Programming Assignment 5

Goal

- Get familiar with UDP socket API.
- Program simple network based multiplayer game.

Description

In this assignment, we will design and implement a simple network based multiplayer snake game. We will use pygame to do the graphic rendering part. I have included a demo how to use pygame library implement a local Snake game in `snake.py` file. For the details of each API used, please refer to the pygame API doc.

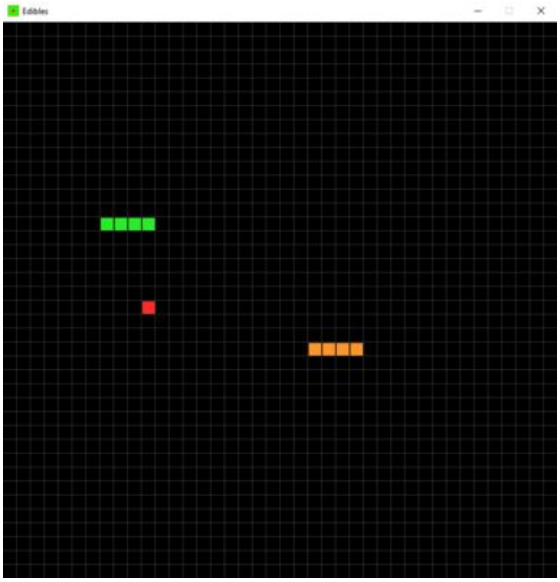
Rules of multiplayer snake game

In this assignment, we limit the requirements to only support two player snake game. When a two player snake game starts, you will see something like the picture below.

- Snake in green represents the snake you control.
- Snake in orange represents the snake your opponent controls.
- Apple is represented by the red dot.

The goal is to outlast your opponent in this game. You will lose the game if (1) your snake collide with the edge of the window, or (2) your snake collide with the tail/body of itself, or (3) your snake collide with the tail/body of your opponent's snake.

The game is considered a draw if two snakes bump heads.



Life of a game

The game is presented by a fix-size `32 by 32` board. Each cell in the board won't be shown as a 1 by 1 pixel in the UI (which will be too small), instead each cell will be shown as 20 by 20 pixel block.

The first player is responsible for creating the game, using the following command

```
python3 client.py create [game_id] [nick_name] [port_number]
```

The `game_id` will be unique to identify the game, `nick_name` will be used to show who wins the game at the end, and `port_number` is used to specify at which this client will receive messages from the game server. For example, you may use something like below to create a game

```
python3 client.py create 123 snakemaster 8080
```

After this is done, the board of the snake game should show up, with "`waiting for opponent`" printed in the middle. Like the picture shown below.

The second player can use the following command to join a game.

```
python3 client.py join [game_id] [nick_name] [port_number]
```

The `game_id` should be the same as the one first player created, `nick_name` will be used to show who wins the game at the end, and `port_number` is used to specify at which this client will receive messages from the game server. For example, you may use something like below to join a game.

```
python3 client.py join 123 masterkiller 8181
```

After this is done, the board of the snake game for both players should show "`game is about to start`" for a second, and then start.

When the game ends (there is a winner or a draw), the board of the snake game should show "`XXX is winner`" or "`It is a draw`".

