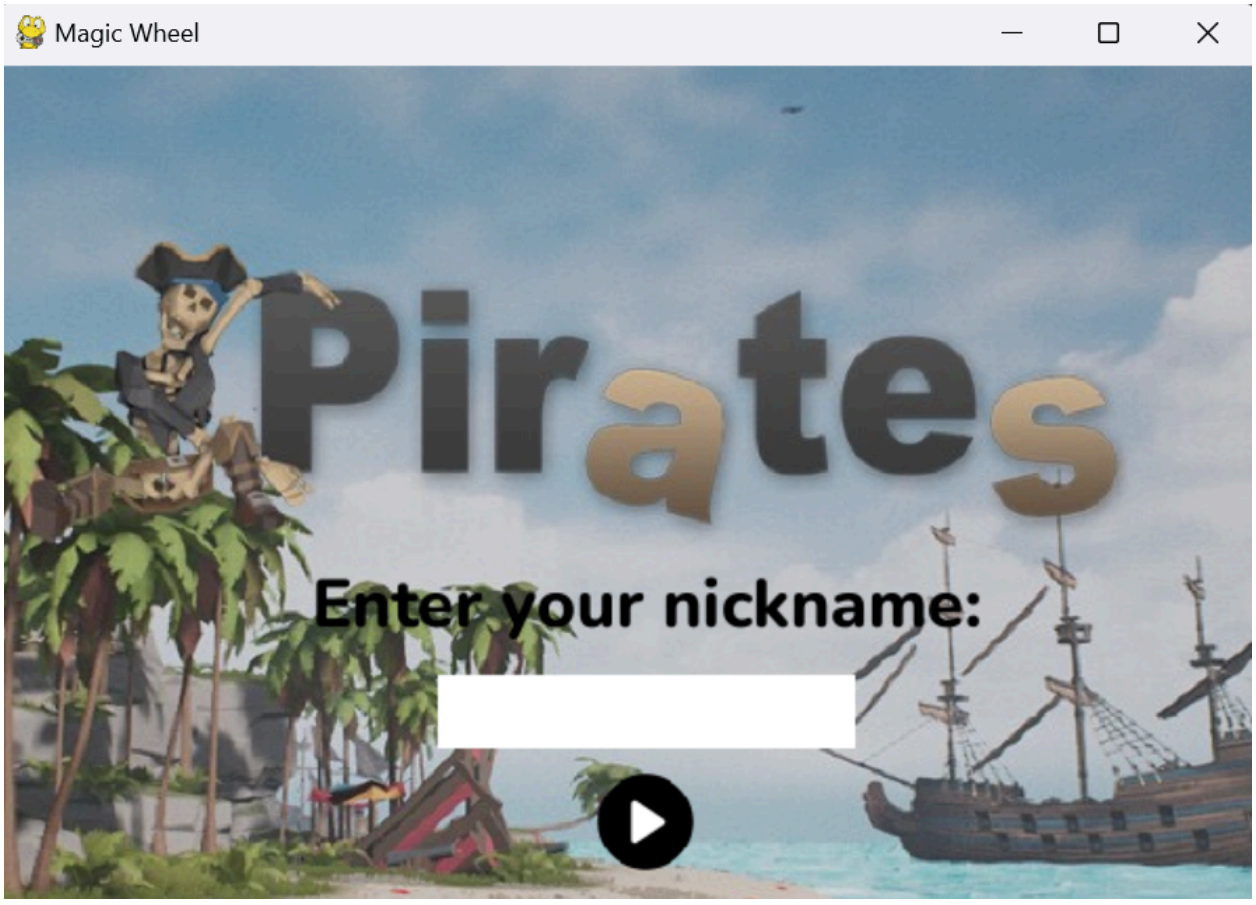
CS494: Socket Programming

GAME 00: MAGICAL WHEEL



Nguyen Truc Nhu Binh - 21125034

Nguyen Hoang Minh - 21125018

21TT1 (APCS)
FIT@HCMUS

INTRODUCTION

Game calculation: (4 + 8) mod 3 = 0 → Magical Wheel

GAME STORY

In the heart of the treacherous Caribbean lies a mysterious island guarded by the notorious Pirate King, Blackbeard. Legends speak of a chest buried deep within the island's caverns, holding unimaginable riches. But beware, for Blackbeard's cunning has safeguarded the chest with a secret code known only to him and his crew. As a daring band of pirates, you must decipher the code before your rivals. With each turn, you'll risk revealing a letter of the keyword, inching closer to unlocking the chest's bounty. But be swift and strategic, for one wrong guess could mean the end of your journey, marked by the unforgiving justice of the pirate's code.

