**INFO 6205 Spring 2022 Project**

***Menace***

* Introduction
  + Aim

Implement a training to get player strategy for winning the tic-tac-toe. Set up a server to using training result and human strategy to offer a challenge for player.

* + Approach

Firstly, train 2 agents to play against each other and save their policy.

Secondly, load the policy and make the agent to play against human.

* Program
  + Data Structures & classes

**Data Structure:**

1. **Training class**

* List<Integer> **state**: size of 9 demonstrates 9 grids in a board. The elements are either 0(Blank),1(X) or 2(O).
* Hashtable < List<Integer >, List<Integer>> **steps** : key is the current state of a board, value is the next move state.
* Hashtable<List<Integer>, Hashtable< List<Integer>, Double> **menaces** : key is the current state, value is all the possible moves corresponding to the current move.

1. **Board class**

* **State** [BOARD\_WIDTH] [BOARD\_WIDTH]: display the chess board. Record every move of player.
* HashSet<> () **movesAvailable:** offer position of board that haven’t been occupied.
* Hashtable() **movesOccupied:** offer position of board that haven’t been put.

**Classes:**

**1、Training**: implement the training process.

Each time after finishing a game, all the steps taken by X and O are recorded in the variant steps.

**Method**:

* pretraining() to initial the “matchboxes” pool,
* greedyAction() to optimized the training and
* BestMoveFromTraining() to offer the best move from training result.

**2、Board:** Represents the Tic Tac Toe board.

**Method**:

* Intial() and reset() to set the board.
* Move() to represent the change and check who win of this game.

1. **Server class**

图示

描述已自动生成

**Fig1.** High level design of our game

Each player connects to our server on a separate thread. So each Tic Tac Toe game running on our server will have a set of 2 threads, one for each player. Since at any instance, our server can have multiple threads, we can say our server is multithreaded.

图形用户界面, 文本, 应用程序

描述已自动生成

**Fig2.** Mark reminds to player

These threads enable the clients to play the game independently. The first client to connect to the server is player X and the second is player O, also we have a sign that reminds the player of the symbol represented, like Fig2. Player X makes the first move.

When the TicTacToeServer application executes, the main method creates a TicTacToeServer object called server. The constructor attempts to set up a ServerSocket. If successful, the program will print "Tic Tac Toe Server is Running".

图形用户界面, 文本

描述已自动生成

**Fig3.** Server Running

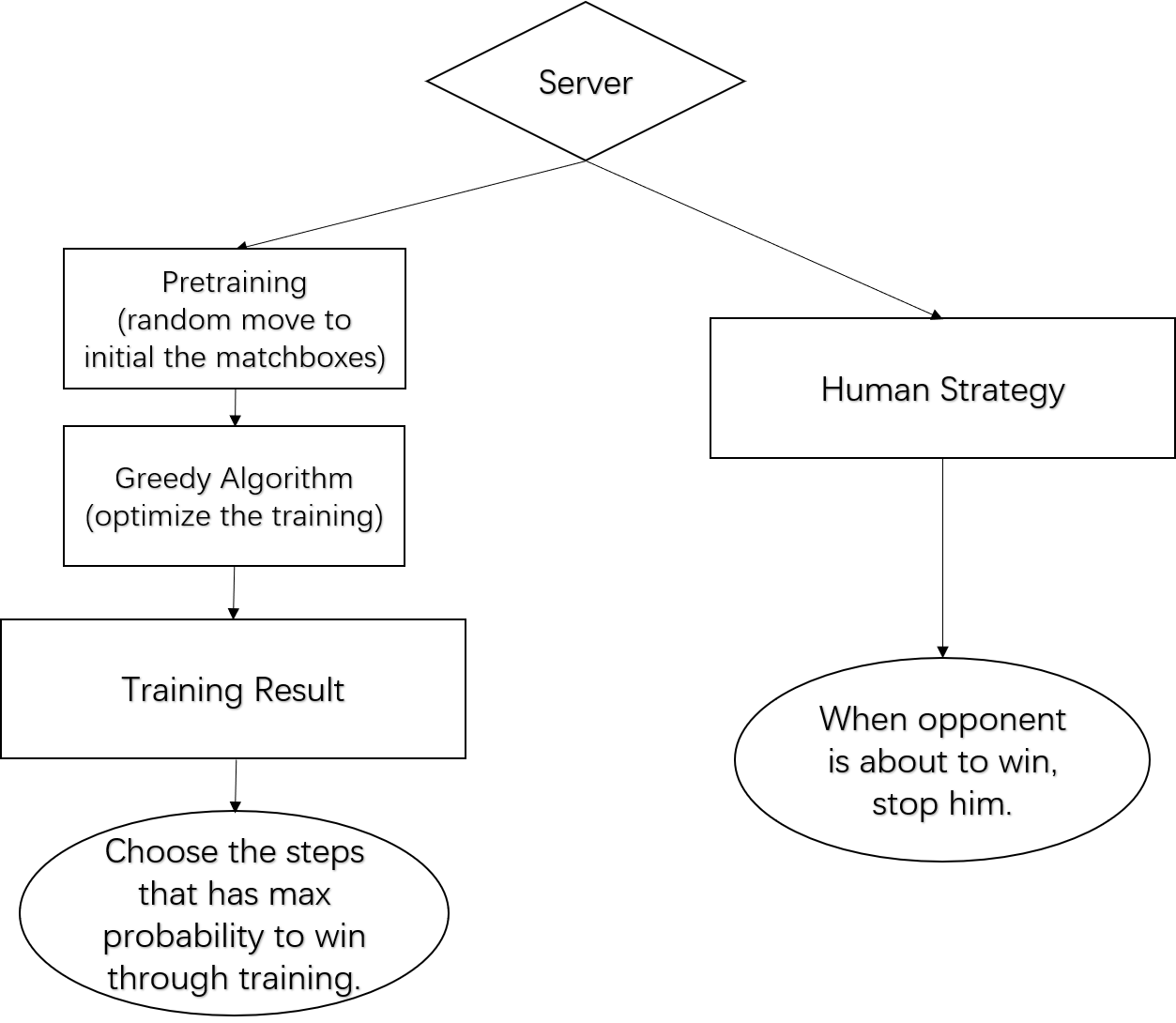
And when TicTacToeClient running, Server receives the Socket object representing t he connection to the client and gets the associated input and output streams. The Server’s run method controls the information that is sent to and received from the each TicTacToeClient. Each message is passed to the processMessage method or processing.

