**INFO 6205 Spring 2022 Project**

**Menace**

Members:

Ishan Khodke

Vishnu Raj Rajagopal

* **Introduction**

**Aim**

The Menace is the AI created using “data science” and reinforcement training. The aim of the project is to come up with an optimum human logic and then simulate the training of Menace by pitting it against the “human”.

**Approach**

The Menace is trained by playing lots of matches against the “human”. The Menace is then awarded points if it wins or draws. It is awarded more points if the result is a win. If the Menace loses, points are deducted for making the moves that lead to the loss. The number of training games need to be enough so that most conditions of the Tic Tac Toe game are covered.

* **Program**

**Data Structures & classes**

**TTT\_Main Class** has the main function.

**Tree** class creates a tree of all possibilities and the best move for any situation given by the tree. Thus, the Tree class births another set of trees that depict all the possible scenarios after the current scenario.

**Menace** class is used to train the Menace. It utilizes a List (basically acting as a stack) that keeps storing the current game’s moves. Menace also uses a Hashmap that has the key as a string that depicts the current state of the game and the value is basically a collection of “beads” or “points”, from which one is randomly selected to choose a move.

**BeadsDistribution** class describes the distribution of various beads for the situation described by a string.

**Players** and **SitAndChoice** classes are used for the ease of the program to keep track of the players and the choices respectively. These are what one would call Data classes.

