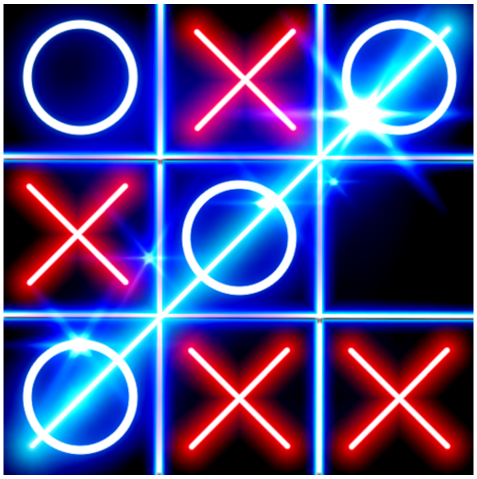
Topic 4: Suzume plays Tic Tac Toe

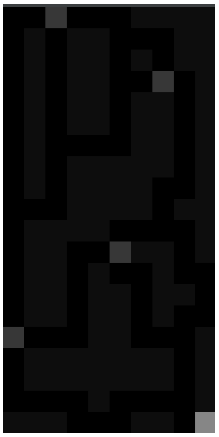


Introduction

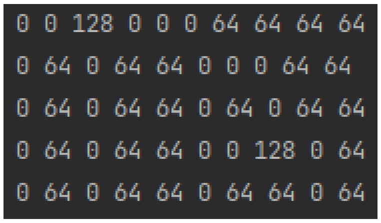
As the skies turn red and the planet trembles, Japan stands on the brink of disaster. However, a determined teenager named Suzume sets out on a mission to save her country. Able to see supernatural paths that others cannot, it's up to her to complete the missions and stop the mysterious creatures that are spreading chaos across the land. A perilous journey awaits as the fate of Japan rests on her shoulders. She has stressed out, therefore, she is reaching out to your team for help.

Problem Statement

She has found four (20x10) map-like papers with grayscale image as shown below:

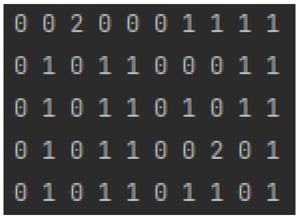
 Example of map pieces	File: Four Pieces of Map After analysing for a while, she feels like there is a hidden map after connecting four pieces of grayscale images together. She is required to decipher these images in order to know the actual map. She has no idea what to do… Fortunately, she came across a man who gave her some useful tips in deciphering this image. She jotted down a few important steps which are: 1. Read the image from file 2. Get the image dimensions 3. Get the pixel values array of the image 4. Convert them to the range of (0-3)

Example of Pixel Values Read from Image:



```
0 0 128 0 0 0 64 64 64 64
0 64 0 64 64 0 0 0 64 64
0 64 0 64 64 0 64 0 64 64
0 64 0 64 64 0 0 128 0 64
0 64 0 64 64 0 64 64 0 64
```

After converting to the range of 0-3:



```
0 0 2 0 0 0 1 1 1 1
0 1 0 1 1 0 0 0 1 1
0 1 0 1 1 0 1 0 1 1
0 1 0 1 1 0 0 2 0 1
0 1 0 1 1 0 1 1 0 1
```

Numbers	Meaning
0	Empty Spaces
1	Obstacles
2	Stations
3	Final Destination

Now, she has four 2D arrays consisting of 20 rows and 10 columns of numbers in the range of 0-3. There are a few words written beside the image and she believes these hints must be related to the **paths** that she has to embark on later.

There is a **map template** with 4 boxes written with different numbers for her to **put in the map pieces** in order to form a complete map. The **number written** is indicating the **number of possible paths for each map piece that passes through exactly 3 stations**. Hence, the correct number of possible paths for each map piece has to be found, only then a complete map can be formed.

