

Project: Socket Programming

Due date: Friday 5pm, 14 May 2021

Drop dead due date: Friday 5pm, 21 May 2021 (5% penalty on raw grade per day)

Worth: 20% of the whole grade

This Project is to expand the knowledge on socket programming. You are required to research further on socket programming in addition to the materials provided in this unit in order to complete the Project. Please set aside enough time to finish this Project rather than leaving it too late, as it will require some time to complete. You are expected to use Python as your programming language, but you are also welcome to use other languages such as C, Java etc. If you do decide to use another language, you **must let the unit coordinator know** before you proceed.

1. Intro: Your new job as a game developer

Congratulations! You've been given an internship at one of the larger game developers, *Electronic Farce*, who have just launched a new and addictive turn-based strategic tile placing game. The company initially started developing this game last year, but due to the pandemic, the engineer in charge left the project and had to focus on maintaining RNG battle royale that was developed back in 2019. EF decided to reboot this game project by hiring you.

The previous engineer managed to write the client code, but did not provide any documentations. Because the client is already written, the message format for communication between the server and clients is already set. You can use the client to test your server as your development progresses. He also wrote a small sample server written in Python. But this server is deficient in that it only allows a single client to play a rather pointless game alone. However, you can inspect this server to see how to pass and receive the predetermined game messages. You can also use the server to see the basic flow of the game.

In a rather spectacular but not altogether surprising fashion, EF have tasked all of the interns (you) to implement their server in a short period of time. Your task is to write an upgraded server for the strategic tile placing game. We provide more specific game details below.

2. The Game

The game in question is a strategic tile placing game. Each player is given a set of random tiles (from a preset list of tiles) by the server. A tile has two points on each side (north, east, south, and west). Each of these eight points is connected to one of the other points on the tile, so there are four connections in total. Example tiles are shown in Figure 1.

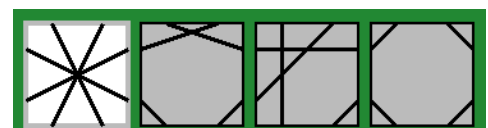


Figure 1. Example tiles (first tile selected in game)

CITS 3002

Computer Networks

The gameplay

The board is initially empty. One at a time, in an order determined by the server, each player places a single tile on the board, ensuring that the first tile is connected to the edge of the board.

In the second turn, each player chooses where their token will enter the board. This must be one of the points on the tile that they placed in the first turn, so there will be either two or four possible locations (depending on whether or not the tile was placed in a corner).

The player's token automatically follows the connection across the tile, reaching a new square of the board. If that board already has a tile, the token will follow the connection in the new tile, continuing until it either reaches an empty square or the edge of the board.

If a player's token reaches the edge of the board, that player is eliminated.

On the third turn, and all subsequent turns, each remaining player may place a single tile from their hand onto the empty square that their token is entering. Any other players who are entering the same square will be moved automatically according to the connections on the placed tile, so it is possible for other players to be eliminated.

If only one player remains alive, that player is considered the winner.

You can find what it looks like as a multiplayer in Figure 2 (initially, you will only be able to play with 1 client with the provided server code). You can also see what kind of messages are being sent between the server and clients in Figure 3.

