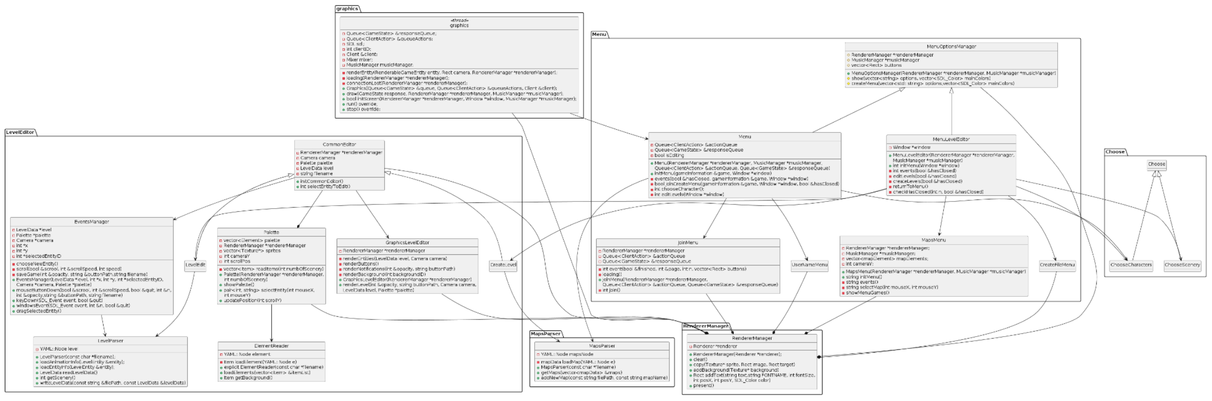
Jazz Jackrabbit 2
Client Technical Documentation

Contents

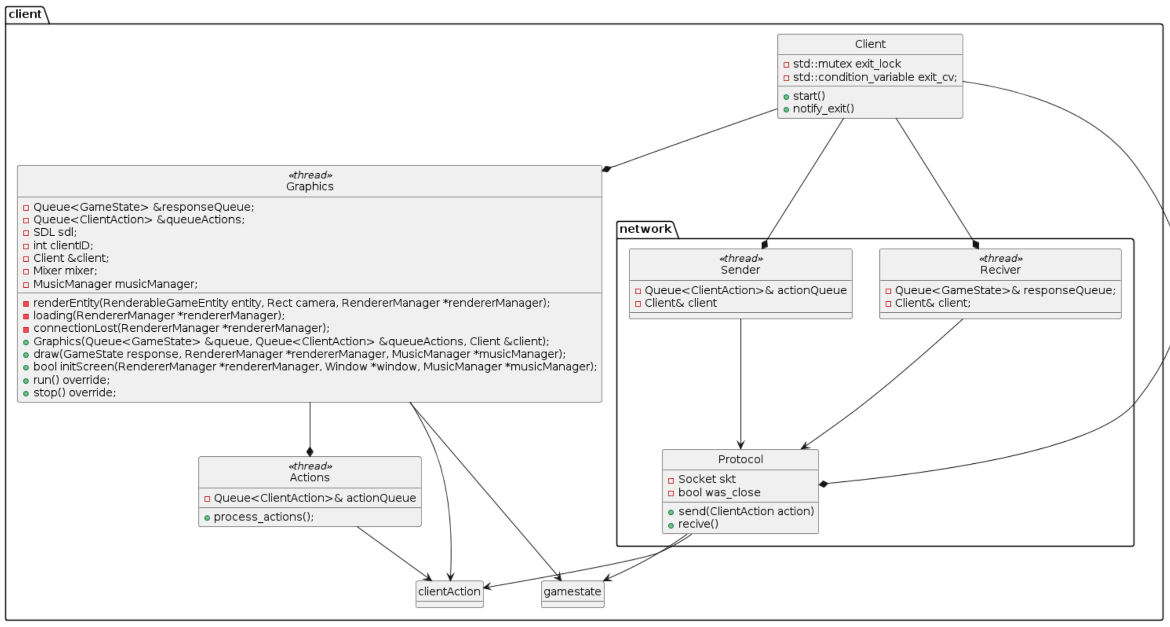
UML diagrams 3
File format and structure 4
MenuOptionsManager 5
Menu 5
MenuLevelEditor 5
MapsMenu 5
JoinMenu 5
UserNameMenu 5
CommonEditor 6
EventsManager 6
GraphicsLevelEditor 6
CreateFileMenu 6
RenderManager 6
Graphics 6
Actions 6
Sender 6
Receiver 7
Protocol 7