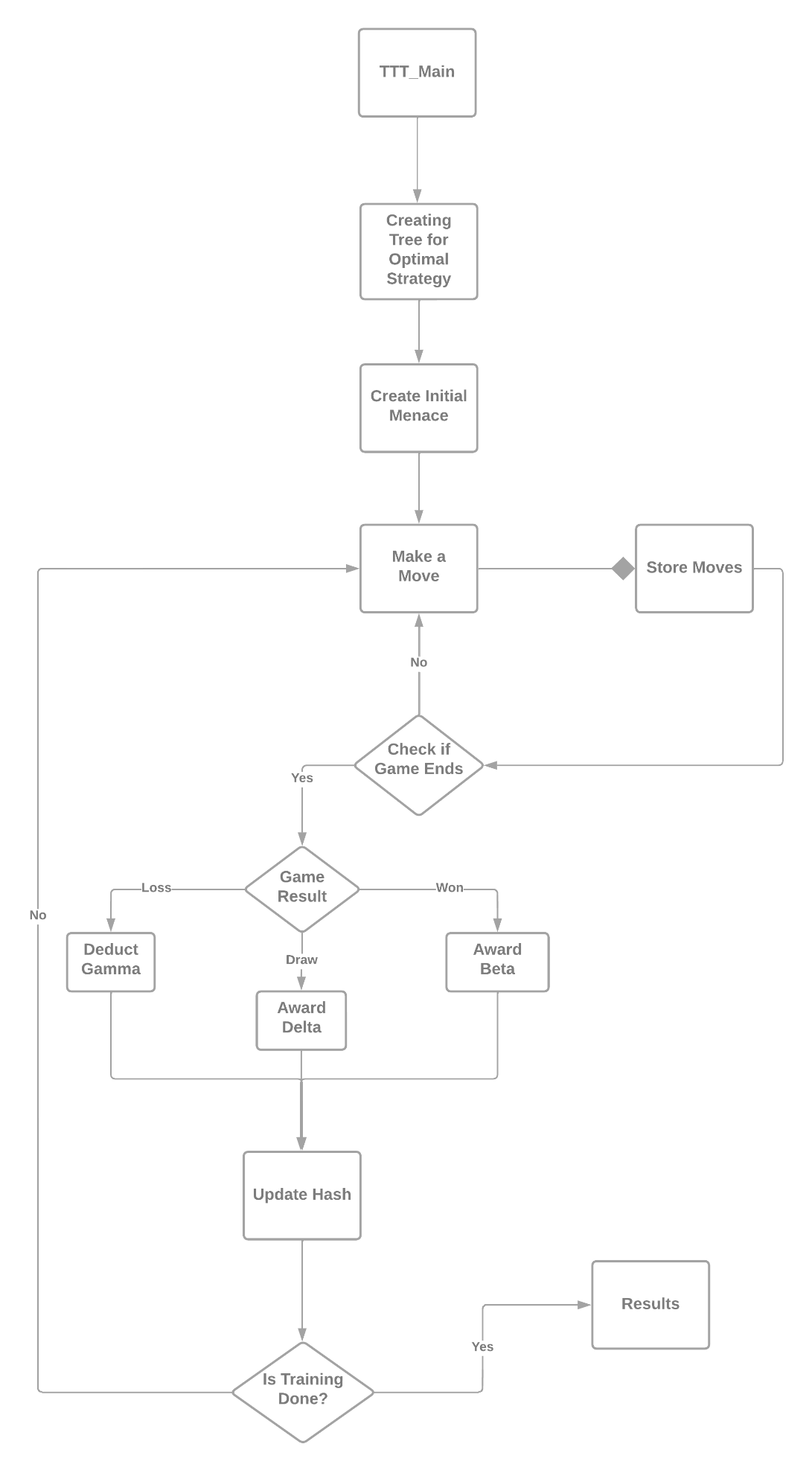
**Algorithm**

Tree class creates a tree of all possibilities and the best move for any situation given by the tree. When creating a tree, each object of the Tree class spawns more Tree class objects that are equal to the number of the possible moves that can be made in the field. These new Tree class objects in turn spawn more Tree objects whose quantity is equal to the number of possible marks in the field. When a particular scenario is “won”, the winner is given a winning score for that mark. Similarly it is given a losing score or a drawing score after it loses or draws. The minimax algorithm is used for basically, bringing the score up to the starting point. The **minimax** algorithm says that the optimum strategy is for one of the players to get the maximum score while the other player gets the minimum score. As such, **the minimizing score** for marking any of the spots on the field is equal to the minimum of all the scores from all the trees that spawn after playing that spot. Similarly, **the maximizing score** for playing any of the available spots on the field is the numerical maximum of all the scores of all the trees that spawn after marking that particular spot. This Tree(root) thus created will act as a “human” playing the best moves. To introduce the “in the zone” effect, a probability between 0.7 to 0.9 is chosen to force the Tree to make a wrong move.

For training the menace the “human simulator” goes toe to toe with the menace. Menace utilizes the BeadsDistribution class to choose the move. **If the number of total beads in all the nine available moves is “sum”, then a random number is generated between zero to “sum”. Depending on this random value, an index of the beads is selected, which basically means we have selected a move based on *weighted probability*.**

The Menace uses **Hashmap to store and update what move to make under the given “situation”**. The situation, which is the key of the Hashmap, is a string representation of the playfield. The value of the HashMap is the distribution of beads. These beads and their distribution will change each time the menace wins, loses or draws. To do so, the Menace makes use of a **List like a Stack**. Each time the menace makes a move, these moves are stored in the list. Once the game ends, this List is traversed and each of the moves made by the Menace are scored based on the result of the match. This list is emptied along the way.

* **Flow Charts (inc. UI Flow)**



* **Observations & Graphical Analysis**

The values of alpha, beta, gamma, delta and the probability of human to make a right move are varied and the variation in training the Menace is studied. Refer to the attached excel file for the tables for analysis.

* **Results & Mathematical Analysis**

The values of alpha, beta, gamma, delta and the probability of human to make the right move are extremely important to train the Menace. These values impact in the following ways:

**Alpha**, that is the number of beads in the start in each matchboxes, is important, because if during the training unluckily bad moves are made in the next steps, the number of beads will reduce to zero. This essentially means that the Menace can not possibly make that move again, even if some good moves that come after this move can potentially win the game for menace. On the other hand, increasing the alpha value will lead to larger distribution of the beads, and more training rounds would need to be taken for training the Menace to make an easily distinguishable move.