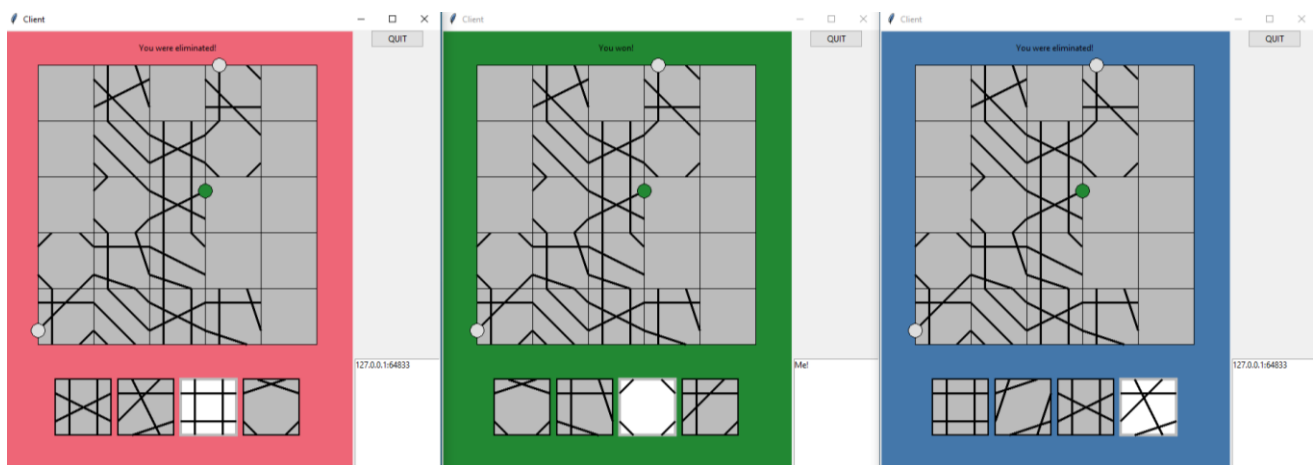


Figure 2. Game play (3 players - green player won)

CITS 3002

Computer Networks



THE UNIVERSITY OF
WESTERN
AUSTRALIA

```
Command Prompt - python server-solo-threaded.py
remove eliminated players: []
read A player placed his/her tile!
client ('127.0.0.1', 64833) msg A player placed his/her tile!
remove eliminated players: []
read A player placed his/her tile!
client ('127.0.0.1', 64835) msg A player placed his/her tile!
remove eliminated players: []
read A player placed his/her tile!
client ('127.0.0.1', 64836) msg A player placed his/her tile!
remove eliminated players: []
read A player placed his/her tile!
client ('127.0.0.1', 64833) msg A player placed his/her tile!
remove eliminated players: []
read A player placed his/her tile!
client ('127.0.0.1', 64835) msg A player placed his/her tile!
remove eliminated players: []
read A player placed his/her tile!
client ('127.0.0.1', 64836) msg A player placed his/her tile!
remove eliminated players: [1]
read A player placed his/her tile!
client ('127.0.0.1', 64833) msg A player placed his/her tile!
remove eliminated players: [1]
read A player placed his/her tile!
client ('127.0.0.1', 64835) msg A player placed his/her tile!
remove eliminated players: [2]
read A player placed his/her tile!
client ('127.0.0.1', 64836) msg A player placed his/her tile!
remove eliminated players: [3]
game finished!
starting countdown

Command Prompt - python client.py
tile 2 at 1, 2 : 0 from 1
Move token: Player has decided its starting position!
Player turn: A new turn has started!
play tile at 3, 1
Place tile: A player placed his/her tile!
tile 2 at 3, 1 : 0 from 2
Add tile 0 to hand
Move token: Player has decided its starting position!
Player turn: A new turn has started!
Place tile: A player placed his/her tile!
tile 2 at 0, 3 : 1 from 3
Move token: Player has decided its starting position!
Player turn: A new turn has started!
Place tile: A player placed his/her tile!
tile 1 at 2, 2 : 2 from 1
Move token: Player has decided its starting position!
Player turn: A new turn has started!
play tile at 5, 0
Place tile: A player placed his/her tile!
tile 0 at 3, 0 : 1 from 2
Add tile 10 to hand
Move token: Player has decided its starting position!
Player eliminated: A player has been eliminated!
Player turn: A new turn has started!
Place tile: A player placed his/her tile!
tile 0 at 0, 4 : 0 from 3
Move token: Player has decided its starting position!
Player eliminated: A player has been eliminated!

Command Prompt - python client.py
tile 2 at 1, 2 : 0 from 1
Add tile 8 to hand
Move token: Player has decided its starting position!
Setting own location
Player turn: A new turn has started!
Place tile: A player placed his/her tile!
tile 2 at 3, 1 : 0 from 2
Move token: Player has decided its starting position!
Player turn: A new turn has started!
Place tile: A player placed his/her tile!
tile 2 at 0, 3 : 1 from 3
Move token: Player has decided its starting position!
Player turn: A new turn has started!
play tile at 2, 2
Place tile: A player placed his/her tile!
tile 1 at 2, 2 : 2 from 1
tile 1 at 2, 2 : 2 from 1
Add tile 1 to hand
Move token: Player has decided its starting position!
Setting own location
Player turn: A new turn has started!
Place tile: A player placed his/her tile!
tile 1 at 2, 2 : 2 from 1
Move token: Player has decided its starting position!
Player turn: A new turn has started!
Player eliminated: A player has been eliminated!
Player turn: A new turn has started!
Place tile: A player placed his/her tile!
tile 0 at 0, 4 : 0 from 3
Move token: Player has decided its starting position!
Player eliminated: A player has been eliminated!
```

Figure 3. Game play (behind the scenes)

Program Flow/Requirements

For sake of simplicity we specify the flow of a single game below.

