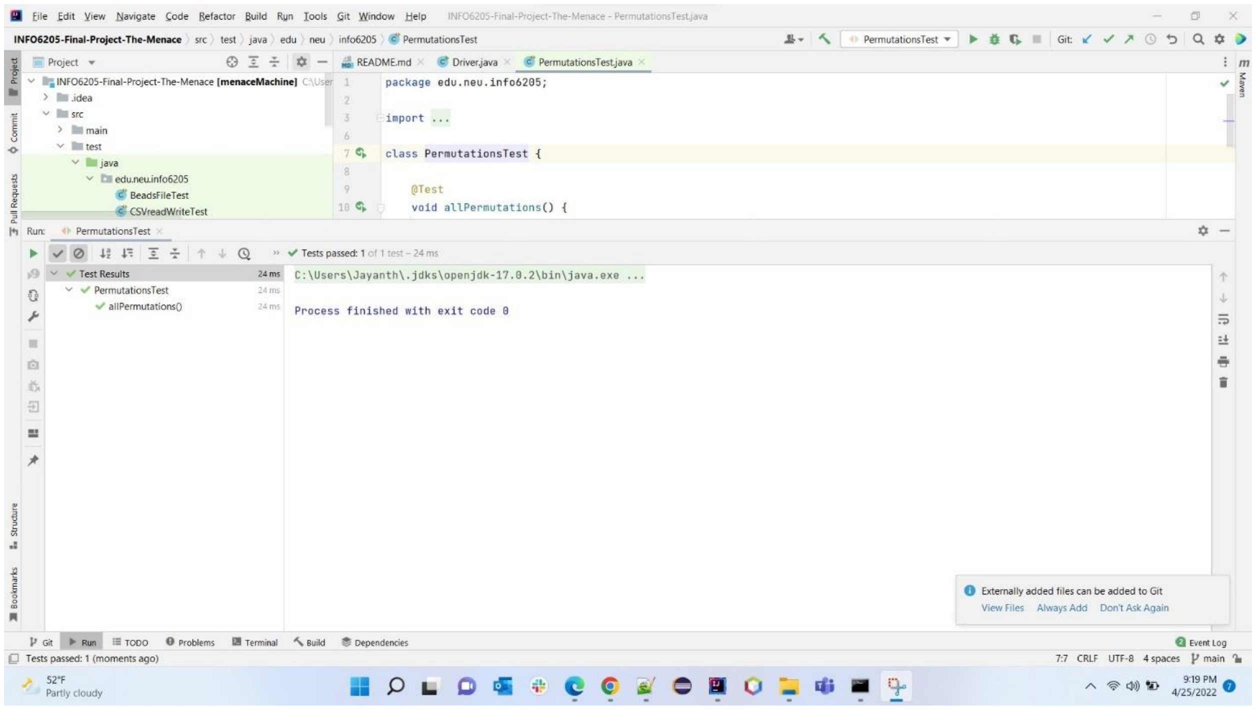
```
package edu.neu.info6205;

import ...

class OptimalStrategyTest {

    @Test
    void findBestMove() {
        String[][] a = {{"N","N","N"},{"N","X","N"},{"N","N","N"}};
        OptimalStrategy os = new OptimalStrategy(a);
        assertEquals( expected: 8, os.findBestMove());
```

Tests passed: 17 of 17 tests – 62 ms

```
Test Results            62 ms
  OptimalStrategyTest   62 ms
    threeDiag()         35 ms
    pythonMod()          1 ms
    center()             1 ms
    twoDiag()
    twoCol()             1 ms
    twoRow()             2 ms
    win()                2 ms
    fork()
    block()              1 ms
    emptyCorner()
    oppositeCorner()     1 ms
    blockFork()          3 ms
    threeCol()           2 ms
    threeRow()           9 ms
    numOpponentForks()   1 ms
    getEmpty()           2 ms
    findBestMove()       1 ms
```

```
C:\Users\Jayanth\.jdks\openjdk-17.0.2\bin\java.exe ...

Process finished with exit code 0
```