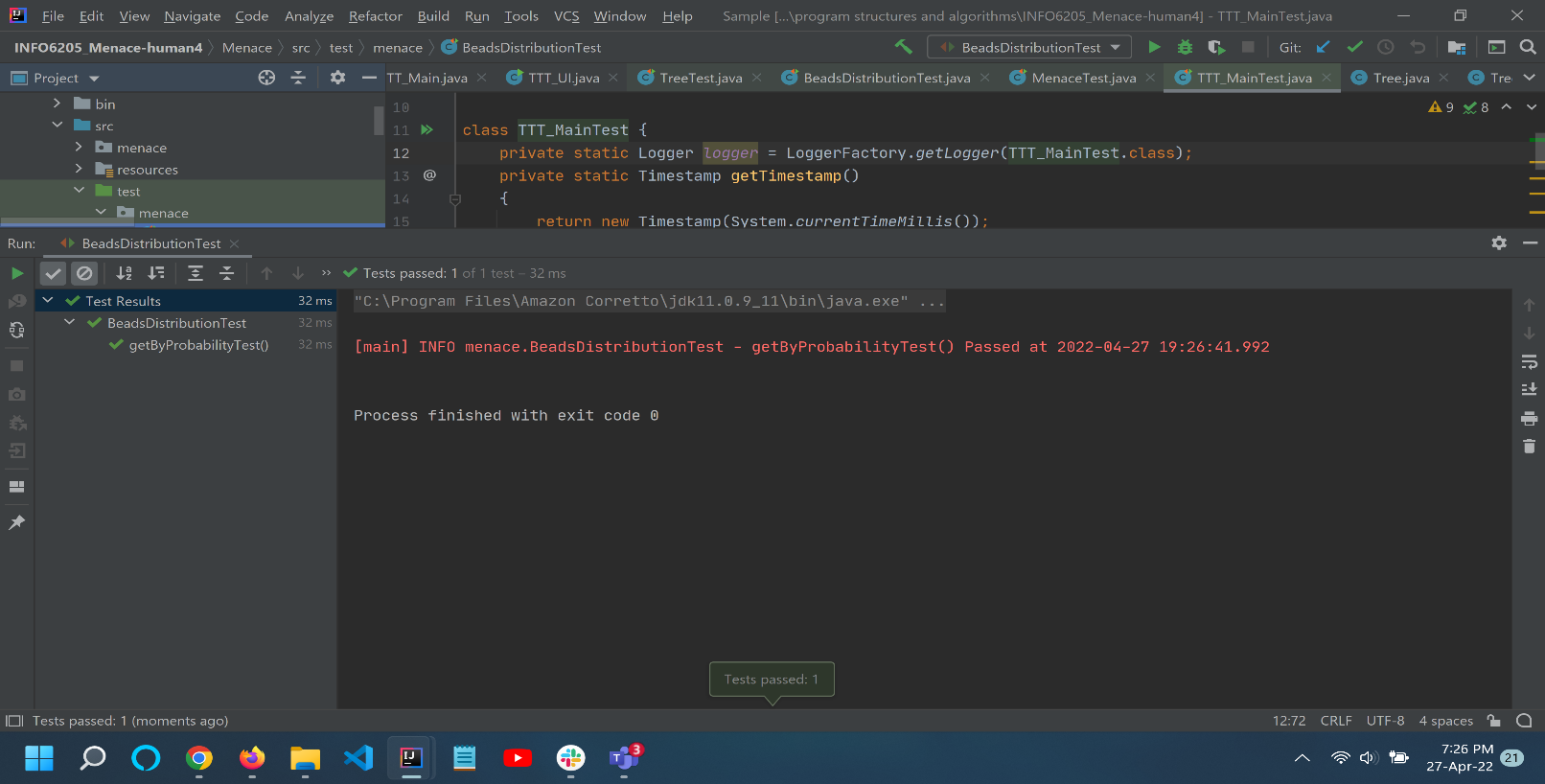
**Beta**, the number of beads awarded on win is important, because adding the beads for a move increases the chance of Menace to make the same move again. This “encouraging” value should be large enough to make a difference in the beads distribution but not too large, or it will mess up the beads distribution to a point that, the Menace can not make any other possibly good moves.

