|  |
| --- |
|  |
| Valepaska |
| An online multiplayer card game |

|  |
| --- |
| Juho Rantala  1-27-2022 |

Version history

|  |  |  |
| --- | --- | --- |
| Changes | Date | Author |
| Created documentation | 25.1.2022 | Juho Rantala |
| Added Group, Lobby and Game | 27.1.2022 | Juho Rantala |
| Removed client managing on EventHandler and moved it to Groups.  Removed broadcast | 27.1.2022 | Juho Rantala |
|  |  |  |
|  |  |  |
|  |  |  |

Contents

[Server side 3](#_Toc96103758)

[Server 4](#_Toc96103759)

[Public Methods 4](#_Toc96103760)

[Protected Methods 4](#_Toc96103761)

[Private Methods 4](#_Toc96103762)

[Handler (interface) 5](#_Toc96103763)

[Public Methods 5](#_Toc96103764)

[EventHandler 6](#_Toc96103765)

[Public Methods 6](#_Toc96103766)

[Protected Methods 6](#_Toc96103767)

[Private Methods 6](#_Toc96103768)

[Member 7](#_Toc96103769)

[Public Methods 7](#_Toc96103770)

[Player 7](#_Toc96103771)

[Public Methods 7](#_Toc96103772)

[Group 8](#_Toc96103773)

[Public Methods 8](#_Toc96103774)

[Protected Methods 8](#_Toc96103775)

[Private Methods 8](#_Toc96103776)

[Lobby 9](#_Toc96103777)

[Public Methods 9](#_Toc96103778)

[Protected Methods 9](#_Toc96103779)

[Private Methods 9](#_Toc96103780)

[TurnOrder 10](#_Toc96103781)

[Public Methods 10](#_Toc96103782)

[Protected Methods 10](#_Toc96103783)

[Private Methods 10](#_Toc96103784)

[Game 11](#_Toc96103785)

[Public Methods 11](#_Toc96103786)

[Private Methods 11](#_Toc96103787)

[Client side 12](#_Toc96103788)

[Template 13](#_Toc96103789)

[<class> 13](#_Toc96103790)

[Public Methods 13](#_Toc96103791)

[Protected Methods 13](#_Toc96103792)

[Private Methods 13](#_Toc96103793)

# Server side

This section covers the documentation for the server side of Valepaska. The code is in **./server/** and the project is defined in **server.pro**.

Server side is responsible to allocate new clients to their lobbies, start and handle their games and removing them after they are finished.

## Server

Class: Server

Files: **server.h** and **server.cpp**

A TCP server which communicates with clients.

This is an abstract class, which is meant to be derived from.

### Public Methods

|  |  |
| --- | --- |
| **Method** | **Explanation** |
| Server(*const* std::string& port = "12345") | Initialize the server. By default, uses port 12345. Creates a listening socket |
| *virtual* ~*Server*() | Closes all sockets |

### Protected Methods

|  |  |
| --- | --- |
| **Method** | **Explanation** |
| bool sendToClient(SOCKET& client, *const* std::string& msg) | Send a string ***msg*** to socket ***client***. Return false if an error occurred |
| bool broadcast(*const* std::string& msg) | Send a string ***msg***to all connected clients. Return false if any failed, but try to send to each client |
| void acceptClients() | Start main loop accepting clients to connect. This loop will run as long as the server is on |
| *virtual* bool *handleEvent*(Event& event)=0 | Pure virtual method to handle event ***event.***  Must be implemented by deriving class |
| *virtual* bool *addClient*(SOCKET client) | Virtual method to add a new client ***client.*** This method will call Server::handle(client) on a detached thread.  Return true if client was added. If the client already existed, return false |
| *virtual* bool *removeClient*(SOCKET client) | Virtual method to remove client ***client***.  If client doesn’t exist return false |

### Private Methods

|  |  |
| --- | --- |
| **Method** | **Explanation** |
| **void closeConnection(SOCKET client)** | Closes connection with client ***client*** |
| **bool hasClient(SOCKET& client)** | Return true if Server has client ***client*** |
| **void stopListen()** | Closes listening socket |
| **void handle(SOCKET client)** | Receive messages from client ***client*** and call Server::handleEvent for each message. Removes client if receive fails |

## Handler (interface)

Class: Handler

Files: **handler.h** and **handler.cpp**

An interface declaring the required methods for a Handler on server side.