文本

描述已自动生成

**Fig4.** Process message method

* Algorithm



**Fig 5.** Process of server’s algorithm

During training, the process for each player is:

* Look for available positions
* Use one strategy to choose move position
* Update board state and add the action to player’s states
* Judge if reach the end of the game and give reward accordingly

Java Implemetion:

1. The chooseAction() use greedy method to balance between exploration and exploitation. Here we set exp\_rate = 0.3 (also known as p), so 70% of the time our agent will take greedy action, which is choosing action based on current estimation of states-value, and 30% of the time our agent will take random action. We store the hash of board state into state-value Hash table, and while exploitation, we hash the next board state and choose the action that returns the maximum value of next state.

The states of each game is returned to updateStatus() and when the agent reach the end of the game, the estimates are updated in our Hash table menaces.

2. State-Value update

To update value estimation of states, we will apply value iteration which is updated based on the formula below

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

The formula simply tells us that the updated value of state t equals the current value of state t adding the difference between the value of next state and the value of current state, which is multiplied by a learning rate α (Given the reward of intermediate state is 0).[1]

1. The getReward() checks sum of rows, columns and diagonals, and return 1 if p1 wins, -1 if p2 wins, 0 if draw. At the end of game, 1 is rewarded to winner and 0 to loser. we also consider draw is also a bad end, so we give our agent p1 0.1 reward even the game is tie (also we tried different reward to leverage the best performance)
   * Invariants

Board

* + - X is the starter
    - 3 states for each grid : 0,1,2.
    - Winner State: O/X/DRAW
* Flow Charts (inc. UI Flow)