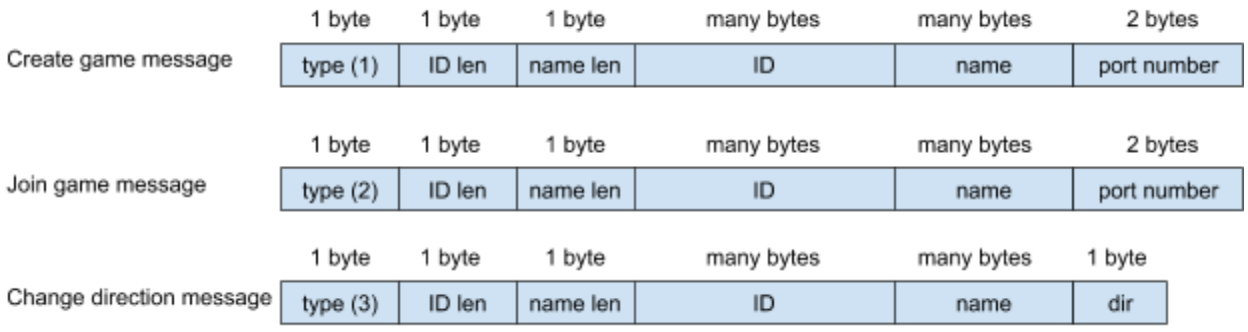
Design

In this assignment, we will use UDP socket to send data between client and server. And we assume the network is "reliable" (no message will be lost or reordered). This assumption is mainly used to simplify your logic, and we already practiced how to implement reliability in the previous homework.

In online gaming, the server usually keeps all the state information (e.g. snake position on the board, position for the apple, moving directions for each snake, etc.), and clients only send control information to the server (e.g. create game, change snake direction, etc.).

Client to server message format

In our design, client only sends control information to the server (e.g. change snake direction). We design the application layer message format as below



Detail specifications

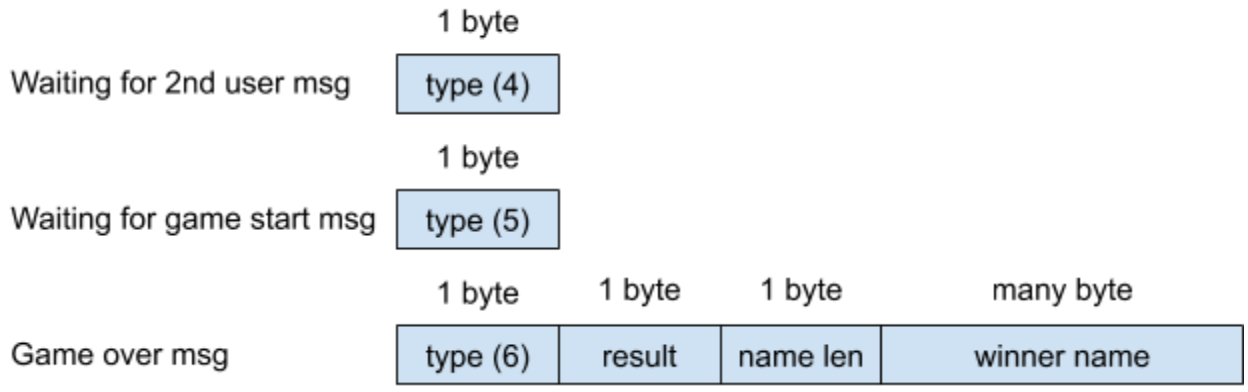
- Create game message is sent when the first player creates the game, which contains the following fields
    - message type (value 1) using 1 byte
    - game_id length using 1 byte

    - nick_name length using 1 byte
    - game_id
    - nick_name
    - port_number using 2 bytes
- Join game message is sent when the second player joins the game, which contains the following fields
    - message type (value 2) using 1 byte
    - game_id length using 1 byte
    - nick_name length using 1 byte
    - game_id
    - nick_name
    - port_number using 2 bytes
- Change direction message is sent whenever client uses keyboard to change the moving direction of the snake, which contains the following fields
    - message type (value 3) using 1 byte
    - game_id length using 1 byte
    - nick_name length using 1 byte
    - game_id
    - nick_name
    - direction using 1 byte (0 means UP, 1 means RIGHT, 2 means DOWN, 3 means LEFT)

Server to client message format

In our design, server will send the state information to the client, which controls what information should be rendered to users. The server is sending a message to clients every 50 ms. Depending on what state the game is in, please choose one of the following messages.

When the game is created, both users join, or the game is over, the server sends the following messages to the clients.