### Public Methods

|  |  |
| --- | --- |
| **Method** | **Explanation** |
| *virtual* void *deckEvent*(Card card)=0; | Handle event when someone plays from deck |
| *virtual* void *print*(std::string message)=0; | Output message |
| *virtual* void *pendingDiscard*(discardID id)=0; | Handle discarding cardstack |
| *virtual* void *newClaim*(int rank, int amount)=0; | Handle new claim |

## EventHandler

Class: EventHandler : protected Server, public Handler

Files: **eventhandler.h** and **eventhandler.cpp**

A class to handle server events and manage lobbies. Derives from Server class and Handler interface.

### Public Methods

|  |  |
| --- | --- |
| **Method** | **Explanation** |
| EventHandler() | Initialize EventHandler. Create latest lobby |
| ~*EventHandler*() | Delete all lobbies |
| void **start**(Lobby\* lobby) | Call EventHandler::toGame if there are enough members in ***lobby*** |

### Protected Methods

|  |  |
| --- | --- |
| **Method** | **Explanation** |
| *virtual* bool *handleEvent*(Event& event) *override* | Passes event to its handler. Return false if event is unknown |
| *virtual* bool *addClient*(SOCKET client) *override* | Overrides Server::addClient. Return false if Server::addClient returns false. Otherwise add ***client*** to latest lobby and create new if needed |
| *virtual* bool *removeClient*(SOCKET client) *override* | Overrides Server::removeClient. Return false if Server::removeClient returns false. Removes ***client*** from lobby and destroys the group if it’s empty. |

### Private Methods

|  |  |
| --- | --- |
| **Method** | **Explanation** |
| bool sendEvent(Event& event) | Return false if event command is unknown.  Send ***event*** to the client specified in ***event***. |
| bool isHandler(command command) | Return true if command is valid to handle |
| bool isGenerator(command command) | Return true if command is valid to send |
| void createNewLobby() | Creates new empty lobby |
| Game \*toGame(Lobby\* lobby) | Transfer clients from ***lobby*** to new Game and delete ***lobby***. Returns the new Game |
| Group \*getGroupByClient(SOCKET client) | Return pointer to the Group where ***client*** is. If not found, return nullptr |

## Member

Class: Member

Files: **member.h** and **member.cpp**

A class to represent a member

### Public Methods

|  |  |
| --- | --- |
| **Method** | **Explanation** |
| Member(id id); | Create a member with given ID |
| *virtual* ~*Member*(); | No functionality |
| id getId(); | Return ID |

## Player

Class: Player : public Member

Files: **player.h** and **player.cpp**

A class to represent a Player in game

### Public Methods

|  |  |
| --- | --- |
| **Method** | **Explanation** |
| Player(id id); | Initialize Member |
| ~*Player*(); | Delete player’s hand (cards) |
| void add(cards cards); | Add all cards |
| void remove(cards cards); | Remove all cards |
| bool hasCards(cards cards); | Return true if player has all cards |
| int getCardCount(); | Return number of cards in hand |
| void print(); | Print player ID and cards |
| void clear(); | Clear hand |

## Group

Class: Group

Files: **group.h** and **group.cpp**

A class to handle a group of members

### Public Methods

|  |  |
| --- | --- |
| **Method** | **Explanation** |
| *virtual* bool *add*(id member) | Adds ***member*** if not already in group. Return false if already in group |
| *virtual* bool *remove*(id member) | Removes ***member*** if in group. Return false if not in group |
| *virtual* bool *hasMember*(id member) | Return true if ***member*** is in group |
| std::vector<id> getIds() | Return all player IDs |
| int size() | Return group size |
| bool **isFull**() | Return true if there are MAX\_SIZE amount of members |
| bool **isEmpty**() | Return true if there are no members |
| *virtual* void *transferTo*(Group\* other) | Add all members to other group |

### Protected Methods

|  |  |
| --- | --- |
| **Method** | **Explanation** |
| *virtual* bool *add*(Member\* member); | Add existing member |
| *virtual* bool *remove*(Member\* member); | Remove given member |
| *virtual* bool *hasMember*(Member\* member); | Return true if pointer refers to a member in group |
| *virtual* Member \**getMember*(id id); | Return member with given ID |
| *virtual* std::vector<Member\*> *getMembers*(); | Return vector of Members |

### Private Methods

|  |  |
| --- | --- |
| **Method** | **Explanation** |

## Lobby

Class: Lobby : public Group

Files: **lobby.h** and **lobby.cpp**

A group that keeps track on ready-state of members

### Public Methods

|  |  |
| --- | --- |
| **Method** | **Explanation** |
| Lobby(EventHandler\* eventHandler) | Link ***eventhandler*** |
| void setState(id member, bool isReady) | Change ***member*** state to ***state***. Do nothing if ***member*** not in Lobby |
| bool *add*(id member) *override* | Return false if Group::add returns false. Insert new ***member*** with ready-state false. |