GAME RULES

1. The server chooses a keyword in the database, then sends the length of the keyword and its description to all clients. All characters of the keyword are initially hidden and only shown when a player correctly guesses that there are such characters in the keyword.
2. The player takes turns to guess the characters that the chosen keywords have or the entire keyword. The correctness of the player answers are all announced to all players. More specifically, if they guessed a character in the keyword and the keyword indeed contained the guessed characters, all positions with such character are no longer hidden.
3. The player can only guess the keyword after two turns have taken place (these two turns do not have to be necessarily made by the currently guessing player).
4. In a turn, one of the following scenarios can take place:
    a. If the current player chooses to guess a character and the guessed character is correct, the number of points of the player is increased by one. If all characters of the keyword have been shown, the game ends. Otherwise, the current player is allowed to continue to guess.

b. If the current player chooses to guess a character and the guessed character is incorrect. The turn is moved to the player who has not been disqualified in the cycle and can be found earliest in the cycle of the player. There is no penalty for making a wrong guessed character.
c. If the current player chooses to guess the keyword and the guessed keyword is correct, the number of points of the player is increased by five and the game ends.
d. If the current player chooses to guess the keyword and the guessed keyword is incorrect, the player would be disqualified and no longer be allowed to make any guesses in future turns (the player would be skipped in the future turns). However, there is no direct penalty related to points (the number of points of the player is not decreased as a penalty for making a wrong keyword guess).

5. The game ends when one of the following conditions happens:
    a. The keyword has been correctly guessed.
    b. All players have been disqualified.
    c. No one correctly guesses the keyword after five rounds have taken place (a round is considered to take place if the turn of all players in the cycle have been considered).
6. The winner of the game is the player earning the highest number of points. If there are multiple players with the highest number of points, they are all winners.
7. When a player joins while the game has already started they will become a watcher (able to follow the game but can not answer).
8. When a player disconnects, they will die in the game and be marked as red.

CLIENT-SERVER COMMUNICATION

1. **Packet structures**

- The packets sent between clients and the server all have two main parts:

- Status code: The “status code” is used to quickly identify the type of packet, what is the purpose of the sent packet.
- Content: The “content” mainly contains the data sent between clients and servers via packets. In this program, the “content” is often a short announcement message string or a serialized json file.

2. **Packets sent between clients and the server**

- In this program, a message sent by client to server is called a “Request” and A message sent by server to client is called a “Response”.

a. Request

- There are only 4 types of requests defined in this program. Their descriptions are provided by the following table.

Type	Purpose or meaning	Usage
NICKNAME_REQUEST	- The client sends the nickname they want to choose.	- The “content” only contains a short string which is the name that the user wants to choose.
ANSWER_SUBMISSION	- The client sends the answer made by the user.	- The “content” only contains a short serialized json file. - This file contains the guessed character and the guessed keyword made by the user. It is expected that one of these two fields should have null value.
CLOSE_CONNECTION	- The client closes its connection with the server.	- The “content” only contains a short announcement string.
RESTART_NEW_MATCH	- The client who was a player of the previous game wants to join a new game.	- The “content” only contains a short announcement string.

```python
import enum

class RequestStatusCode(enum.IntEnum):
    NICKNAME_REQUEST = 1
    ANSWER_SUBMISSION = 2
    CLOSE_CONNECTION = 3
    RESTART_NEW_MATCH = 4
```

b. Response

- There are 16 types of responses defined in this program. Their details are shown in the following table.