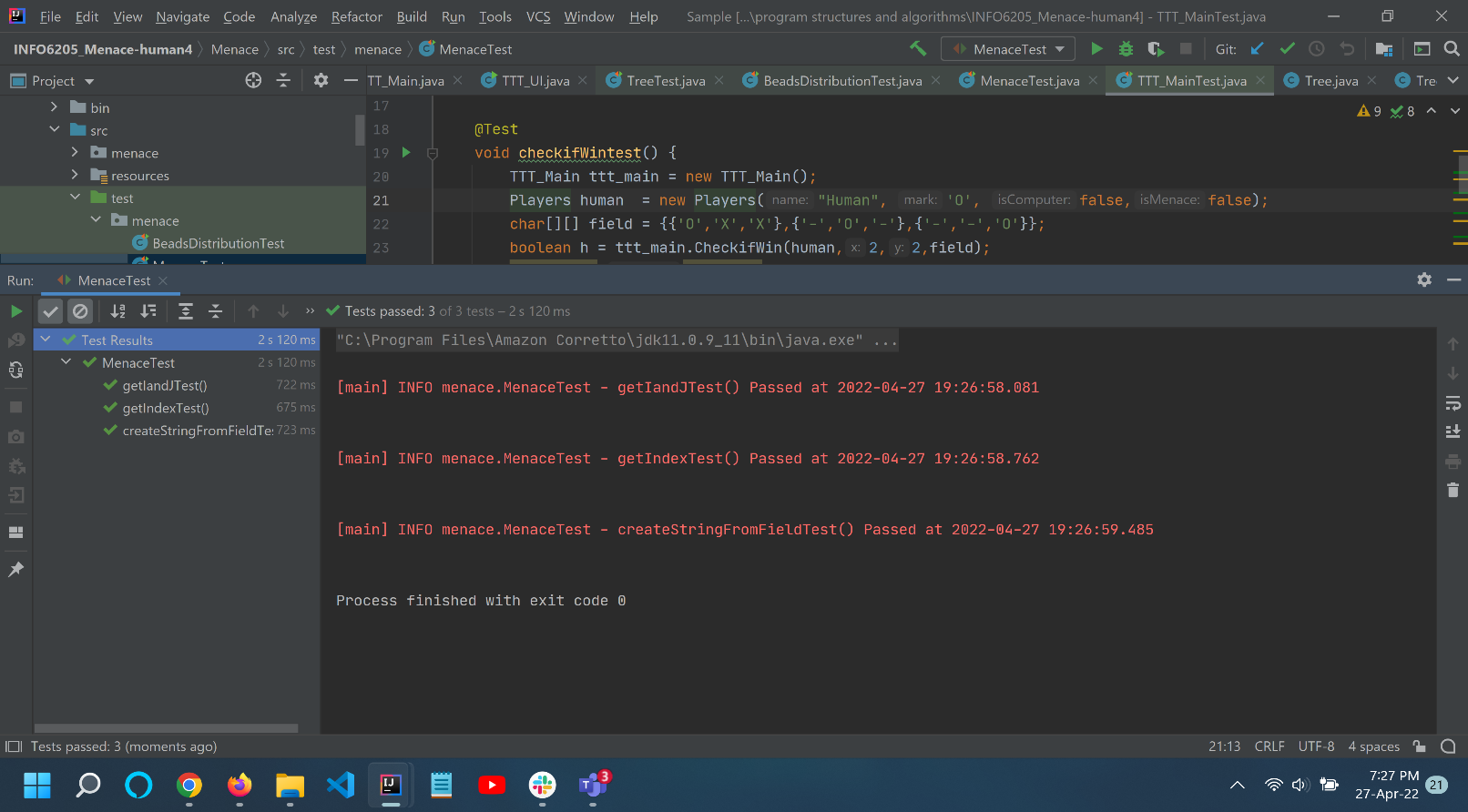
**Gamma**, the number of beads deducted after a loss is important, because it discourages the Menace from making the same mistake again. This value should be, of course, negative and be enough to make a difference in the beads distribution. This value should not be so large in magnitude that may hamper the potential of that move to eventually win games for Menace.