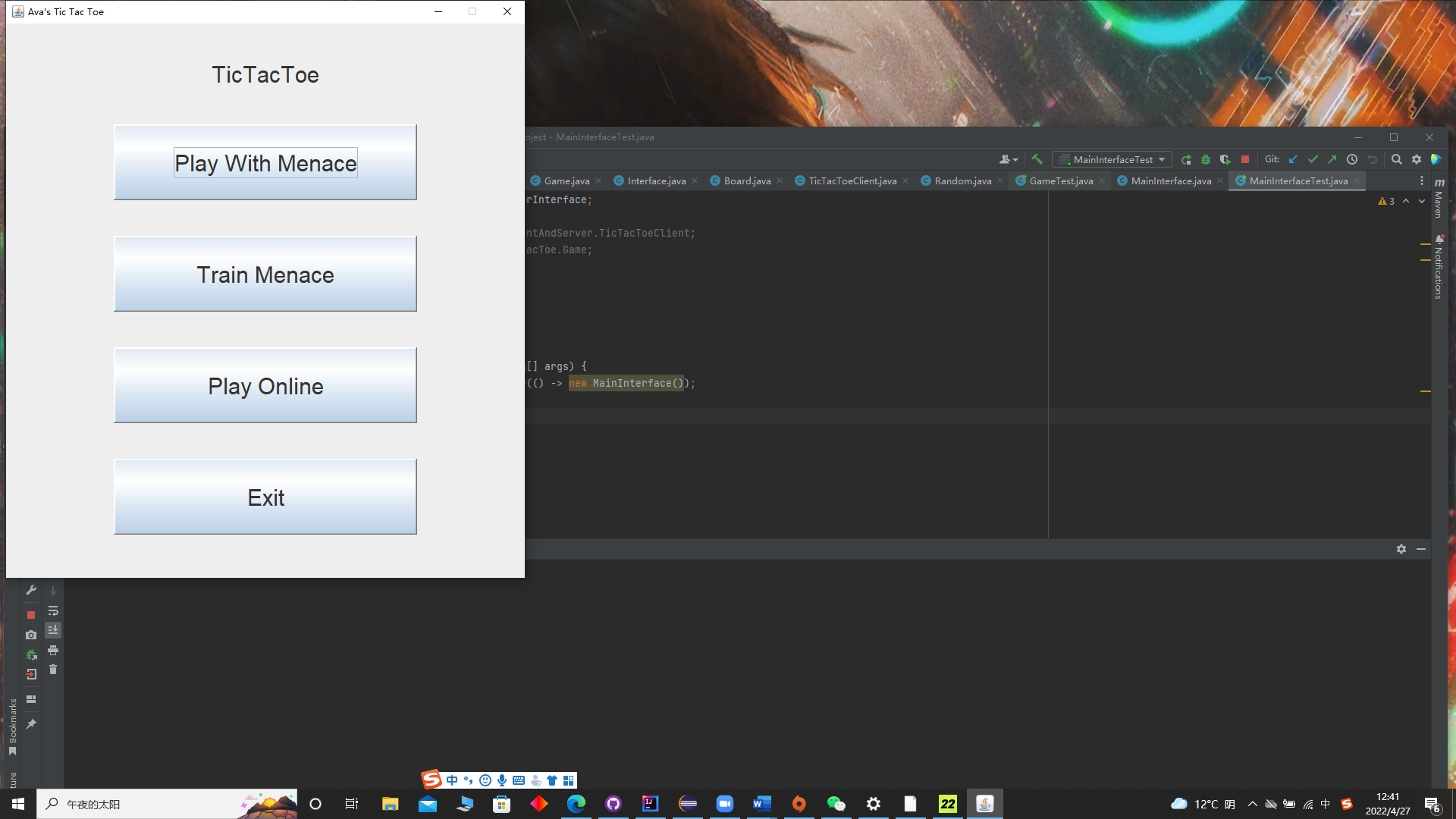
GUI

**Method:**

* **playButton()**: enter the button and then play with the menace.
* **trainButton():** enter the button and then train the menace.
* **onlineButton():** enter the button and then play with player.
* **exitButton()**: enter the button and then exit.
* **returnBtn():** enter the button and then go back to the main menu.

Our project adds two different modes to the normal TicTacToe game. The first is PVE, which is playing with the menace. The second is PVP, where you play with other players on the game server through multiple threads. This is done by clicking different buttons in the main interface of our GUI.

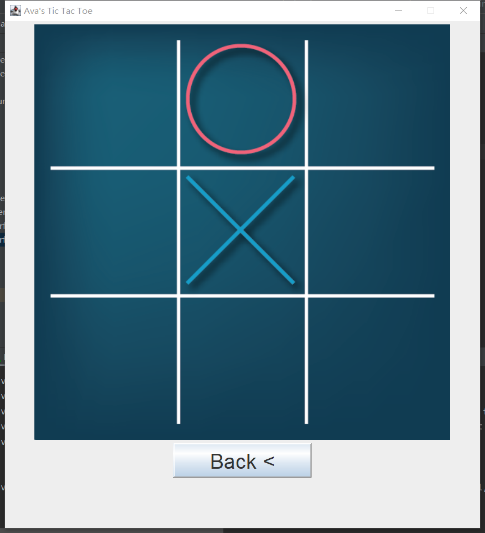
By clicking on the first Play with Menace button, we jump to an interface where the player plays the job we've trained on Menace. Clicking on the second Train Menace is tens of thousands of iterations of the algorithm on our intelligence model. By clicking the third Play Online button, we Play the game with other players on the server. Finally, click the EXIT button to exit the game.