Type	Purpose or meaning	Usage

NICKNAME_REQUIREMENT	- The server wants to receive a valid nickname chosen by the receiving client.	- The “content” only contains a short announcement string.
INVALID_NICKNAME	- The server does not accept the nickname chosen by the client because it is valid. - The client should choose and send another nickname, which should be valid.	- The “content” only contains a short announcement string.
NICKNAME_ALREADY_TAKEN	- The server does not accept the nickname chosen by the client because it has already been chosen by another player . - The client should choose and send another nickname, which should be different from nicknames registered earlier.	- The “content” only contains a short announcement string.
NICKNAME_ACCEPTED	- The server accepts the nickname chosen by the client.	- The “content” only contains a short announcement string.
BROADCASTED_MESSAGE	- The server sends a message, which is often a short announcement in the program, to every client.	- The “content” only contains an announcement string.
GAME_FULL	- There have been enough people joining the game as players. - The receiving client can only watch the game.	- The “content” only contains a short announcement string.
ANSWER_REQUIRED	- The server wants to receive an answer by the client.	- The “content” only contains a short serialized json file.
QUESTION_SENT	- The server sends a question.	- The “content” only contains a short serialized json file.
GAME_ENDED	- The game has ended.	- The “content” only contains a short announcement string.
BROADCASTED_SUMMARY	- The server sends a summary, which includes different statistics, information about game state and player, to every client.	- The “content” contains a serialized json file.
GAME_STARTED	- The game has ended.	- The “content” only contains a short serialized json file.

BROADCASTED_PLAYER_ANSWER	- The server sends an answer made by a player to every client.	- The “content” only contains a short serialized json file.
BROADCASTED_RANK	- The server sends the rank table to every client.	- The “content” contains a serialized json file.
RESTART_ALLOWED	- The server allows former players to join a new game.	- The “content” only contains a short serialized json file.
WAIT_GAME_START_REQUIRED	- The game has not started yet and the server wants the client to wait until the start of the game.	- The “content” only contains a short serialized json file.
SERVER_CLOSE_CONNECTION	- The server stops connections between it and clients. - The clients should also close connections after they have received this response.	- The “content” only contains a short serialized json file.

3. The process of sending packets between clients and the server

- After the client has established a connection with the server, if there are currently enough people joining the game as players (some of these people might have not finished registering nicknames), the server will send a response with *GAME_FULL* type. Otherwise, the server requires the client to choose a nickname by sending a response with *NICKNAME_REQUIREMENT* type.

- After the client receives the *NICKNAME_ACCEPTED* response, it will wait for the response *GAME_STARTED*.

- From the start of the registration process to when the game ends and the server waits for its administrator to decide whether a new game would be started, the server frequently sends the *BROADCASTED_SUMMARY* response to allow clients to update their user interfaces as well as ensure game logic.

- When the server wants to stop running, it will send a *SERVER_CLOSE_CONNECTION* response to every client before it stops. In the program, these responses are sent when the server administrator decides to hold no further game matches. Otherwise, if the server administrator decides to hold another game match, the *RESTART_ALLOWED* will be sent to every client who has just played in the previous match.

- Similarly, when a client wants to stop, it should send a *CLOSE_CONNECTION*

response to the server before it stops.

- When all players have registered successfully (their nicknames are valid and distinct), the server sends the *GAME_STARTED* response to every client, including both players and watchers.