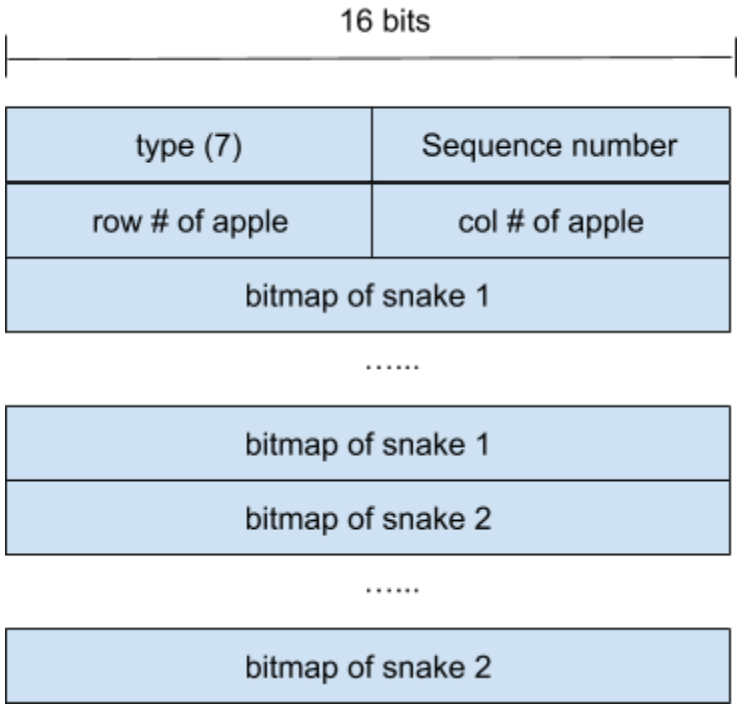


Detail specifications

- Message indicating waiting for 2nd user. This message will be sent when the 1st user created the game, but the 2nd user hasn't joined yet.
    - message type (value 4) using 1 byte

- Message indicating waiting for the game to start. This message will be sent when the 2nd player is joined, and before the game actually starts (there is a 1 sec delay to make sure both players get prepared to start)
    - message type (value 5) using 1 byte
- Message indicating the game result, which contains the following fields.
    - message type (value 6) using 1 byte
    - result using 1 byte (0 means draw, 1 means there is a winner)
    - nick_name length using 1 byte (this field exists when result field has value 1)
    - nick_name (this field exists when result field has value 1)

During the game, the server will send the following message to both clients every 50 ms.



Detail specification on each field

- message type (value 7) using 1 byte
- sequence number using 1 byte
- row number of the apple using 1 byte (value can range from 0 to 31)
- column number of the apple using 1 byte (value can range from 0 to 31)
- bitmap of snake 1 using 128 bytes
    - each row has 32 positions, using 1 bit for each, and total 32 rows
- bitmap of snake 2 using 128 bytes
    - each row has 32 positions, using 1 bit for each, and total 32 rows

Grading policy

100 points in total.

- Correct use of UDP socket
- [40 pts] Client implementation

    - [5 pts] Send create game message correctly
    - [5 pts] Send join game message correctly
    - [10 pts] Send change direction message correctly
    - [10 pts] Correctly receive messages from server
    - [10 pts] Correctly render according to the messages received from the server
- [40 pts] Server side implementation
    - [5 pts] Send waiting for 2nd user to join message correctly
    - [5 pts] Send waiting for game to start message correctly
    - [5 pts] Send game over message correctly
    - [15 pts] Send in game information correctly
    - [10 pts] Correctly receive control messages from clients and update accordingly
- [20 pts] Game logic
    - [5 pts] Correctly moves snake given their current directions
    - [5 pts] Correctly handle snake eating apples
    - [10 pts] Correctly implement the game termination logic (win or draw)

Submission instruction

Please your code and report to all TAs and cc z.sun@northeastern.edu by the deadline.