PermutationsTest.java

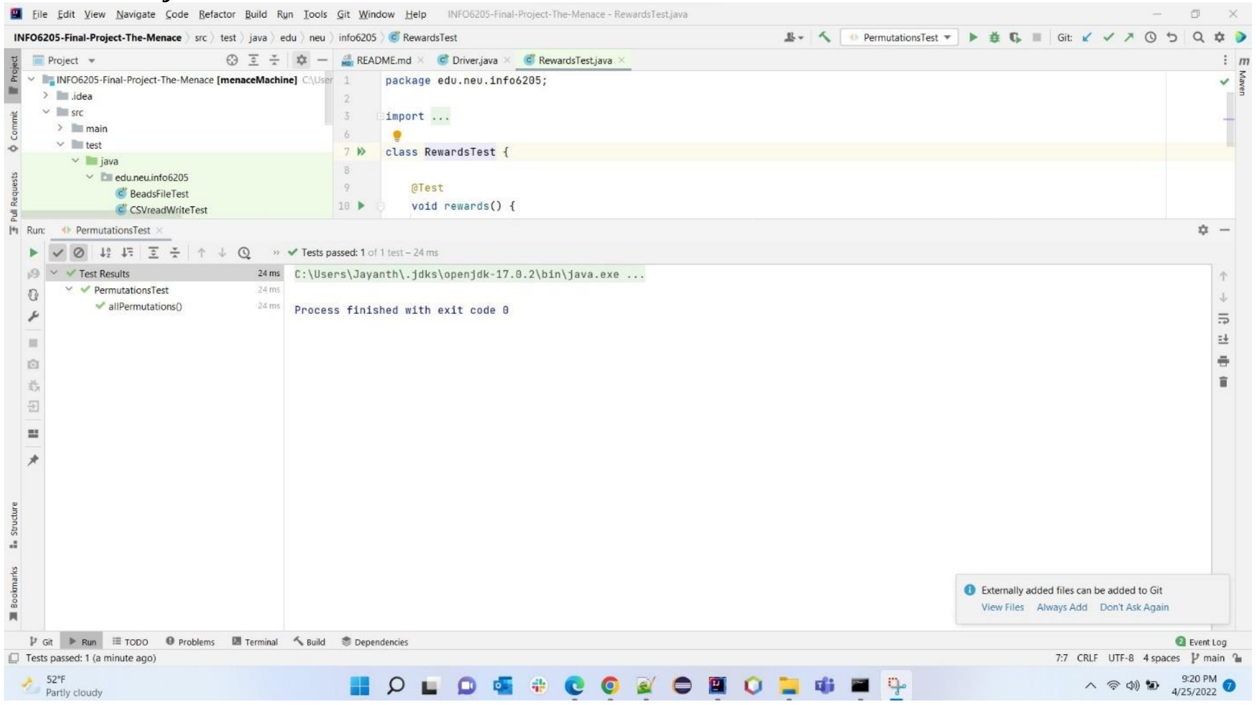


```
package edu.neu.info6205;

import ...

class PermutationsTest {

    @Test
    void allPermutations() {
```

Tests passed: 1 of 1 test – 24 ms

```
C:\Users\Jayanth\.jdks\openjdk-17.0.2\bin\java.exe ...

Process finished with exit code 0
```

RewardsTest.java

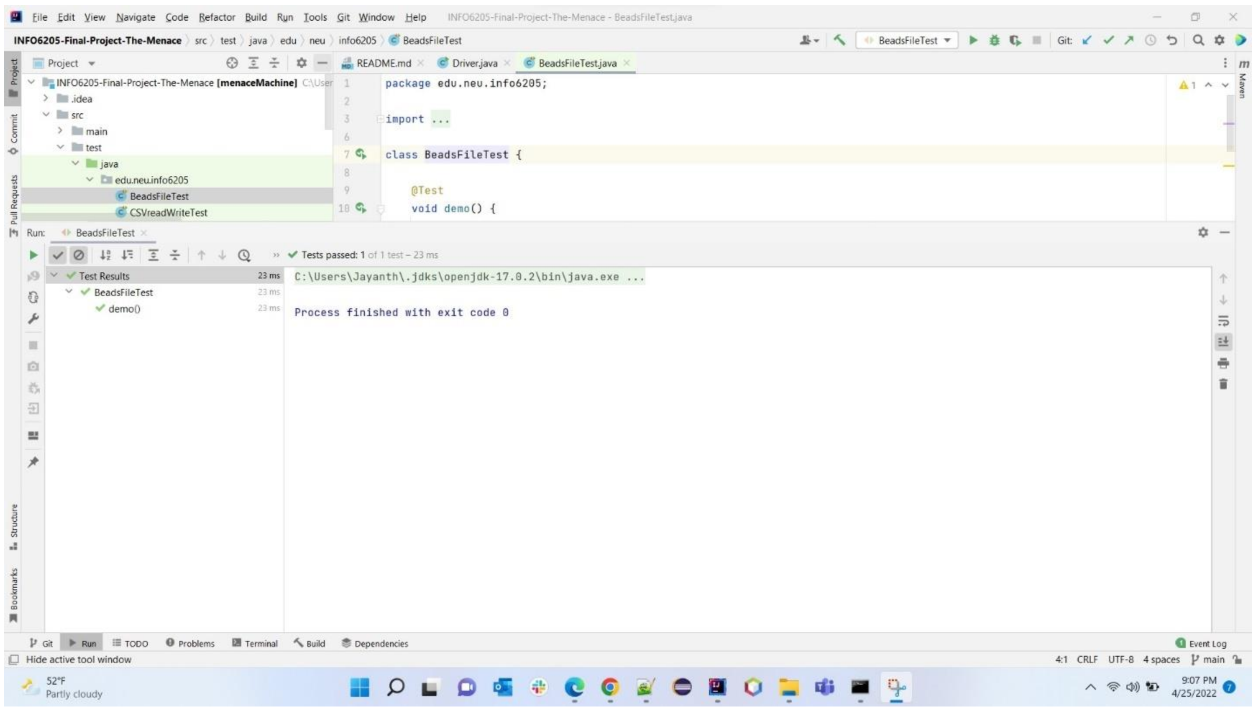


```
package edu.neu.info6205;

import ...

class RewardsTest {

    @Test
    void rewards() {
```