**Fig 6.** Main interface screenshot

**Main interface screenshot:**

Regardless of whether you are in PVP or PVE mode, you will see a Back button, which will prompt you to exit the game if you click it before the end of the game.



**Fig 7.** Game interface screenshot

Each TicTacToeClient application maintains its own GUI version of the Tic-Tac-Toe board on which it displays the state of the game. The clients can place a mark only in an empty square on the board. Inner class Square implements each of the nine squares on the board. When a TicTacToeClient begins execution, it creates a JTextArea in which messages from the server and a representation of the board using nine Square objects are displayed.

形状

描述已自动生成

**Fig8.** GUI version

* Observations & Graphical Analysis

**Fig 9.** Different Result for Different Exp\_rate

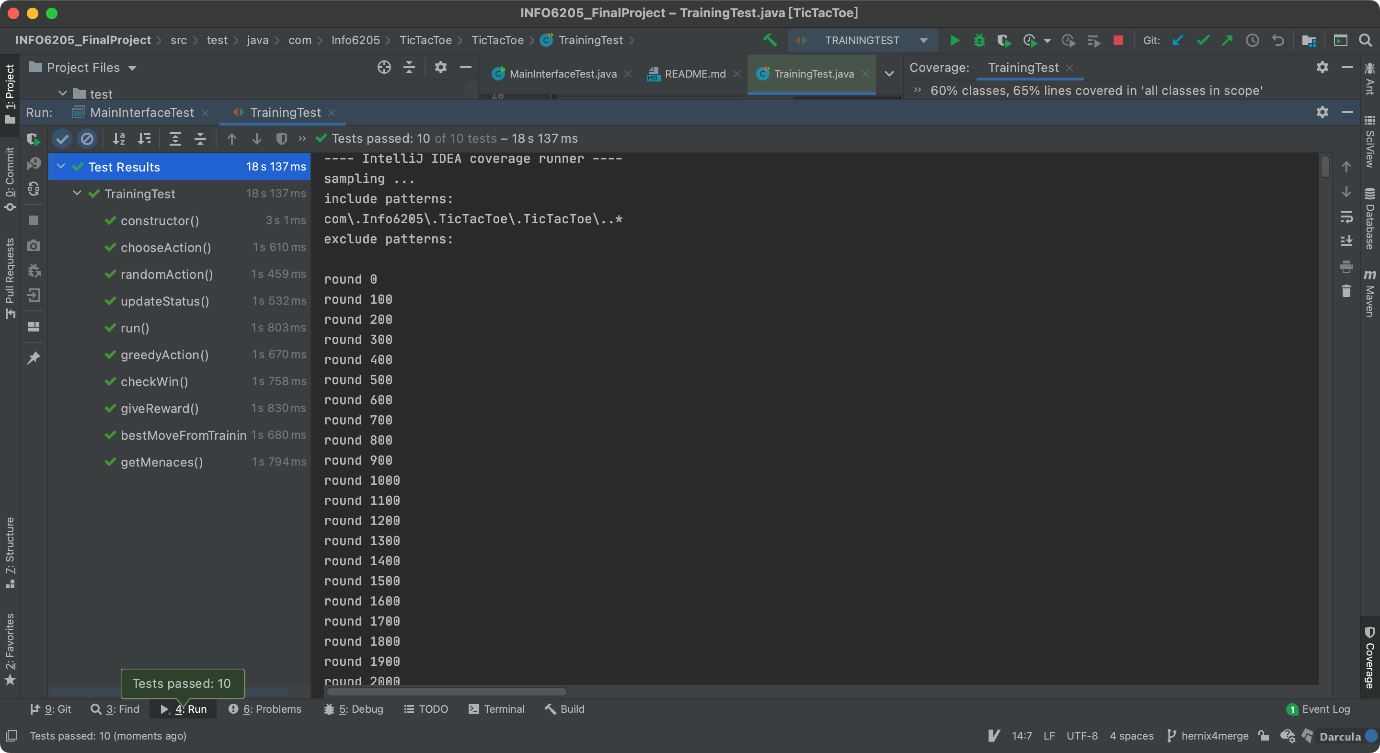
Different exp\_rate means we use different combination to set of the training, when exp\_rate is 0.3. This means we will have 30% pretraining use random move and then 60% use greedy algorithm.