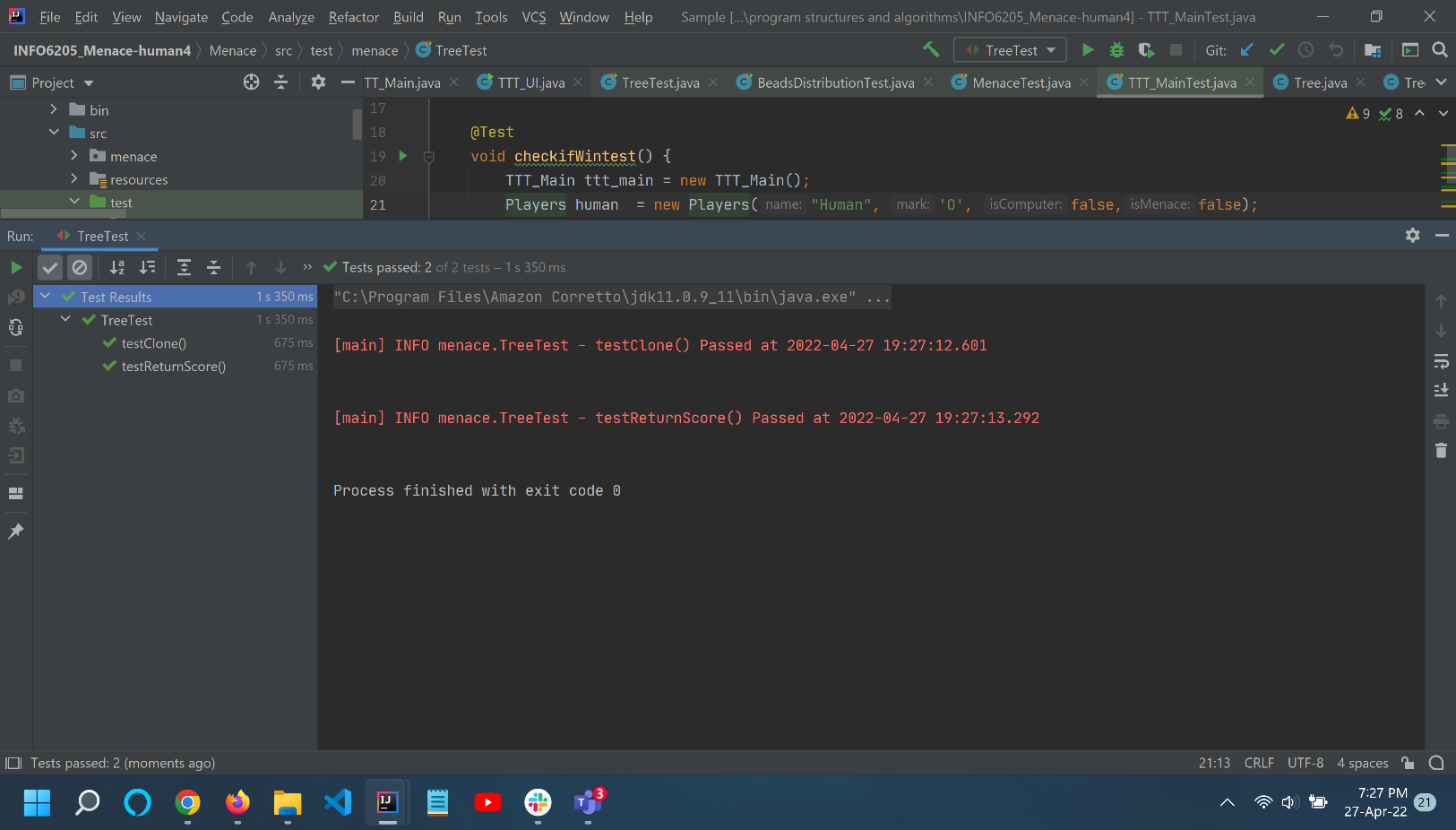
**Delta**, the number of beads added after a draw is important because a drawn match against a near perfect opponent is also considered a good match for the Menace. The Menace is awarded points, but not as much as when it would have won.