Tests passed: 1 of 1 test – 24 ms

```
C:\Users\Jayanth\.jdks\openjdk-17.0.2\bin\java.exe ...

Process finished with exit code 0
```

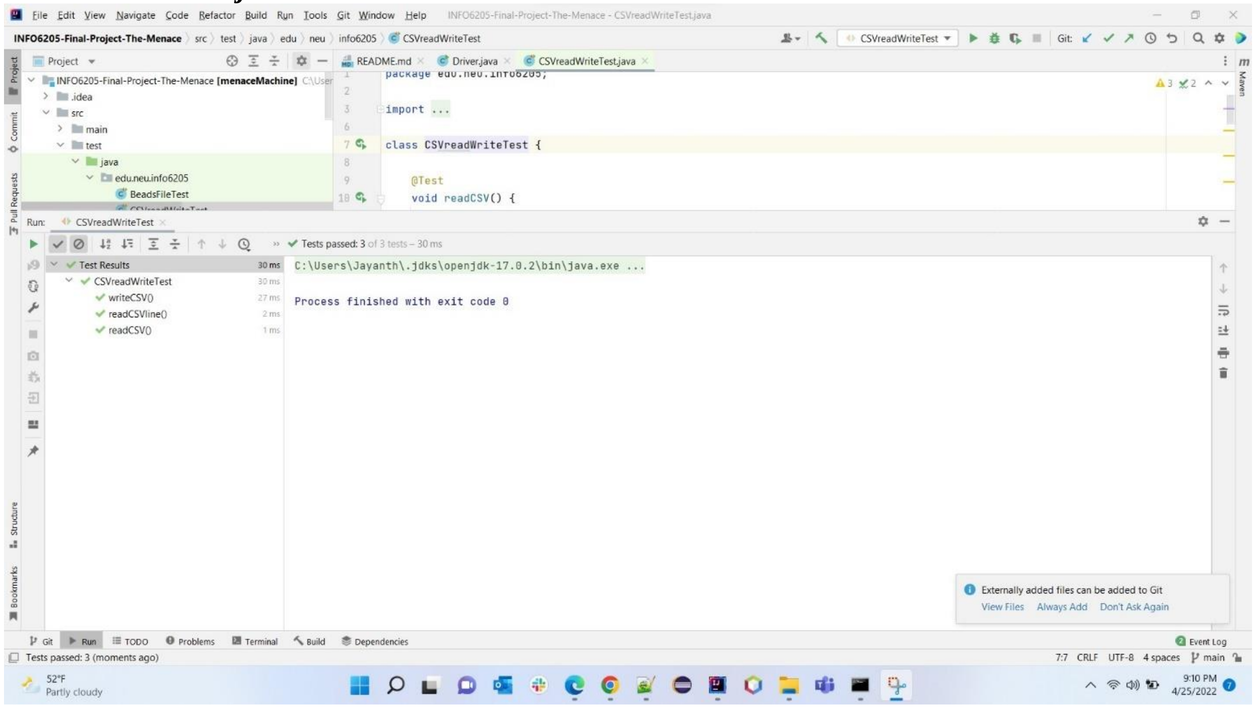
BeadsFileTest.java



```java
package edu.neu.info6205;

import ...

class BeadsFileTest {

    @Test
    void demo() {
```

```
Tests passed: 1 of 1 test – 23 ms

Test Results        23 ms
  BeadsFileTest     23 ms
    demo()          23 ms

C:\Users\Jayanth\.jdks\openjdk-17.0.2\bin\java.exe ...

Process finished with exit code 0
```

