Table of Contents

I. Overview

In the last couple of years the use of Java has become very widespread in the IT - industry. It is used mainly for Internet software, but it is also popular for regular applications and in embedded systems. Java is not only simple but also supported multiple platforms. In this project, we will use Java for developing an airforce game which allows multiple players to play together on LAN.

II. Basic features

- The game is run over network following Client – Server architecture.
- Many clients can connect to server. After connecting to server, they must login and can request to join game or start a new game (when the number of clients is enough).
- In a game stage, clients control their plane move and shoot. The enemy planes move from top to bottom with the speed that is increased when the game level increases.
- The information of a game stage is updated real-time to every client.
- When the enemy plane is shot by clients, it is removed from the screen.
- When the plane of client collides with the enemy, the player will lose the game and the rest player still can continue to play.

III. Extensive features and implementation

- Player can select plane's style before starting game.

  There are three kinds of plane for the player to select freely. Players should be able to select a plane that they feel comfortable playing and make their planes easy to distinguish on the battlefield.

- Enemy's plane can fire.

  Each frame, enemy has 5% to fire, and the interval between 2 shots is not less than 1 second.

- Player's plane can not only move left, right but also move forward and backward.

  Use 4 more action code: UP_PRESSED, UP_RELEASED, DOWN_PRESSED, DOWN_RELEASED and send the corresponding code when player press / release a button. The server then receives the code, update the coordinates of that player.

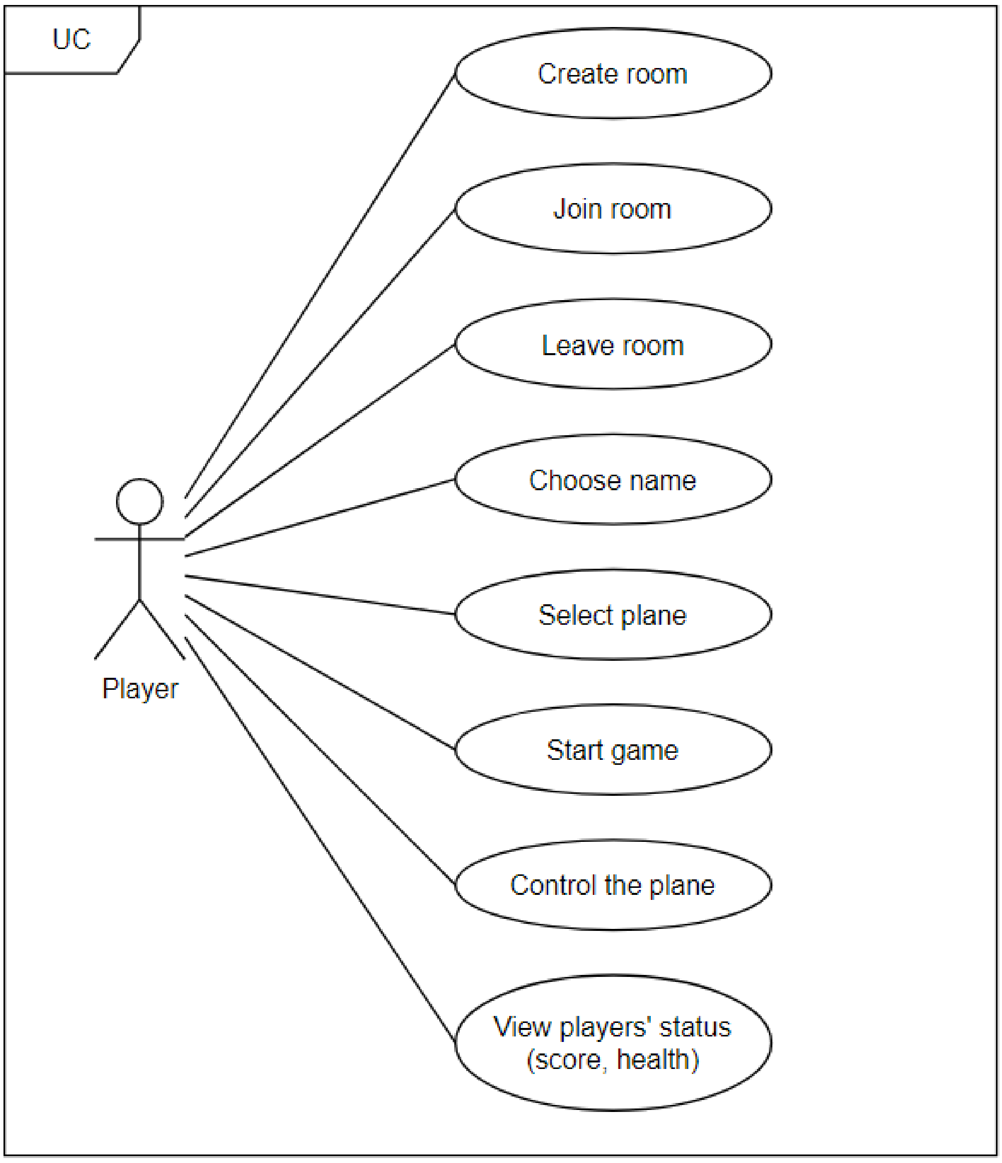
- Player has 3 lives, each time his plane collides with enemy's plane or get shot by the enemy, he will lose a life. The player will lose the game when his health reaches 0.