16	41
38	27

When Suzume looks at the back of this map template, there is a sentence written:

"4 doors are created, 3 of them will eventually turn into an obstacle while **only the one at the right bottom corner leads you to heaven**"

After contemplating for seconds…

Suzume: "Hmm, it seems like I have to change 3 of the final destinations in the complete map from '3' to '1' and the right bottom corner is where I am heading to…"

These are the **rules and regulations** that she has to obey while moving around:

1. She can only **move adjacently** (left, right, top, down), strictly not diagonally.
2. She is **not allowed** to walk past the **obstacles** which are labelled as '1'.
3. She is **not allowed** to walk past the **visited 'pixel**' again in a single journey.

EMERGENCY!!!

The mysterious creatures have realised your team is assisting Suzume secretly to complete her tasks. They are irritated, hence they have kidnapped all of you to their camp site. In order for all of you to live, you have to develop 3 games as their station missions. Suzume will have to play and win in those games, meaning that you will have to create an engine that plays against her. Unfortunately, you can't make the engine bad at the game, but considering the hardships that Suzume is facing, you decided to select the easy game of Tic Tac Toe (TTT).

These are the ideas and rules & regulations for the TTT variants you have in mind:

1. Regular Tic-Tac-Toe

A regular game of TTT in a 5x5 square, players take turns placing shapes either a cross (X) or a nought (O), the winner is the first player to place 3 of their shape in either a horizontal, vertical, or diagonal row.

2. Misère/Reverse Tic-Tac-Toe

A reverse game of TTT in a 3x3 square, but in reverse, players take turns placing shapes either a cross (X) or a nought (O), the loser is the first player to place 3 of their shape in either a horizontal, vertical, or diagonal row, automatically giving the other player the win.

3. Your own variant of Tic-Tac-Toe

In this game you can choose any variant of Tic-Tac-Toe, given it is not one you made up, and proceed to create the implementation for it. Upscaled variants of the regular and reverse TTT are not viable choices, rather options like Treblecross or Wild Tic-Tac-Toe can be considered.

For each of the variants an engine is to be implemented, an engine will be the computer playing the moves against the player, the engine should not play **random moves**, but rather should play **smart moves** on the basis of a certain ruleset or calculations. The engine also has to include 3 different difficulty levels (Easy - Medium - Hard), increasing in the level of difficulty, the engine must play less optimal moves on the easy level, slightly optimal moves on the medium level, and mostly optimal moves on the hard level

Basic Requirements (12 marks)

Pixel Map (5 marks)

1. Pixel Reading (1 mark)

Since four grayscale images are given to Suzume, you have to assist her in extracting the important information from it. You should be able to read the pixel values and organise them in an array-like form as shown above. After that, you have to do some simple mathematical calculations on the pixel values and convert them to the numbers in the range of 0-3 without amending the numerical patterns among them. If you have successfully done it, **you should have 4 2D arrays with the numbers of 0-3** written in it.

2. Formation of Complete Map (1 marks)

Since you have the map pieces with you now, you have to **utilise the algorithms that you have learnt or are going to learn**, and help Suzume in determining **how many possible paths** there are for each map piece that **passes through exactly 3 stations**.

After getting the number of possible paths for each map piece that passes through exactly 3 stations, you should be able to insert the map piece into their respective position in the map template according to the numbers written on it. If you have inserted all of the map pieces correctly, **you should be able to get a (40x20) big map represented by a 2D array**. Do not forget to refer back to the hint written at the back of the map template (Attached above). Only 1 final destination should be present in the big map ultimately.

3. Search for Possible Paths (1 mark)

Since you have the (40x20) complete map with you now, you have to utilise **another algorithm** that you have learnt or are going to learn, and help Suzume in determining **how many possible paths** are there for her to reach the final destination from her current position (top left corner of the complete map) and **passes through exactly 4 stations**.

For example, if you have decided to use depth-first search to search for the possible paths of the map pieces, you have to use another algorithm such as breadth-first search to search for the possible paths of the complete map.

4. Answer Decryption (1 mark)

In order to check whether the number of possible paths of the complete map that you have found is correct or not, you have to **decrypt this weird number - 17355**. The answer is blurred, but you still saw something like **1x2x2 (?)**.