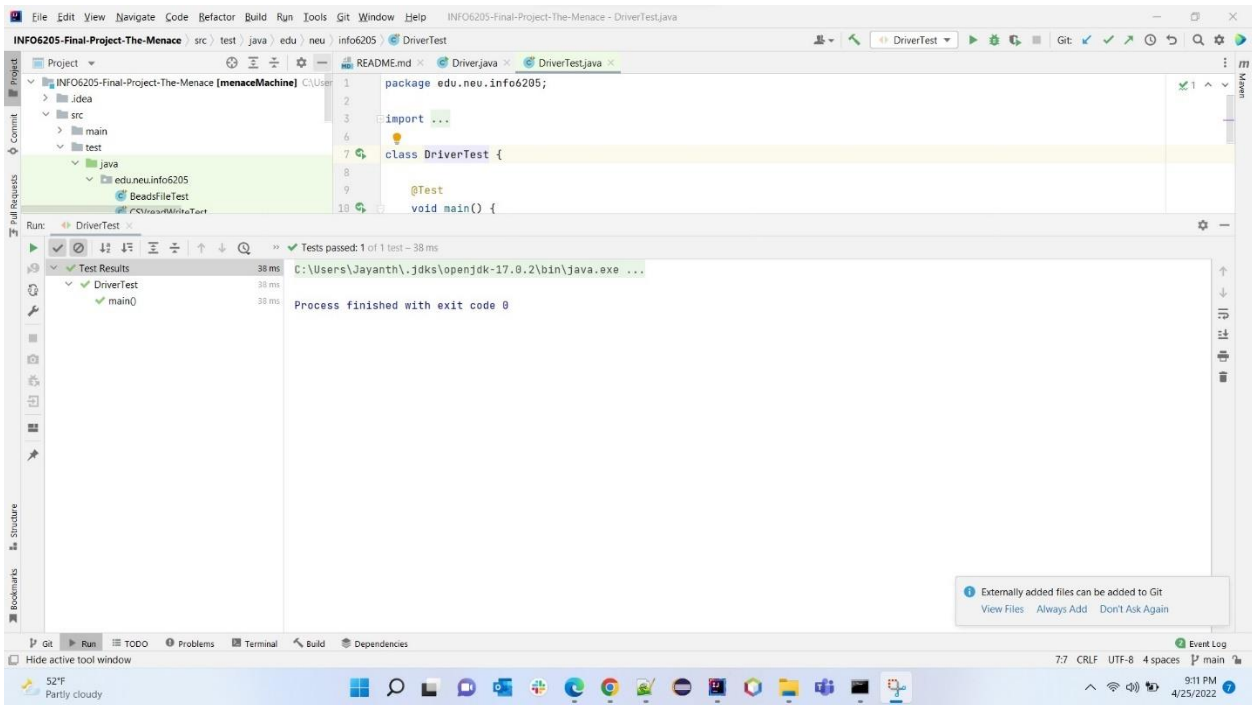
CSVreadWriteTest.java



```java
package edu.neu.info6205;

import ...

class CSVreadWriteTest {

    @Test
    void readCSV() {
```

```
Tests passed: 3 of 3 tests – 30 ms

Test Results          30 ms
  CSVreadWriteTest    30 ms
    writeCSV()        27 ms
    readCSVline()      2 ms
    readCSV()          1 ms

C:\Users\Jayanth\.jdks\openjdk-17.0.2\bin\java.exe ...

Process finished with exit code 0
```