* Results & Mathematical Analysis

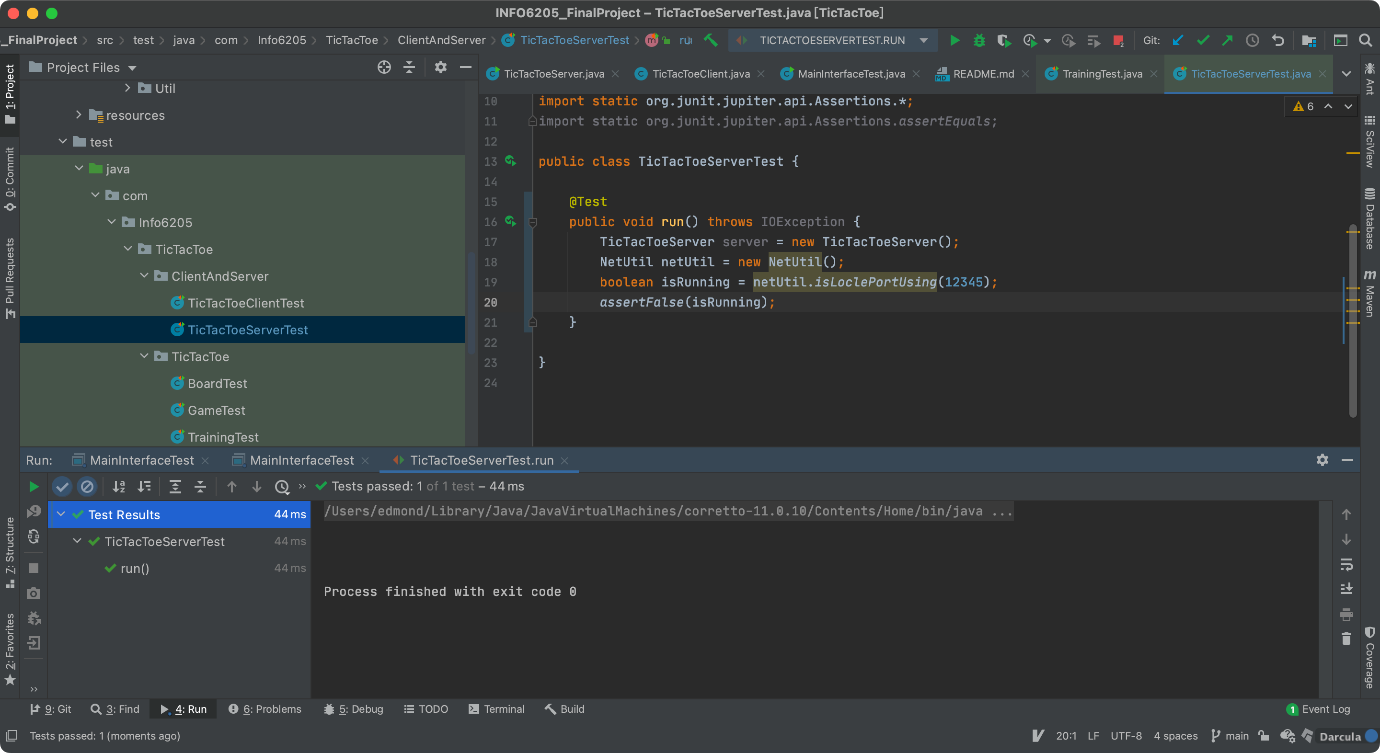
**Analysis for Fig 4.**

Different exp\_rate means we use different combination to set of the training, when exp\_rate is 0.3. This means we will have 30% pretraining use random move and then 60% use greedy algorithm. We can see in the very first beginning. Center position got best chance to win when we use the parameter. Corner got best performance when the exp\_rate is 0.4 and 0.1 for edge position.

* Testcases



**Fig 10.** test result for Training class



**Fig 11.** test result for Training class

* Conclusion

We use exp\_rate 0.3 as the training parameter, combined with human strategy and get the best situation. You can try to play with our server and have fun!

* References

1. Jeremy Zhang. “Reinforcement Learning — Implement TicTacToe” online posting. 19 May. 2019 Towards Data Science <https://towardsdatascience.com/reinforcement-learning-implement-tictactoe-189582bea542>