4. Steps and conditions to check the packet at server and client

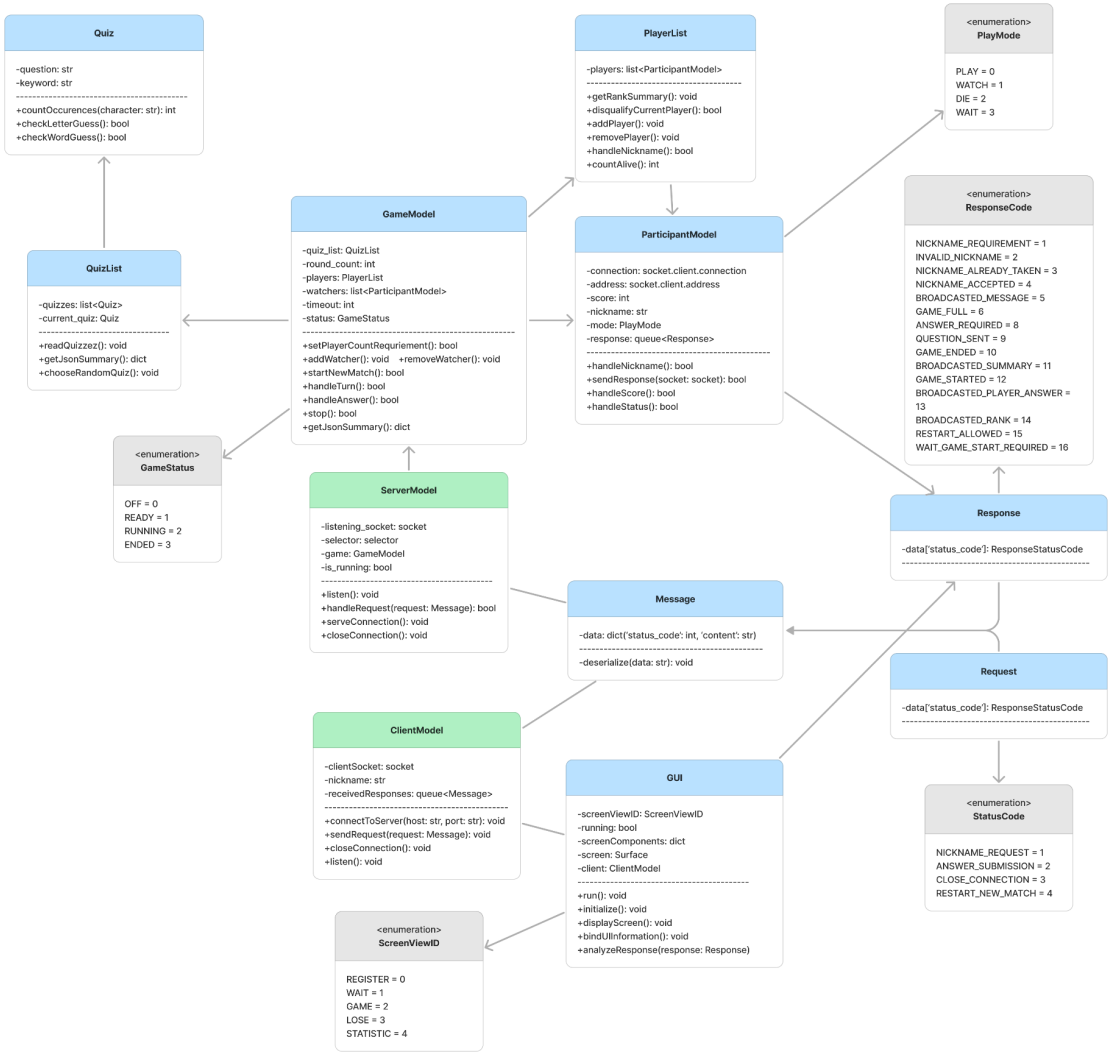
- The steps to check the packet at server and clients include:

- Extract the “status code” and “content” parts from the packet.
- For some responses or requests, the “content” parts will be deserialized after they are extracted from the received packets.
- The server and clients will decide their actions according to the given “status code” and “content” parts.
- During processing packets, the server or clients will ignore the packet if its status code can not be recognized.
- The following image shows a part of source code where the packets (responses in this case) are processed. As it can be seen, the program extracts the “status code” and “content” from the packet and the program then uses “status code” to further run the corresponding statements. For instance, when the *NICKNAME_ALREADY_TAKEN* or *INVALID_NICKNAME* responses are received, the program would update its user interface to let users know what problem the chosen nickname has.

```
def analyzeResponse(self) -> bool:
    response = self.client.getReceivedResponse()

    if response is None:
        return False

    statusCode = response.getStatusCode()
    content = response.getContent()

    if statusCode == ResponseStatusCode.BROADCASTED_SUMMARY:
        self.summary = json.loads(content)

        self.bindSummaryUI()

        #print(f"[CLIENT] Received summary: {json.dumps(self.summary, indent = 4)}")

    elif statusCode == ResponseStatusCode.NICKNAME_ALREADY_TAKEN:
        self.openScreenComponents['notify'].changeTextContent(MessageTextConstants.ERROR_MESSAGE_REGISTER)

    elif statusCode == ResponseStatusCode.INVALID_NICKNAME:
        self.openScreenComponents['notify'].changeTextContent(MessageTextConstants.INVALID_MESSAGE_REGISTER)

    elif statusCode == ResponseStatusCode.BROADCASTED_RANK:
        self.rankSummary = json.loads(content)
        self.bindStatisticUI()
        self.screenViewID = ScreenViewID.STATISTIC
```