DriverTest.java



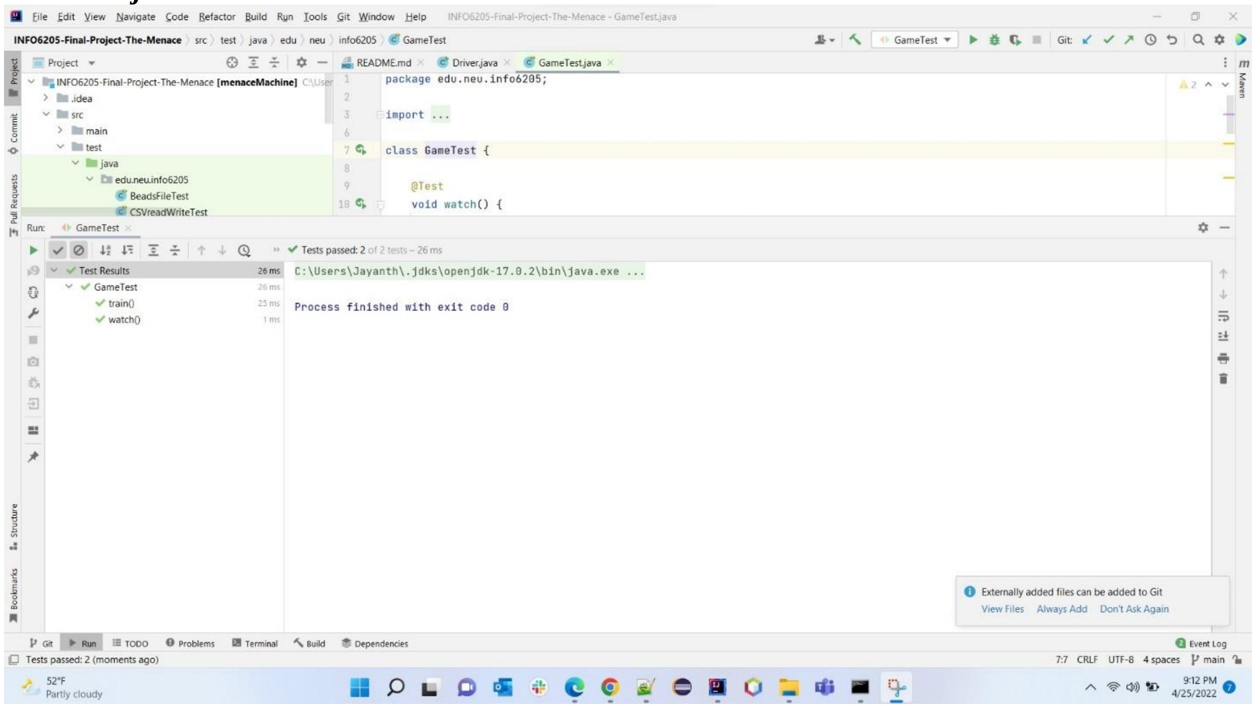
```
package edu.neu.info6205;

import ...

class DriverTest {

    @Test
    void main() {
```

Tests passed: 1 of 1 test – 38 ms

```
Test Results   38 ms
  DriverTest   38 ms
    main()     38 ms
```

```
C:\Users\Jayanth\.jdks\openjdk-17.0.2\bin\java.exe ...

Process finished with exit code 0
```

GameTest.java



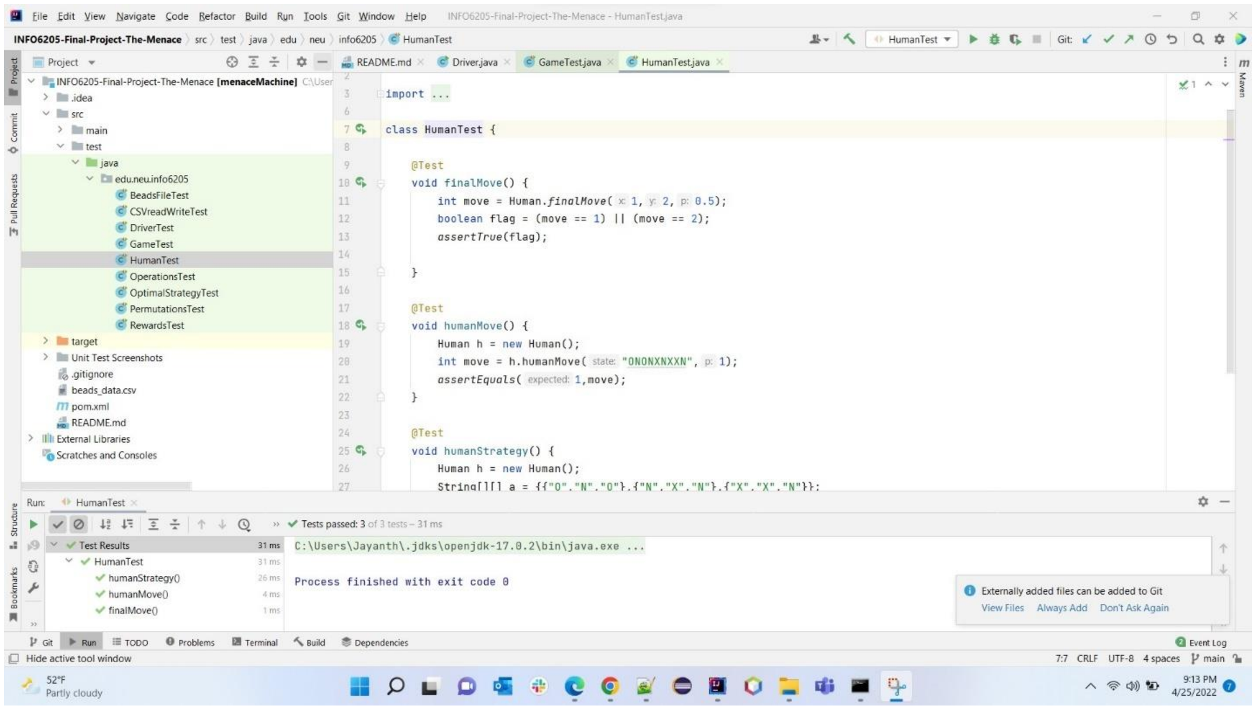
```
package edu.neu.info6205;

import ...

class GameTest {

    @Test
    void watch() {
```