1. A selection of at most four players is chosen from all connected clients.
2. A random turn order for the four players is also determined.
3. Each client is notified that a new game is starting.
4. Each player is provided tiles to fill their starting hand.
5. While more than one player is alive:
 - a. Notify all clients whose turn it is
 - b. Wait for the current player to make a valid move
 - c. Process the results of the move
 - i. Notify all clients of new tile placements
 - ii. Notify all clients of new token positions
 - iii. Notify all clients of any eliminated players
 - iv. If there are less than two players remaining, exit the loop
6. If the current player played a tile, send them a new tile
7. Switch to the next (remaining) player's turn

When a game is completed, the server should start a new game if there are enough clients available. Otherwise it should wait for more clients to join.

CITS 3002

Computer Networks



Exceptions in gameplay

Note that much of the game logic about legal tile placement, etc., is already handled for you in the `tiles.py` module. However, your server will still need to ensure that players behave correctly, for example:

- Players should only be allowed to play tiles that are currently in their hand.
- If a player sends a message when it is not their turn, it should be ignored.
- Players should only be able to choose a token position on their second turn (i.e. when they don't already have a token on the board).

3. Tasks

Your goal is to implement a server which implements the game described above. To get started, you are provided with:

- `tiles.py` - This module defines essential constants and gameplay logic, which is shared by both the client and the server.
- `server.py` - This module implements a basic server that allows a single client to play a single game with no other participants, and very little error checking.
- `client.py` - This module implements the game client. You do not need to understand this code, but you may like to inspect it to understand how the client expects to interact with the server.

To guide your efforts, we provide four 'tiers' of tasks to complete. Solutions which meet Tier 1 requirements are minimally viable, higher tiers offer more granulated mark allocations for solving more complex problems. We will only assess a single submitted implementation (i.e. the tiers 'stack' on each other rather than implementing the requirements of each separately).

Tier 1 - Enabling multiple players

Implement a basic server which can play a single game with two players. All players are expected to behave themselves insofar as they will only play cards from their hands, and they will only choose valid starting locations. Players may send messages when it is not their turn - you can choose to either ignore these messages, or to wait and process them when it is the players turn next.

All connections are expected to be fast and stable, so your server does not need to worry about clients disconnecting partway through the game (all of our players politely stay until the conclusion, even if they are destined to lose).

CITS 3002

Computer Networks



Tier 2 - Scaling up

Extend your server so that when a game finishes, if there are enough clients still connected to the server, it begins a new game. Optionally, you can wait for a certain amount of time before starting the new game (so that the players can reflect on the outcome of the previous game).

Your server should also be able to handle games of up to four players. If there are more than four clients connected, your server should randomly pick four of the clients to be the players. All other connected clients will be spectators, and should still receive updates about the game state. Any message received from a spectator or an eliminated player should be ignored (it should not carry over to the next game, if there is one).

Tier 3 - Connection issues

In addition to all previous requirements, your server must be able to handle unexpected network behaviour. For simplicity, we will consider two scenarios (but you should think about more and report about how you implemented those handlers).

- If a player exits during the middle of a game, they should be eliminated from the game. If it was that player's turn, the server should gracefully proceed to the next player's turn. If there is only one player remaining after the disconnection, the game should be finished.
- The server should also handle players attempting to join during a game. These players may be sent updates about the game state, but they should not be allowed to play any turns during this game (they are effectively spectators).

Tier 4 - Player issues

Next, we should make the experience nicer for players that join partway through an existing game, by helping them to catch up on the game state. To do this completely (for the existing client) we need to:

- Notify the client of the id number of all players that started in the current game, using `MessagePlayerTurn`.
- Notify the client of all players that have been eliminated from the current game, using `MessagePlayerEliminated`.
- Notify the client of the real current turn using `MessagePlayerTurn` again.
- Notify the client of all tiles that are already on the board, using `MessagePlaceTile`.
- Notify the client of all token positions, for players that have a token position, using `MessageMoveToken`.

When this is complete, a client joining the server at any time during an existing game should see the same game board as any other client.

Finally, let us ensure that the game continues for all players, even if we don't hear from a particular player for a long time. If a player doesn't make a valid move within 10 seconds, then the server should make a valid move for them:

CITS 3002

Computer Networks



- If it is the player's first turn, the server should choose a random tile from the player's hand, and place it on a random, empty border square, with a random rotation value (from 0 to 3 inclusive).
- If it is the player's second turn, the server should choose a starting token position at random from the positions available to the current player.
- If it is any subsequent turn, the server should choose a random tile from the player's hand, and place it on the square that the player's token is currently on, with a random rotation value (again, 0-3).