DATA STRUCTURE

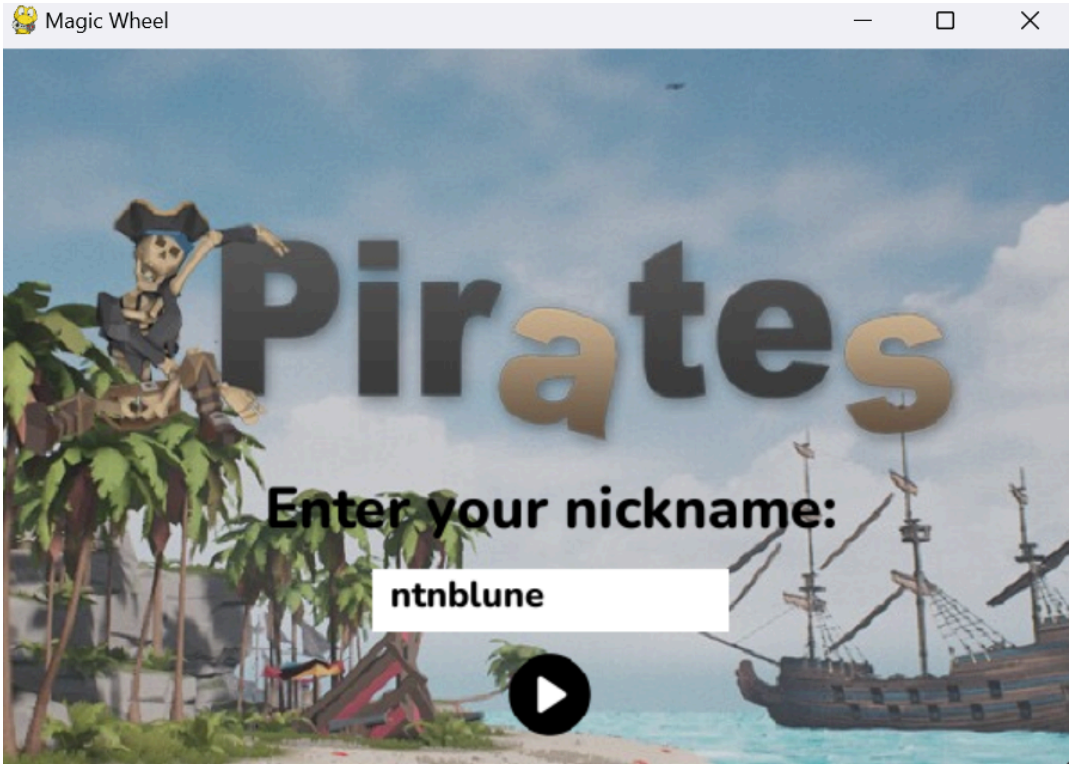
The class diagram provided below serves as a foundational blueprint for our program. It was designed prior to the commencement of the coding phase to provide a clear architectural understanding of the system. While the functions and variables depicted in the diagram are concise, it's important to note that there may be additional elements present in the source code that are not explicitly illustrated here.