Tests passed: 2 of 2 tests – 26 ms

```
Test Results   26 ms
  GameTest     26 ms
    train()    25 ms
    watch()     1 ms
```

```
C:\Users\Jayanth\.jdks\openjdk-17.0.2\bin\java.exe ...

Process finished with exit code 0
```

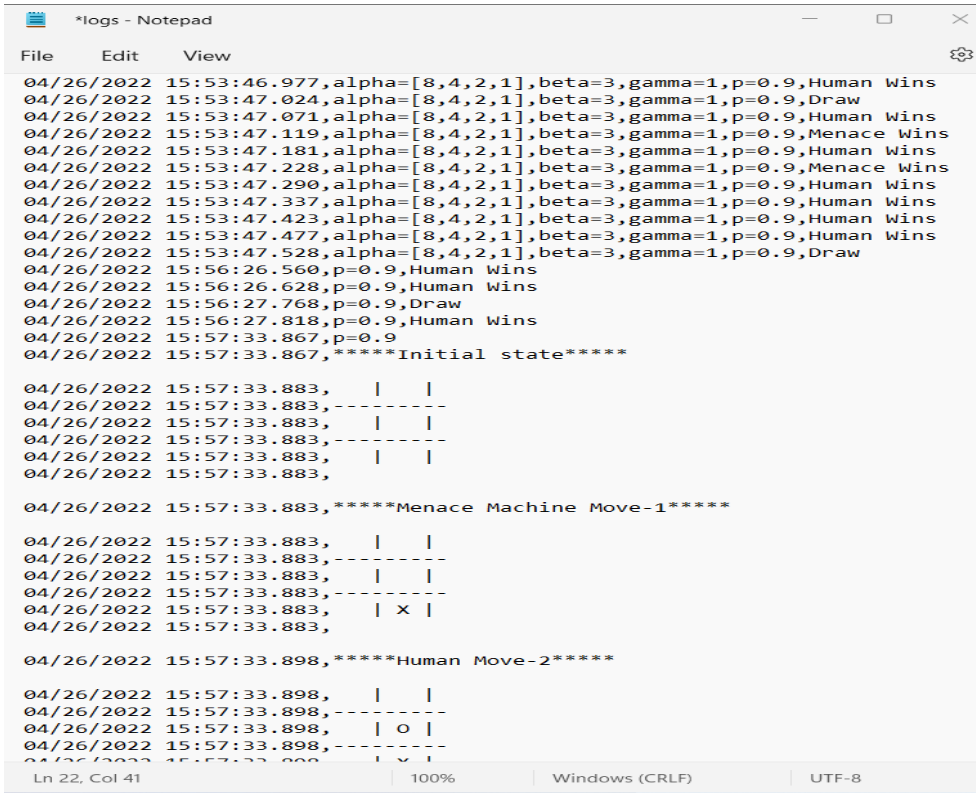
HumanTest.java



```java
import ...

class HumanTest {

    @Test
    void finalMove() {
        int move = Human.finalMove( x: 1, y: 2, p: 0.5);
        boolean flag = (move == 1) || (move == 2);
        assertTrue(flag);

    }

    @Test
    void humanMove() {
        Human h = new Human();
        int move = h.humanMove( state: "ONONXNXXN", p: 1);
        assertEquals( expected: 1,move);
    }

    @Test
    void humanStrategy() {
        Human h = new Human();
        String[][] a = {{"O","N","O"},{"N","X","N"},{"X","X","N"}};
```