The **process of encryption** was written on a paper that you have found aside:

1. The secret key chosen is 7
2. Convert the number to binary format.
3. Divide the binary string into blocks of three digits, starting from the right. If the binary string is "1011010", the blocks would be "010" and "110".
4. For each block:
    a. Add the value of (secret key modulo 2) to the binary value of the block. (e.g., if the binary value is 101, it has to be converted to its actual value first, which is 5 and the value of (secret key modulo 2) will be added into it to make it 6.)

  b. After that, it has to be converted back to binary format in blocks of three digits. (e.g., if the value is just 2, converted to binary format would be 10, you have to pad it to make it 010.)

5. Convert each modified block back to a binary string and concatenate them to form the final encrypted binary string.
6. Convert the padded binary string back to an integer using the Integer.parseInt method with the base parameter set to 2 (indicating that the string is a binary string).
7. Return the final encrypted integer.

5. Search for Shortest Path (1 mark)

Assuming that you have finished the previous tasks easily, Suzume has acknowledged all of the possible paths. However, she needs your help to **determine the shortest path** for her to reach the final destination using the least time. You will also have to let her know **how many steps, including its direction**, are needed for that particular path. For example: “Right, Right, Down, Left” indicating 4 steps are needed with its direction provided. **If there is more than 1 path available**, please let her know all paths so that she can choose the path that she is more favourable of.

Tips: You are free to use any of the algorithms to solve this question, even the algorithms that you have used on any of the previous questions.

<u>Tic Tac Toe (6 marks)</u>

1. Tic-Tac-Toe Implementations (3 mark)

3 variants of TTT games should be implemented with all the rules, such as identifying win, losses, and ties. The 3 games should be working in a complete manner with no problems in stopping the game when needed, or when invalid moves are played. The implementation of these games should include proper console interaction to make it

easier for the user to play against the engine, that is rules and the format of the game are read to the player before starting, the board is updated on every move, the scores are shown throughout the rounds. Proper error handling also has to be implemented in the code. Players should not be able to play invalid moves on the board, or input any other invalid move formats for that matter.

Error handling to be implemented:

a. Input validation: The program should validate the user's input to ensure that it is within the acceptable range of values. For example, the user's input should be checked to make sure that it is a valid row and column number on the game board.

b. Move legality check: The program should check whether the move made by the user is a legal move, meaning that the selected cell is empty and can be played.

c. Error messages: The program should provide informative error messages in case the user enters an invalid input or makes an illegal move. This will help the user understand why their input was rejected and what they need to do to correct it.

d. Game state checks: The program should check the game state after each move to determine if the game has ended in a draw or if one of the players has won. If the game has ended, the program should inform the user and exit the game.
Note: If the GUI is chosen as an extra feature, these requirements still stand, but instead the messages and warning have to be shown in the GUI

2. Engine Implementation (1 mark)

The 3 games should have an engine that plays them (not necessarily separate). The engine should not play random moves, rather the engine should play smart moves based on some calculations. The code will be evaluated to determine if it fits the criteria. The engine should also have 3 difficulties, and the player should be able to choose from these difficulties before starting the game, the 3 levels should be Easy, Medium, and

Hard. The engine should play less optimal moves on the Easy difficulty, slightly optimal moves on the Medium difficulty, and mostly optimal moves on the Hard difficulty.

3. Save game functionality (1 mark)

Sometimes the player might need to take a break and come back later, add a functionality that allows the player to save the current game and resume it later from the same position they stopped at. The saving functionality should be able to save multiple games and allow the user to name the games being saved, and when loading a game the user should be able to choose the game by name.