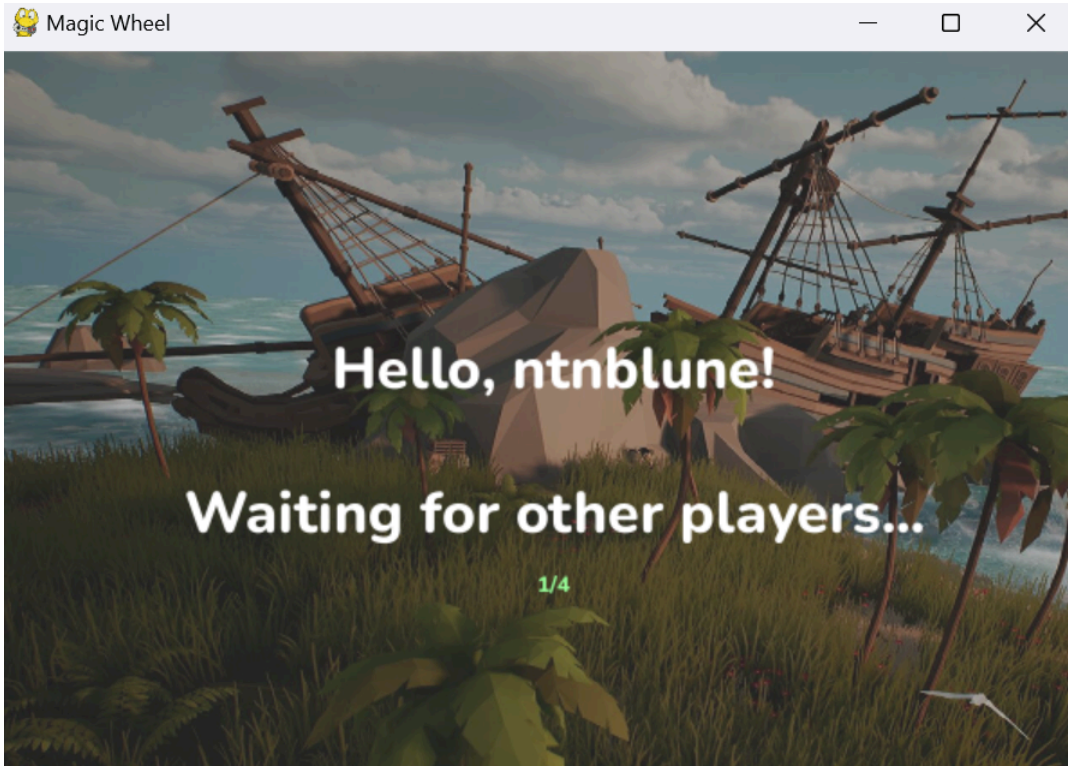
GRAPHICAL USER INTERFACE

Start Screen (Register Screen)

Appear after the client has successfully connected to the server. If your nickname is not valid (already been used by another player/ empty/ invalid symbol) there will be a notify sentence to ask for retrying.