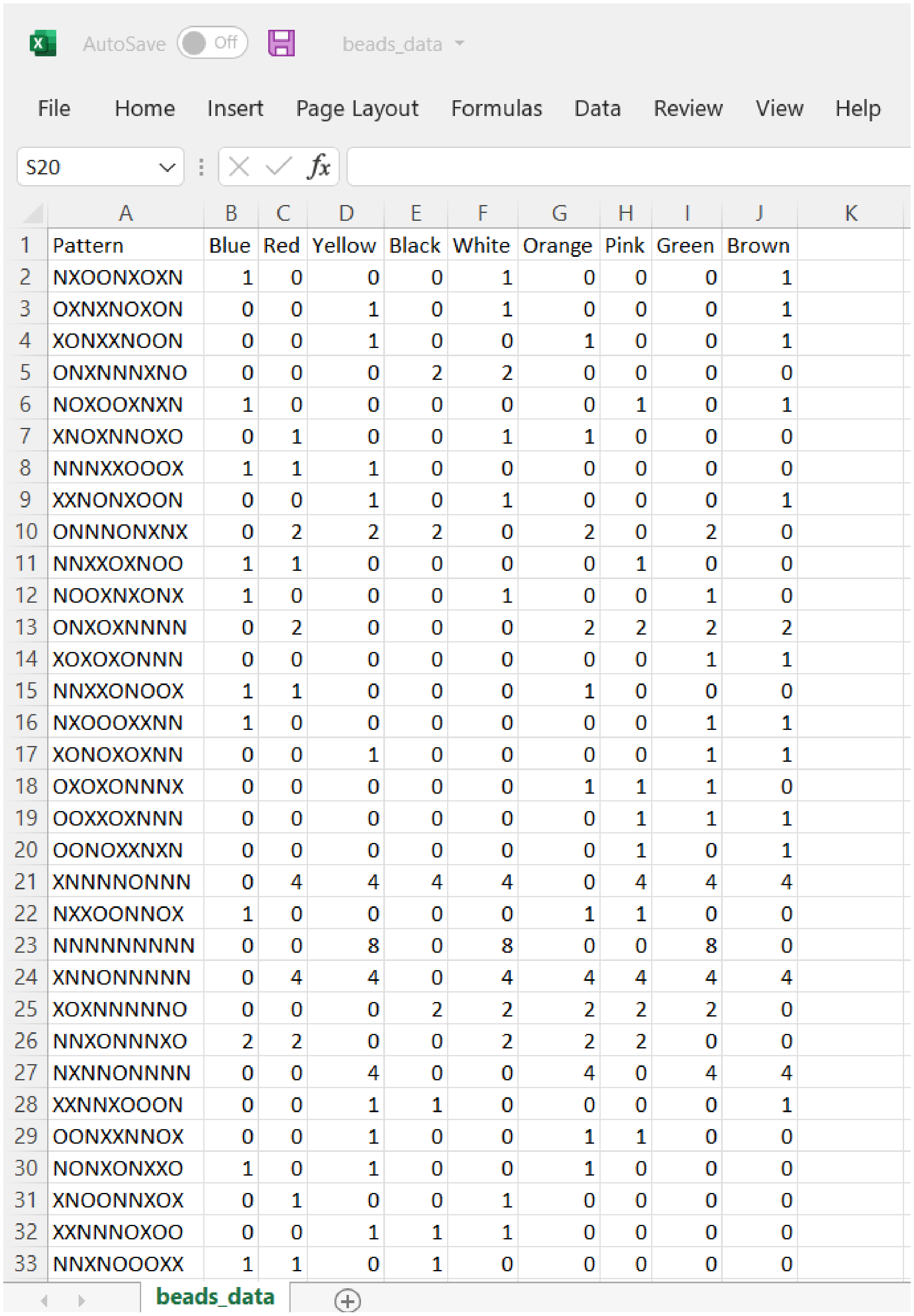
```
Run: HumanTest
Tests passed: 3 of 3 tests – 31 ms
Test Results          31 ms
  HumanTest           31 ms
    humanStrategy()   26 ms
    humanMove()        4 ms
    finalMove()        1 ms

C:\Users\Jayanth\.jdks\openjdk-17.0.2\bin\java.exe ...

Process finished with exit code 0
```