UML diagrams

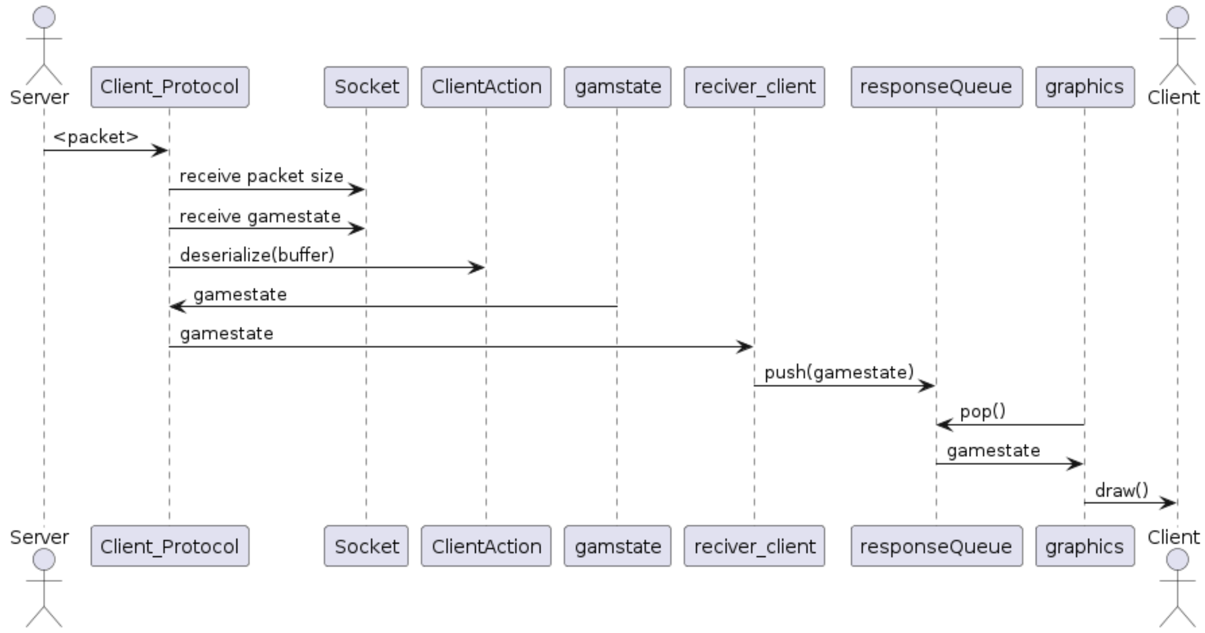
In the following class diagram, the principal classes of the level editor and the menu are depicted, showcasing how they interact with each other.



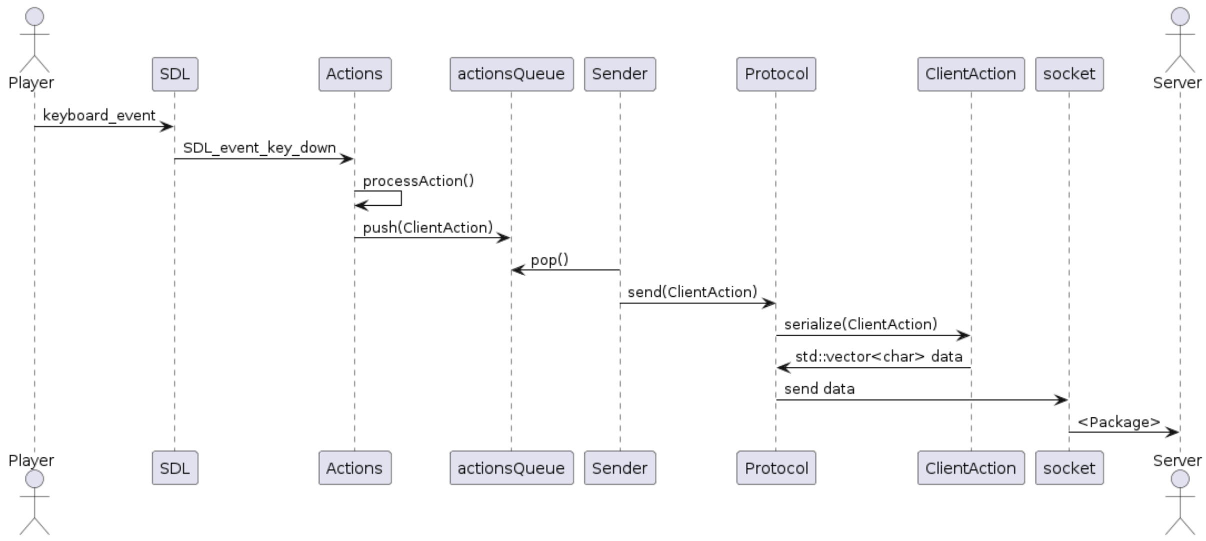
The Following diagram depicts the relationships between the client and various threads it uses to function.