Magic Wheel
Pirates
Enter your nickname:
ntnblune

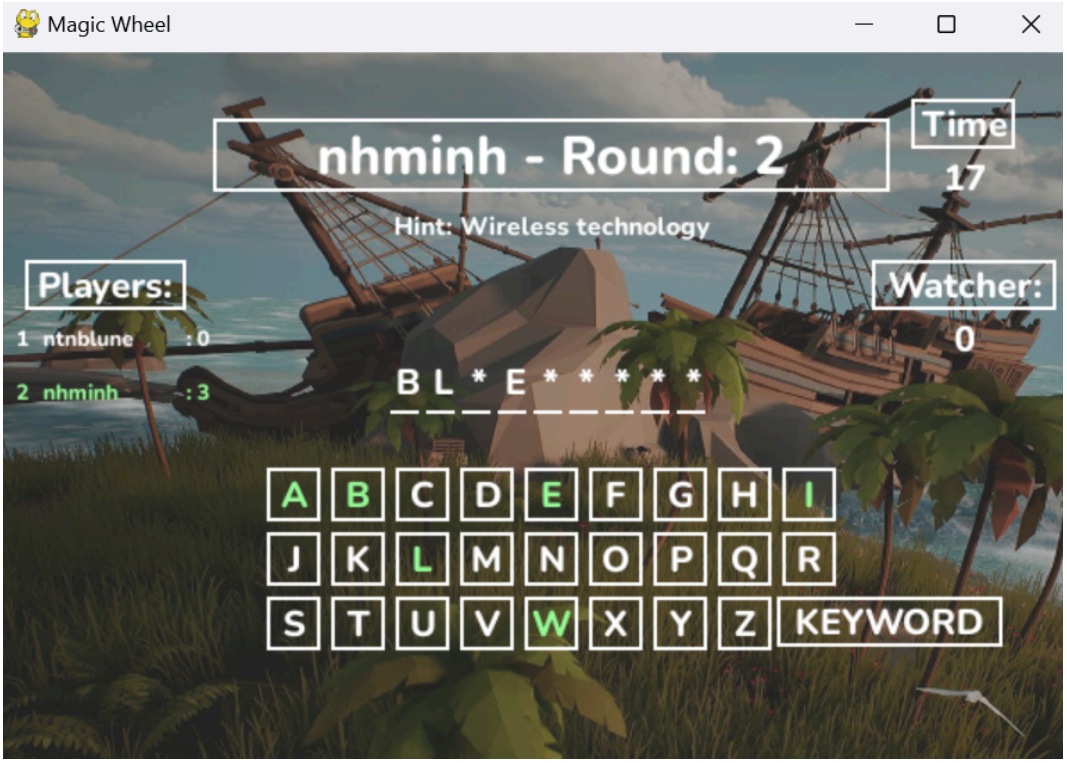
Wait Screen


Magic Wheel
Hello, ntnblune!
Waiting for other players...
1/4

In this screen, you will be acknowledged how many players are required to start the game as well as the number of successfully registered player at the moment, including you.

Ingame Screen

The top area will display your <nickname> - <current round>. Below it is a short hint for the keyword. A scoreboard of all players is also displayed with green highlighted the player having the current turn.



Green keyboards are letters which are predicted by all players. In your turn, to guess a letter, simply click on the corresponding text button on the screen.

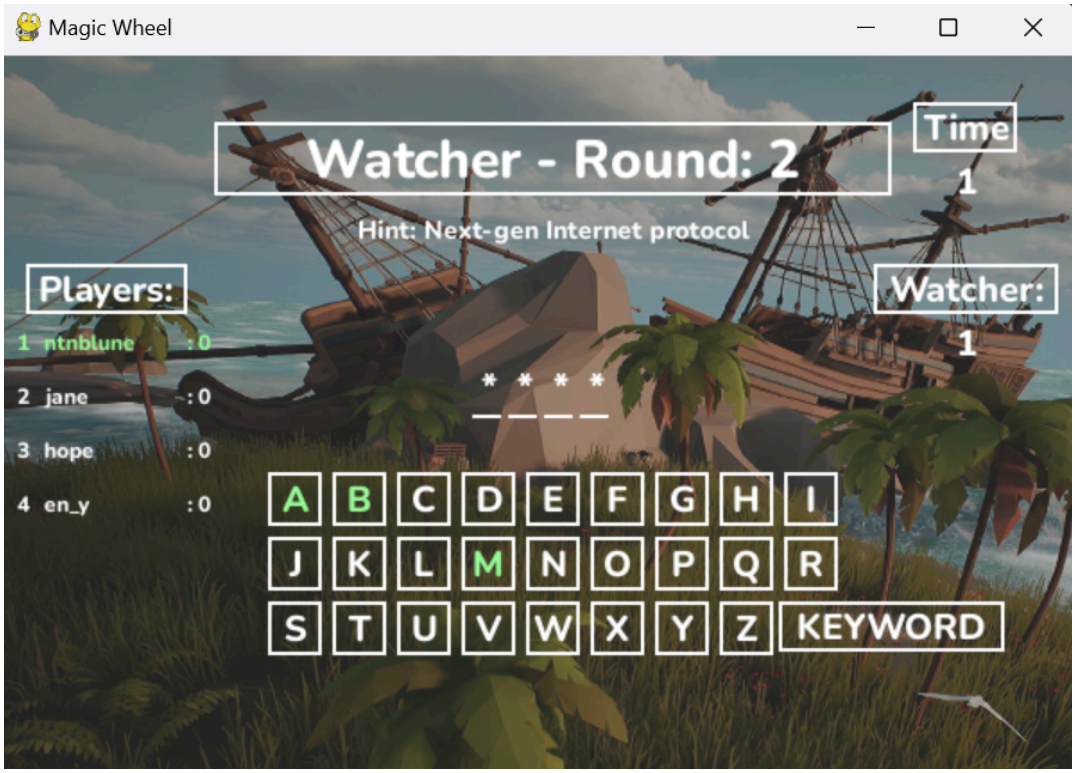
The time is synchronized with the server and will count down from 20 to 0. If it is counted to 0, the player in that turn will be recorded as guessing <null> letter.