10. Sample logs



```
04/26/2022 15:53:46.977,alpha=[8,4,2,1],beta=3,gamma=1,p=0.9,Human Wins
04/26/2022 15:53:47.024,alpha=[8,4,2,1],beta=3,gamma=1,p=0.9,Draw
04/26/2022 15:53:47.071,alpha=[8,4,2,1],beta=3,gamma=1,p=0.9,Human Wins
04/26/2022 15:53:47.119,alpha=[8,4,2,1],beta=3,gamma=1,p=0.9,Menace Wins
04/26/2022 15:53:47.181,alpha=[8,4,2,1],beta=3,gamma=1,p=0.9,Human Wins
04/26/2022 15:53:47.228,alpha=[8,4,2,1],beta=3,gamma=1,p=0.9,Menace Wins
04/26/2022 15:53:47.290,alpha=[8,4,2,1],beta=3,gamma=1,p=0.9,Human Wins
04/26/2022 15:53:47.337,alpha=[8,4,2,1],beta=3,gamma=1,p=0.9,Human Wins
04/26/2022 15:53:47.423,alpha=[8,4,2,1],beta=3,gamma=1,p=0.9,Human Wins
04/26/2022 15:53:47.477,alpha=[8,4,2,1],beta=3,gamma=1,p=0.9,Human Wins
04/26/2022 15:53:47.528,alpha=[8,4,2,1],beta=3,gamma=1,p=0.9,Draw
04/26/2022 15:56:26.560,p=0.9,Human Wins
04/26/2022 15:56:26.628,p=0.9,Human Wins
04/26/2022 15:56:27.768,p=0.9,Draw
04/26/2022 15:56:27.818,p=0.9,Human Wins
04/26/2022 15:57:33.867,p=0.9
04/26/2022 15:57:33.867,*****Initial state*****

04/26/2022 15:57:33.883,   |   |
04/26/2022 15:57:33.883,---------
04/26/2022 15:57:33.883,   |   |
04/26/2022 15:57:33.883,---------
04/26/2022 15:57:33.883,   |   |
04/26/2022 15:57:33.883,

04/26/2022 15:57:33.883,*****Menace Machine Move-1*****

04/26/2022 15:57:33.883,   |   |
04/26/2022 15:57:33.883,---------
04/26/2022 15:57:33.883,   |   |
04/26/2022 15:57:33.883,---------
04/26/2022 15:57:33.883,   | X |
04/26/2022 15:57:33.883,

04/26/2022 15:57:33.898,*****Human Move-2*****

04/26/2022 15:57:33.898,   |   |
04/26/2022 15:57:33.898,---------
04/26/2022 15:57:33.898,   | O |
04/26/2022 15:57:33.898,---------
```

11. Sample beads file



	A	B	C	D	E	F	G	H	I	J	K
1	Pattern	Blue	Red	Yellow	Black	White	Orange	Pink	Green	Brown	
2	NXOONXOXN	1	0	0	0	1	0	0	0	1	
3	OXNXNOXON	0	0	1	0	1	0	0	0	1	
4	XONXXNOON	0	0	1	0	0	1	0	0	1	
5	ONXNNNXNO	0	0	0	2	2	0	0	0	0	
6	NOXOOXNXN	1	0	0	0	0	0	1	0	1	
7	XNOXNNOXO	0	1	0	0	1	1	0	0	0	
8	NNNXXOOOX	1	1	1	0	0	0	0	0	0	
9	XXNONXOON	0	0	1	0	1	0	0	0	1	
10	ONNNONXNX	0	2	2	2	0	2	0	2	0	
11	NNXXOXNOO	1	1	0	0	0	0	1	0	0	
12	NOOXNXONX	1	0	0	0	1	0	0	1	0	
13	ONXOXNNNN	0	2	0	0	0	2	2	2	2	
14	XOXOXONNN	0	0	0	0	0	0	0	1	1	
15	NNXXONOOX	1	1	0	0	0	1	0	0	0	
16	NXOOOXXNN	1	0	0	0	0	0	0	1	1	
17	XONOXOXNN	0	0	1	0	0	0	0	1	1	
18	OXOXONNNX	0	0	0	0	0	1	1	1	0	
19	OOXXOXNNN	0	0	0	0	0	0	1	1	1	
20	OONOXXNXN	0	0	0	0	0	0	1	0	1	
21	XNNNNONNN	0	4	4	4	4	0	4	4	4	
22	NXXOONNOX	1	0	0	0	0	1	1	0	0	
23	NNNNNNNNN	0	0	8	0	8	0	0	8	0	
24	XNNONNNNN	0	4	4	0	4	4	4	4	4	
25	XOXNNNNNO	0	0	0	2	2	2	2	2	0	
26	NNXONNNXO	2	2	0	0	2	2	2	0	0	
27	NXNNONNNN	0	0	4	0	0	4	0	4	4	
28	XXNNXOOON	0	0	1	1	0	0	0	0	1	
29	OONXXNNOX	0	0	1	0	0	1	1	0	0	
30	NONXONXXO	1	0	1	0	0	1	0	0	0	
31	XNOONNXOX	0	1	0	0	1	0	0	0	0	
32	XXNNNOXOO	0	0	1	1	1	0	0	0	0	
33	NNXNOOOXX	1	1	0	1	0	0	0	0	0	

12. References

- Problem Statement
- Logic for Optimal Strategy
- Logic for CSV I/O operations
- sl4j logging framework
- Reinforcement learning elements