client
Client
std::mutex exit_lock
std::condition_variable exit_cv;
start()
notify_exit()
«thread» Graphics
Queue<GameState> &responseQueue;
Queue<ClientAction> &queueActions;
SDL sdl;
int clientID;
Client &client;
Mixer mixer;
MusicManager musicManager;
renderEntity(RenderableGameEntity entity, Rect camera, RendererManager *rendererManager);
loading(RendererManager *rendererManager);
connectionLost(RendererManager *rendererManager);
Graphics(Queue<GameState> &queue, Queue<ClientAction> &queueActions, Client &client);
draw(GameState response, RendererManager *rendererManager, MusicManager *musicManager);
bool initScreen(RendererManager *rendererManager, Window *window, MusicManager *musicManager);
run() override;
stop() override;
network
«thread» Sender
Queue<ClientAction>& actionQueue
Client& client
«thread» Reciver
Queue<GameState>& responseQueue;
Client& client;
«thread» Actions
Queue<ClientAction>& actionQueue
process_actions();
Protocol
Socket skt
bool was_close
send(ClientAction action)
recive()
clientAction
gamestate

The following diagram showcases the flow of events during the processing of a response from the server:



The following diagram shows the flow of events corresponding the processing of the input done by the player



File format and structure

The YAML file format has been employed to define the structure of each implemented file. As previously detailed in the server documentation, the structure for level files has already been described.

In this section, we will elaborate on the file structure utilized for accessing maps designed for gameplay or editing purposes, as well as the structure employed for the elements in the level editor palette.

For the first case, the file maps.yaml is structured as follows:

The key "maps" represents a sequence of mappings, where each item in the sequence (indicated by -) is a mapping containing:

-path: A relative path pointing to the respective YAML file that contains map data.

-name: The assigned name for each map.

Regarding the latter scenario, we have files related to the palettes named "beachLevelItems.yaml" or "aiwLevelItems.yaml". Both files share the following structure:

They include the key "background", which defines the background of the scenario. Additionally, "elements" is a key that represents a sequence of mappings, each one containing:

-id: Represents the unique identifier associated with each element.

-type: Denotes the name that specifies the type of entity represented.

MenuOptionsManager

The class MenuOptionsManager is responsible for displaying and creating the level editor and main menu.

Menu

The Menu class provides the user with the option to either play or edit levels.

MenuLevelEditor

The class MenuLevelEditor provides the option to create or edit levels.

MapsMenu

The class MapsMenu allows for selecting a map.

JoinMenu

The class JoinMenu allows for displaying options to enter an existing game.

UserNameMenu

The class UserNameMenu allows for requesting the player's name for the game.

CommonEditor

The class CommonEditor enables all common actions for the level editor, for both created and edited levels.

EventsManager

The class EventsManager is responsible for managing all events of the level editor, such as selecting an entity, dragging and dropping it, saving the current state of the level, closing the window, etc.

GraphicsLevelEditor

The class GraphicsLevelEditor is responsible for rendering the graphics of the level.

CreateFileMenu

The CreateFileMenu class allows assigning a name to the map and the corresponding file when creating a new map using the level editor.

RenderManager

The class RenderManager is dedicated to handling everything related to the rendering of the graphical part.

Graphics

The class Graphics facilitates the initialization of the main menu, and if the user decides to start a game, it triggers the ActionHandler thread, communicates with the server, and renders the current game state.

Actions

The class Action, or ActionHandler thread, takes care of processing the keyboard inputs done by the player to then send it to the server.

Sender

The class Sender takes care of sending the actions done to the player towards the server

Receiver

The class Receiver takes care of receiving the response of the server and sending it to the client

Protocol

The class Protocol translates the packages received through the socket and takes care of sending the clients actions through the socket