Players can also know how many watchers are watching them (in the dark).

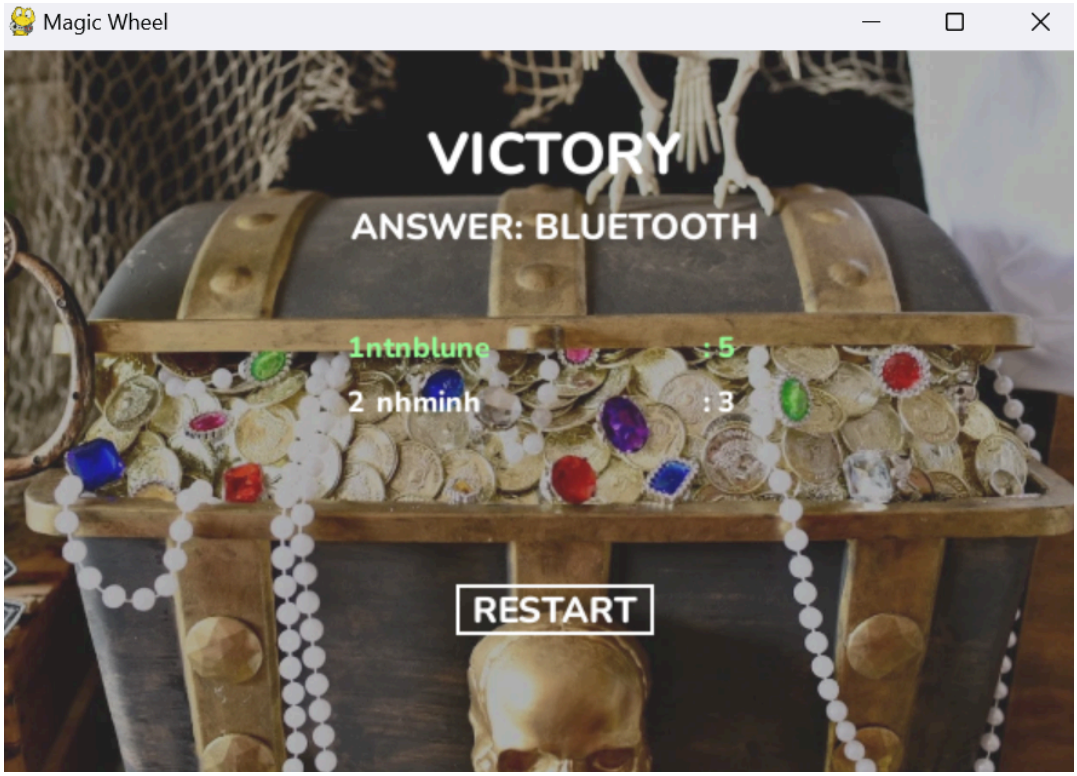
Keyword button is able to be pressed with respect to the rule mentioned before. When pressing this button, a text form appears so as to input the guessing word.

Watcher Screen

Quite similar to the player's screen, but watchers can not participate in the game.



Game Over Screen



The “RESTART” button is enabled only when the server administrator decides to hold another game.

SELF-EVALUATION

No.	Name	Student ID	Contribution (%)
1	Nguyen Hoang Minh	21125018	50%
2	Nguyen Truc Nhu Binh	21125034	50%

No.	Requirements	Score	Evaluate

1	Use C/C++, Java, C#, Python	2	2 (Python)
2	Implement whole gameplay properly	3	3
3	Socket Non-blocking	2	2
4	Have a good GUI (MFC, WPF, Swing, etc.)	3	3 (Pygame)
	Total	10	10

RESOURCE

Demo video: CS494 - INTERNETWORKING PROTOCOL - SOCKET PROGRAMMING (youtube.com)

Source code: https://github.com/nguyenhoangminh31082003/CS494-Lab

REFERENCES

1. https://www.pygame.org/news
2. NTVinh2000/MagicalWheelSubfolder (github.com)