4. Player Account & Leaderboard (1 mark)

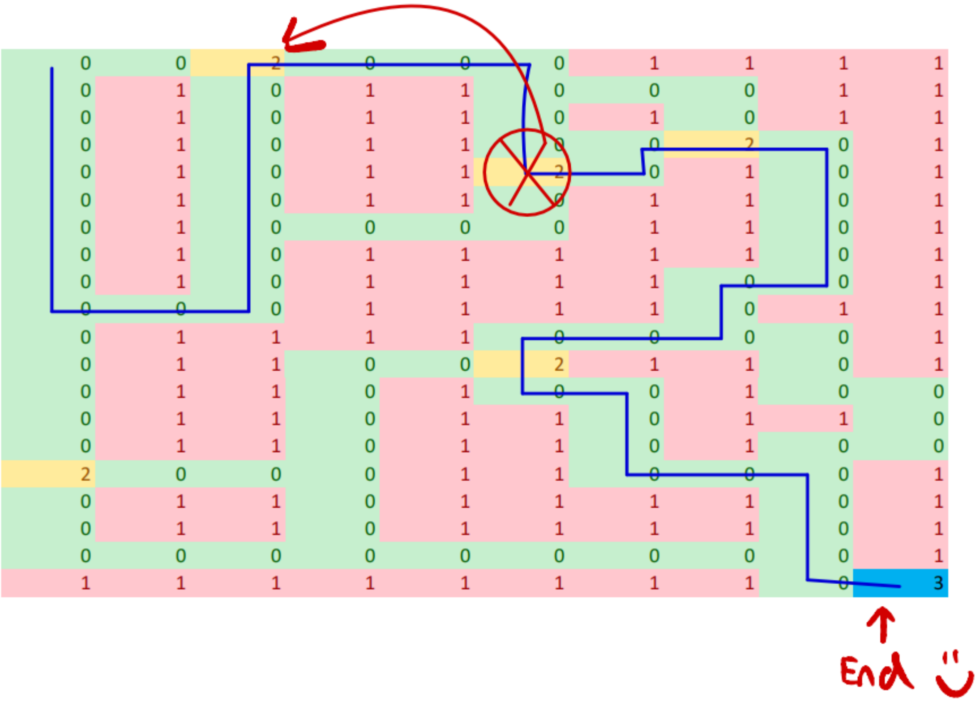
To make the game more interesting and engaging, creating a system that makes the players create a user to play with, will be a nice idea, complete with a username and a password so no one can log in to their account. Players will need to log in to their account before playing, players will be able to load saved games they played previously against the engine, other players should not be able to load others' saved games. The game should also implement a leaderboard that shows the ranking of the player based on a metric chosen by you (Win/Loss ratio, Total Wins, etc.).

Note: This feature is not to be implemented with the "Connecting the dots" feature, rather it should be separate.

Connecting the Dots (1 mark)

It's time to connect all of the dots together! Since you created the station games that she is going to face later on, it is better if you could do something to get Suzume prepared for the upcoming challenges. You came up with an idea to **create a simulation system**, integrating one of Suzume's shortest paths (if there is more than 1 shortest path) with the games that she is going to play in different stations. With this system, Suzume or any of your friends can keep practising and help Suzume in pulling through all of the upcoming tasks effortlessly.

In short, the basic requirement for this part is to at least create a **console interaction**, make Suzume stop at respective stations and play the station games. If she wins, she can proceed to the next station, else she has to fall back to the previous station. If she fails at the very first station, her journey ends. Likewise, if she succeeds in all of the stations, her journey ends, but with glory.



0	0	2	0	0	0	1	1	1	1
0	1	0	1	1	0	0	0	1	1
0	1	0	1	1	0	1	0	1	1
0	1	0	1	1	0	0	2	0	1
0	1	0	1	1	2	0	1	0	1
0	1	0	1	1	0	1	1	0	1
0	1	0	0	0	0	1	1	0	1
0	1	0	1	1	1	1	1	0	1
0	1	0	1	1	1	1	0	0	1
0	0	0	1	1	1	1	0	1	1
0	1	1	1	1	0	0	0	0	1
0	1	1	0	0	2	1	1	0	1
0	1	1	0	1	0	0	1	0	0
0	1	1	0	1	1	0	1	1	0
0	1	1	0	1	1	0	1	0	0
2	0	0	0	1	1	0	0	0	1
0	1	1	0	1	1	1	1	0	1
0	1	1	0	1	1	1	1	0	1
0	0	0	0	0	0	0	0	0	1
1	1	1	1	1	1	1	1	0	3

Assuming this is the shortest path that Suzume has chosen and whenever she reaches number ‘2’, the program has to be switched to any randomised TTT engine with all of TTT basic functionalities equipped. If she fails at the 2nd station as shown above, she has to fall back to the previous station. The game will be ended only if Suzume has failed on the first station or she has succeeded in all of the stations. She may try the games for unlimited times but all of her records should be kept.