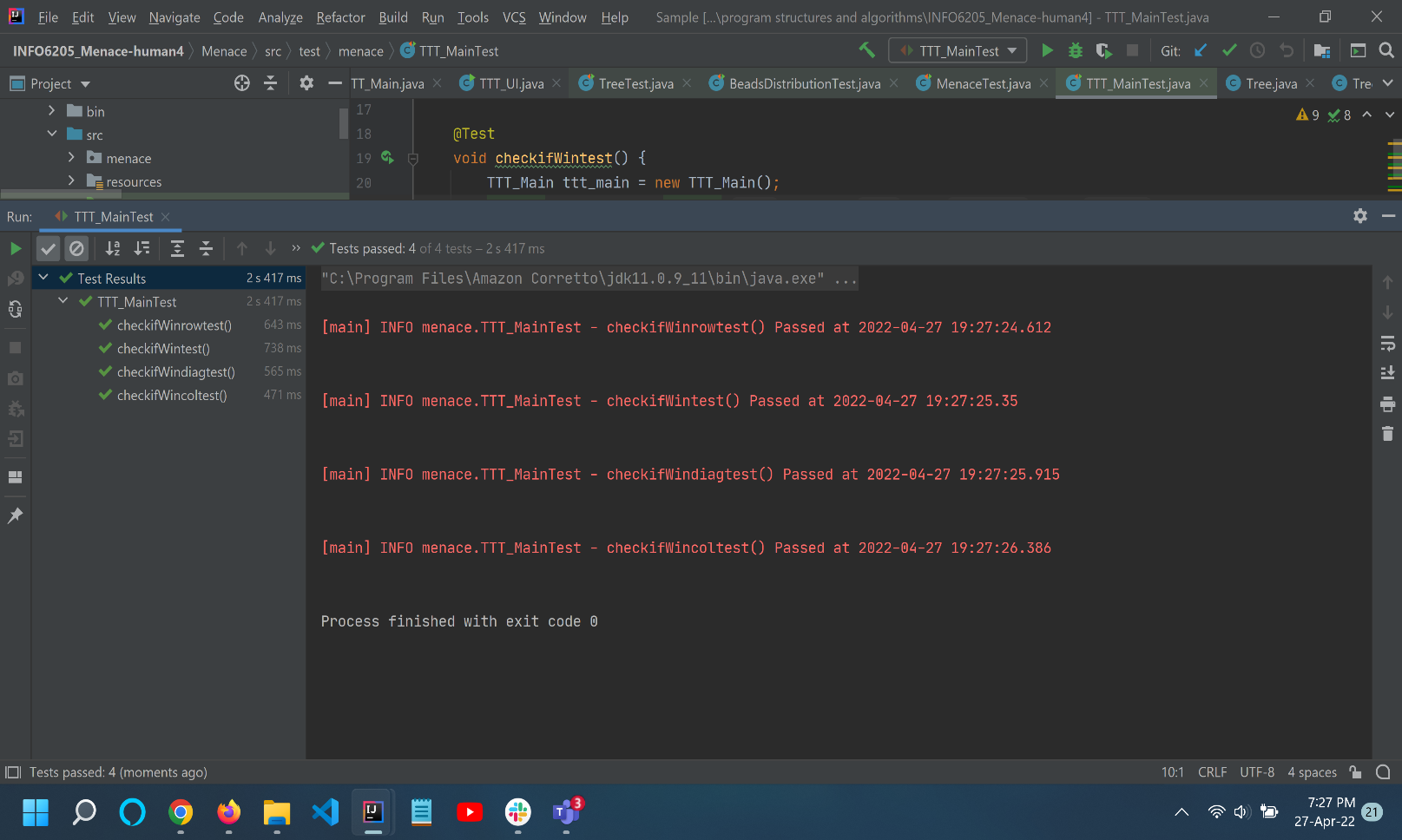
The **probability** of the human to make right/wrong decisions is important. This is because, making a random move may create new values in the Hashmap that did not previously exist. But training against a “bad” human would lead to that much wrong beads distribution. If the probability is p and the human plays a total of 4 moves, the probability is now p^4 for the human to play a “perfect” game. Thus, 0.9 leads to 0.9x0.9x0.9x0.9 = 0.66.

* Testcases









Test Cases run successfully.

* Conclusion

The Menace is trained in a very delicate manner. A “bad” beginning to the training session may lead to the entire bad training of the menace as the distribution of beads may not ever recover. But with better modelling, Menace can be trained using reinforcement training, which would **probably** make the right move to any move played by the opponent. For the training of menace, the perfect balance of alpha, beta, gamma, delta values are required.

* References

<https://www.neverstopbuilding.com/blog/minimax>

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

[Christmas Lectures 2019: How to Bend the Rules - Hannah Fry](https://www.youtube.com/watch?v=TtisQ9yZ2zo&t=2210s)