  By this way, a player can recover from making a disastrous mistake. Multiple lives also allow novice players a chance to learn a game's mechanics before the game is over.

- Background sound is integrated

  Background sound helps greatly to emphasis and highlight the feeling of being on the battlefield.

IV. Use case diagram



V. Design the application protocol

Client and Server communicate through socket by TCP/IP protocol. Communication information of client and server will be encapsulated into a packet so that client or server can detect easily and handle them suitably.

Packet design:

No.	Packet	Information	Used at
1	AddConnectionRequestPacket	PlayerName: String IsMaster: Boolean	Client requests connection to server
2	AddConnectionResponsePacket	Id: Integer IsConnectSuccess: Boolean PlayerName: String Message: String	Server replies client's connection request
3	ChangePlaneTypeRequestPacket	Id: Integer PlaneType: Integer	Client requests change type of plane
4	ClosedServerNotificationPacket	Message: String	Server notices to clients that it is closed
5	GameOverPacket	PlayersInGame: ArrayList	Server notices game over to clients
6	NotReadyWarningPacket	Message: String	Server notices that all clients must be ready
7	PlayerIngameActionPacket	Action: Enum	Client sends its action to server
8	ReadyRequestPacket	Id: Integer IsReady: Boolean	Client sends ready state to server
9	RemoveConnectionPacket	Id: Integer PlayerName: String	Client notices to server about its disconnection
10	ServerNotFoundPacket	Message: String	Server notices to client that it exists no room
11	StartGameRequestPacket		Room master client requests start game
12	StartGameResponsePacket	PlayersInGame: ArrayList	Server replies the start game request
13	UpdateIngameInfoPacket	PlayersInGame: ArrayList PlayerBullets: ArrayList EnermyBullets: ArrayList Enermies: ArrayList	Server sends update information in game stage to clients
14	UpdateRoomInfoPacket	ClientsInRoom: ArrayList	Server sends update information in room to clients

VI. Implement client and server following the application protocol with Java

- Server:
  - Server has a thread that always listens at socket with local host of InetAddress and port 6666 so that every client in LAN can request connection to communicate.

```
while (running) {
    try {
        Socket socket = serverSocket.accept();
        initSocket(socket);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

  - Server will create a new connection to communicate with client when it requests connection, but this connection will be closed immediately if the room is full.
  - If server's room is not full, server will provide an ID to client and maintain the connection among them.
- Client:
  - Client requests connection at socket with port 6666
  - If client is added to the room, it will store the ID provided by server for communication.
- Communication:

Client and server always listen the information transmitted by the other. Their Event Listener will detect each type of packet by checking instance of class and handle it suitably.

```
} else if (p instanceof GameOverPacket) {
    GameOverPacket packet = (GameOverPacket) p;
    handleGameOverPacket(packet);
}
```

  - Server:
    - Listens requests of clients and sends response information.
    - Updates the information of game stage every 1/60 second when starting game stage (60 FPS).
  - Client:
    - Sends requests to server.
    - Listens the information and responses from server and handles them and post an event GUI for update.
    - In order to help the GUI update immediately when client finished handling the information received from server, we use the Event-driven architecture supported by EventBus library.

VII. References

https://github.com/Arcxes/Java-Multiplayer-Tutorial

https://computermaster5089.blogspot.com/2018/03/how-to-make-sky-force-game-in-java_22.html

https://mvnrepository.com/artifact/com.google.guava/guava