Extra Features (4 marks)

1. Graphical User Interface (GUI)

a. Graphs

You may create an attractive map like “Bomberman’s map”, with each block symbolising each pixel and display the possible paths and the shortest path in the map with different colours. (This is just a suggestion, you may build a different map to visualise your possible paths and shortest path too)

b. Tic-Tac-Toe

A GUI that shows the game board and allows the user to play against the engine. You may refer to many existing TTT designs online but it would be more fun if you could create your own TTT game board isn’t it?

2. Engine Optimization

Playing against a slow player can oftentimes be very boring. Usually, engine implementations are slow because of the nature of implementation, and the larger the search space the slower the engine. Engines have different ways of implementation, some ways include searching for optimal moves structured in a certain data structure, others can use machine learning, or just simple rules to follow when making the moves. Different methods of optimization can be used to make the search process faster, any method of optimization is viable as long as it makes the engine play faster. A comparison of the engine with and without the optimization needs to be implemented to showcase the optimization and difference.

**Note: This feature may not be applicable to all engines*

3. Different Modes

To add more of an exciting element to the game, allow friends to challenge each other, or allow players to see the perfection of an optimal game played by 2 engines. The main mode to be implemented is player VS engine, the player plays against the engine in this mode. Player VS player (PVP) and Engine VS Engine (EVE) can be implemented as extra features. For PVP, two players can take turns playing on the same device, and for EVE the engine will play against the engine just simulating the two engines and showing their moves, for the EVE mode, the user choosing the mode should be able to choose the difficulties of each engine playing.

4. Take back move

Undoing a move is an extra functionality you can implement to make the game more interesting and easier to play, it allows the player to go back a move and play another move if they think that is better. The player should be able to do so in their turn, taking back a move will reset the board to how it was in the player's previous turn, allowing them to play again.

5. Game Analysis

Sometimes we play a game not knowing if we are doing the right or wrong thing, to learn we need to understand the mistakes we made. Add a post-game analysis option where the player can choose to replay the game and see their previous moves evaluated as poor or good moves. The implementation is not restricted to anything specific as long as moves of the player are evaluated, for example by comparing it to the most optimal move.

6. Win Probability

Throughout the game, players will not be very sure if they are winning or losing, so to make it more clear for them you can add a win probability meter. This allows the players to have a better understanding of their position. The win probability should be updated with every move, since each move will either make you closer to the win or move you further away. The win probability meter should be displayed throughout the rounds either as two percentages one for the player and the other for the engine or as a visualised bar that goes up and down to indicate the higher probability.

Tips & Comments

1. Use Git to do version control
2. Learn Pixel Reading from Image
3. Learn Graph Algorithms
4. Learn Common Encryption & Decryption Theories
5. Learn about Games and Game theories
6. Ask questions
7. Be Creative!

References & Contact Information

1. Storyline and concept
   https://en.wikipedia.org/wiki/Suzume_(film)

2. Graph Algorithms
   https://towardsdatascience.com/10-graph-algorithms-visually-explained-e57faa1336f3

3. Tic Tac Toe
   Tic-tac-toe - Wikipedia
   Tic-tac-toe variants - Wikipedia

m,n,k-game - Wikipedia

Turn-based strategy - Wikipedia

Wild tic-tac-toe - Wikipedia

Treblecross - Wikipedia

Contact Information:

Lim Jia Yu (Pixel Map & Connecting the Dots):

lim.jia.yu.desmond@gmail.com (012-4315562)

Mujahed Yahya (Tic Tac Toe):

yahyamurad197@gmail.com (011-23239542)