### Protected Methods

|  |  |
| --- | --- |
| **Method** | **Explanation** |

### Private Methods

|  |  |
| --- | --- |
| **Method** | **Explanation** |
| bool **isReady**() | Return true if all members are ready |
| void **signalReady**() | Calls eventHandler::start |

## TurnOrder

Class: TurnOrder : public Group

Files: **turnorder.h** and **turnorder.cpp**

A class to keep track of turn in a group

### Public Methods

|  |  |
| --- | --- |
| **Method** | **Explanation** |
| TurnOrder(); | Initialize class |
| ~*TurnOrder*(); | Calls deleteOrder |
| *virtual* bool *add*(id id) *override*; | Add to order if Group::add was successful |
| *virtual* bool *add*(Member\* member) *override*; | Add to order if Group::add was successful |
| *virtual* bool *remove*(id id) *override*; | Remove from order if Group::remove was successful |
| *virtual* bool *remove*(Member\* member) *override*; | Remove from order if Group::remove was successful |
| Member\* next(); | Change to next turn. Remove new member in turn |
| void shuffle(); | Randomize the order |
| Member \*getTurn(); | Return member in turn |
| std::vector<Member\*> getMembersInOrder(); | Return all members in an ordered vector where the order starts from member in turn |
| bool isInTurn(Member\* member); | Return true if member is in turn |
| bool isInTurn(id id); | Return true if member with given id is in turn |
| Member\* turnTo(Member\* member); | Change turn until member is in turn. Throws a TurnException if member is not in order |
| Member\* turnTo(id id); | Calls overloaded turnTo with a member with given ID |

### Protected Methods

|  |  |
| --- | --- |
| **Method** | **Explanation** |
| void addToOrder(Member\* member); | Add member to order |
| void removeFromOrder(Member\* member); | Remove member from order |
| bool hasMemberInOrder(Member\* member); | Return true if member is in order |

### Private Methods

|  |  |
| --- | --- |
| **Method** | **Explanation** |
| void deleteOrder(); | Deletes order, but doesn’t delete members |

## Game

Class: Game : public TurnOrder

Files: **game.h** and **game.cpp**

Class to handle the gameplay. Inherits from TurnOrder

### Public Methods

|  |  |
| --- | --- |
| **Method** | **Explanation** |
| Game(Handler\* handler); | Link handler, create empty deck and cardstack |
| ~*Game*(); | Delete deck and cardstack |
| bool *add*(id id) *override*; | Add new player with given ID |
| bool *add*(Member\* member) *override*; | Add new player with given member’s ID |
| Player\* getPlayer(id id); | Return player with given ID |
| std::vector<Player\*> getPlayers(); | Return a vector of all players |
| void initGame(); | Remove cards from all, generate new deck and shuffle it. Draw cards to all players |
| void print(); | Print game state |
| bool play(id player, cards cards, int claimRank); | Call overloaded play with player with given ID |
| bool play(Player\* player, cards cards, int claimRank); | Return true if player was allowed to play given cards and claim them with given claim. This method might invoke handler’s pendingDiscard. Actions are specified in the rules |
| void deckPlay(id player); | Call overloaded deckPlay with player with given ID |
| void deckPlay(Player\* player); | If player is allowed to play, invoke handler’s deckEvent with drawn card |
| bool suspect(id player); | Call overloaded suspect with player with given ID |
| bool suspect(Player\* player); | Return true if player was allowed to suspect. Actions are specified in the rules |
| bool discard(discardID id); | This method should be only called form handler after invoking pendingDiscard. Return true if cardstack was discarded |
| void checkWin(); | Checks if player in turn won. Actions are specified in the rules |

### Private Methods

|  |  |
| --- | --- |
| **Method** | **Explanation** |
| void draw(Player\* player); | Draw 1 card |
| void draw(Player\* player, int amount); | Player draws specified number of cards from deck if deck is not empty |
| void drawTo(Player\* player, int target); | Player draws cards until they have specified number of cards in hand or deck is empty |
| void takeAll(Player\* player); | Player takes all cards from cardstack |
| bool toDiscard(); | Return true if deck should be discarded |
| bool isValidPlay(cards cards, int claim); | Return true if cards can be claimed as claim |
| bool isValidClaim(int claim); | Return true if given claim can be played now |

# Client side

# Template

## <class>

Class: <class>

Files: **<class>.h** and **<class>.cpp**

<description>

### Public Methods

|  |  |
| --- | --- |
| **Method** | **Explanation** |

### Protected Methods

|  |  |
| --- | --- |
| **Method** | **Explanation** |

### Private Methods

|  |  |
| --- | --- |
| **Method** | **Explanation** |