4. Report

In addition to your server code you must write a report to discuss the following items.

- How would you scale up your solution to handle many more clients?
- How could you deal with identical messages arriving simultaneously on the same socket?
- With reference to your project, what are some of the key differences between designing network programs, and other programs you have developed?
- What are the limitations of your current implementation (e.g. scale, performance, complexity)?
- Is there any other implementations outside the scope of this project you would like to mention?
- Any other notable things to discuss.

Your report should include your name and student ID number. It should be no more than 2 pages.

5. Submission

Submit your **server code** and **report** on LMS by the deadline. Also note the following:

Code submission: You should submit a single code that captures all the tiers you have implemented.

Test environment: You may wish to specify the test environment if you find that a specific environment is needed to test your server (e.g., Linux, Mac). Specify this clearly either in the module docstring, or separate readme file, or anything else appropriate and easy to find/follow.

CITS 3002

Computer Networks



THE UNIVERSITY OF
WESTERN
AUSTRALIA

6. Rubrics

	Criteria	Highly Satisfactory	Satisfactory	Unsatisfactory
Tier 1 (35 marks)	<ul style="list-style-type: none"> Satisfy all conditions of tier 1 tasks Game play with two clients Code is reasonable to read with good comments 	Demonstrated the ability to implement all game features for tier 1 <ul style="list-style-type: none"> Correctly implement all functionalities for tier 1 tasks. Code is clearly legible with sufficient comments to understand. 	Demonstrated the ability to implement some game features for tier 1 <ul style="list-style-type: none"> Many functionalities for tier 1 tasks are implemented. Code is clearly legible with limited comments to understand. 	Failed to demonstrate the ability to implement tier 1 <ul style="list-style-type: none"> Very few functionalities for tier 1 tasks are implemented. Code is not legible without sufficient comments to understand.
Tier 2 (15 marks)	<ul style="list-style-type: none"> Multiple (up to 4) players can play at the same time Start new game with connected clients 	Demonstrated the ability to implement all game features for multi-players <ul style="list-style-type: none"> Players up to 4 can join the game. New games are initiated without errors 	Demonstrated the ability to implement some game features for multi-players <ul style="list-style-type: none"> Players up to 4 can join the game. New games are initiated without errors 	Failed to demonstrate the ability to implement multi-players <ul style="list-style-type: none"> Multi-players are not supported. New games cannot be initiated correctly
Tier 3 (15 marks)	<ul style="list-style-type: none"> Satisfy all conditions of tier 3 tasks Handles player joining/exiting mid-game 	Demonstrated the ability to handle players joining/exiting the game <ul style="list-style-type: none"> Players can exit mid-game without interrupting the server. Players must wait until current game finishes to join the game. 	Demonstrated the ability to handle players joining or exiting the game <ul style="list-style-type: none"> Players can exit mid-game without interrupting the server, or Players must wait until current game finishes to join the game. 	Failed to demonstrate the ability to handle players joining/exiting the game <ul style="list-style-type: none"> Players exiting mid-game interrupts the server. Players joining mid-game interrupts the server.
Tier 4 (15 marks)	<ul style="list-style-type: none"> Satisfy all conditions of tier 4 tasks Players are updated with game states Game continues to play with timeouts 	Demonstrated the ability to implement fair play <ul style="list-style-type: none"> Clients send and receive correct messages with the server receive the updated game states Game continues to play with idle clients 	Demonstrated the ability to implement some fair play <ul style="list-style-type: none"> Clients send and receive correct messages with the server receive the updated game states, or Game continues to play with idle clients 	Failed to demonstrate the ability to implement fair play <ul style="list-style-type: none"> Cannot update the game states for clients Game fails to play with idle clients
Report (20 marks)	<ul style="list-style-type: none"> Comprehensive understanding of the server developed in the project Able to fluently articulate the implementation to the audience. 	Demonstrated the ability to write a comprehensive report <ul style="list-style-type: none"> Detailed discussion for questions asked. Clear explanations and reasonable justifications are provided. In-depth knowledge of the project demonstrated. 	Demonstrated the ability to write a reasonable report <ul style="list-style-type: none"> Some discussion for questions asked. Some explanations and reasonable justifications are provided. Some knowledge of the project demonstrated. 	Failed to demonstrate the ability to write a report <ul style="list-style-type: none"> Failed to discuss the questions asked. Vague or no explanations and justifications are provided. Not much knowledge of the project demonstrated.

This project is out of total 100 possible marks.