The WS_2017 Game
The Programm Architecture

Flavia Brogle
Max Hackinger

May 22, 2017

Contents

Abstract

This article describes the Program Architecture of the WS_2017 Game, starting with a quick overview of all the software components of Game. In the next three sections it goes in to grater detail about how the Client and Server side are constructed and how they communicate.

1 Quick over view of the Game Program

WS_2017 is a round based game that consists of a clients and a server which communicate through the WS_2017 protocol. The protocol is inspired by the design of the POP3 protocol and uses commands that consist of a keyword and arguments.

2 Client side

2.1 Chats and Main Window

We created our own ChatPanel Class which can be used as different kinds of Chats in different Windows. When created the chat command is defined (chatm, chatl or chatw) so it can be used for the three different chat-types there are in our game: - the Main Chat with all the logged-in users, - the Lobby Chat of which one exists per game and - the Whisper Chat with which users can privately chat. The Main Window is one of the biggest Classes which holds the List of all the Games on the Client Side (directly in the JList in which they are displayed) and the users. This way we avoid possible bugs where a game is shown which doesn't exist anymore. All the interaction is read in from the window. It is not directly processed but sent to the server and every client relevant to the information (i.e. the two clients in a whisper chat) including the one which received the user input get the update from the server and only then it is processed.

2.2 Lobby

When a client joins a game the lobby window is automatically opened in which all the players in this game can chat with each other. Additionally another window is created where the client can choose the Children. The Children are handled as JPanels which hold JComponents to customize the Child. When pressing the okay button the window class transforms the information about the children in a string and gives it to the GameController via server.

3 Server side

3.1 Game State and Connection Handling

When a server instance is first spun up, a main thread is crated that in turn spins up a new thread for every incoming TCP connection. All the Structures that Need to be accessed by all clients are in the main thread, this includes ArrayLists, one containing all currently running games and another all connected Users.

3.2 Protocol Validation and Processing

The individual threads of each client handle the processing and validation of the protocol. In the CommandParser commands that are received from the client are received by the server are checked for correct formatting and then split in keyword and argument and then the CommandParser class calls the KeywordParser class that then in turn calls the relevant Command Handler. The Command Handler executes the commands according to how they have been defined in the Protocol.

4 Game

4.1 GameController

The GameController holds the General Information about the game including which users are playing or in which state (WAITING, RUNNING or FINISHED) the game is in. The actual information is stored in the World.

4.2 GameGUI and Game Engine

We separated the all the classes for the game into two packages: engine and gamegui. The gamegui classes are run on the server and synchronously on every client. The gamegui is solely for displaying the current gamestate in the engine package and for the user input (mouse and key listeners). Therefore the gamegui classes are only used on the client side. The users input is sent from the gamegui classes directly to the server, which updates it's own gamestate in the game engine and then sends the input onto the clients. Every game state changing action in the game (i.e. character movement or ending a turn) is processed this way. Actions which are not game state changing (i.e. checking the shooting range of a Child's weapon) are handled directly in the gui of a client and are not sent to the server.

5 Package Hierarchy

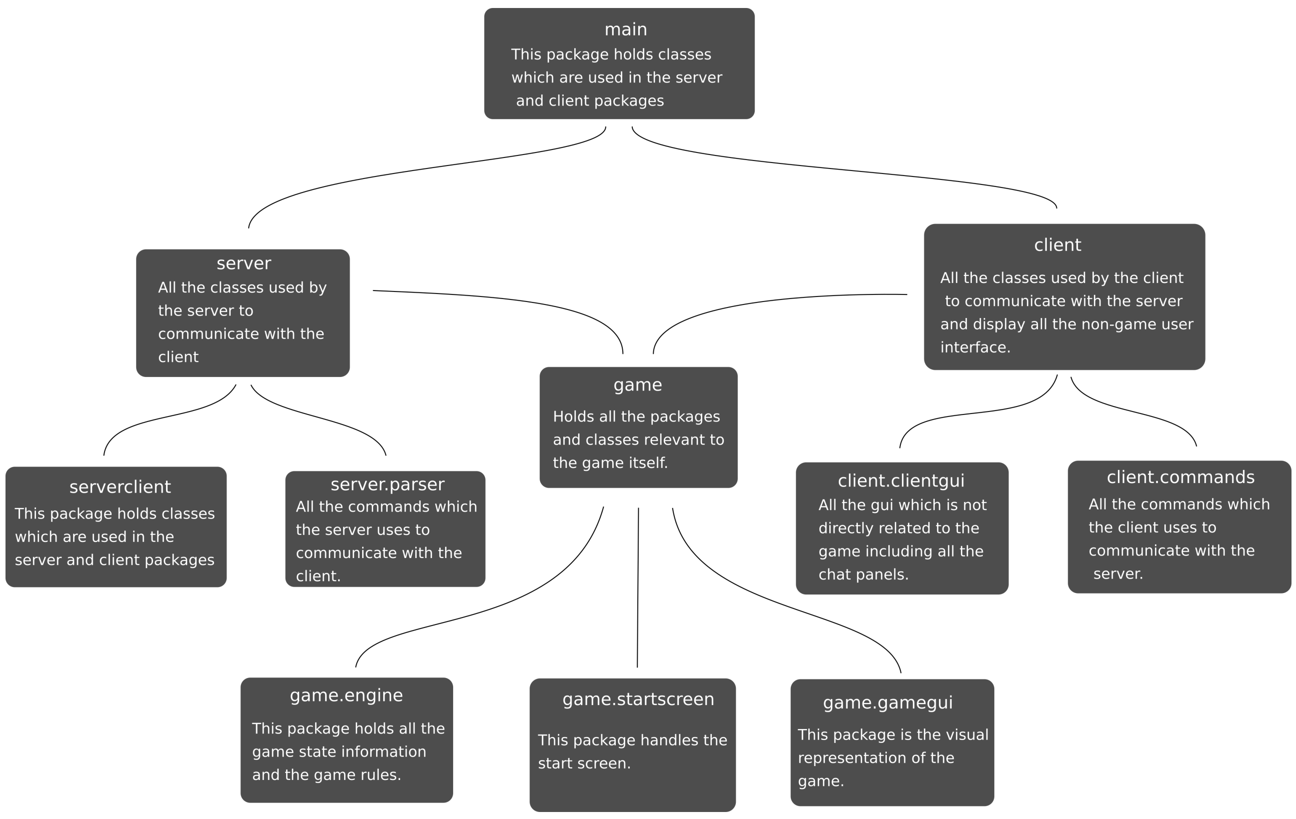
main
This package holds classes which are used in the server and client packages
server
All the classes used by the server to communicate with the client
client
All the classes used by the client to communicate with the server and display all the non-game user interface.
game
Holds all the packages and classes relevant to the game itself.
serverclient
This package holds classes which are used in the server and client packages
server.parser
All the commands which the server uses to communicate with the client.
client.clientgui
All the gui which is not directly related to the game including all the chat panels.
client.commands
All the commands which the client uses to communicate with the server.
game.engine
This package holds all the game state information and the game rules.
game.startscreen
This package handles the start screen.
game.gamegui
This package is the visual representation of the game.

Figure 1: Overview of the package hierarchy used in the game