**Pikkdama – documentation**

We have two main activities both client and server side:

* **ConnectActivity:** It handles the connection of the devices, plus at server side you can select the players and their sequence
* **GameActivity:** The actual game takes place here, at client side players play cards and at server side the server organizes the rounds keeps the scores, and shows cards currently in play

Both server side and client side we have a class handling with communications (**ServerCom, ClientCom**). Here are the exact variables, functions and subclasses:

**ClientCom and ServerCom:**

* **ClientCom(ConnectionActivity), ClientCom(GameActivty) or on server side ServerCom(ConnectionActivity), ServerCom(GameActivty):** Two constructors
* **SocektSendingThread (extends Thread):** Handles socket sending to other devices.  
  Constructor: SocketSendingThread(String destIP, int socketClientPort, String msg)  
  SocketSendingThread.run() starts the thread, e.g. sends the socket.
* **SocketListeningThread (extends Thread):** Listens to other devices, always running in the backgorund. Once it gets a message, it passes through parseReceivedMessage(String gotMsg, String clientIP) where clientIP is the IP of the origin of the message.
* **Void parseReceivedMessage(String gotMsg, String clientIP):** Analyses the incoming messages and takes appropriate actions. In the function we have a huge if statement. It checks weather the ClientCom is created from a connectionActivity or from a GameActivity. In this if statement we have numerous if statements for the different messages, for more see communication protocol.
* **String getIPAddress():** Returns the IP address of the device (e.g. 192.168.0.89)
* **String getSubIP(String ip):** Returns the sub ip of the input ip address (e.g. 192.168.0.89 -> 192.168.0.)
* **connectActivity and gameActivity:** Instances of ConnectActivity and GameActivity from which the instance of the class has been created
* **final int clientReceivingPort, clientSendingPort:** Constanses for the client side ports, it sends the messages to the serer at sending port, and it listens communication (via a SocketListeningThread) at receiving port. At server side these constants are strored in ConnectActivity and GameActivity
* **Boolean running:** If it turns false, SocketListeningThread stops listening. It is needed for the proper destruction of an instance of the class.

We have some minor classes both client side and server side:

**Card (both client and server):** It represents a card.

* **String type:** It stores both the suit and the value of the card in the form of X\_YY where X is the suit-code and YY is the value:  
  The possible values of X:  
  - 1: Hearts (kőr)  
  - 2: Diamonds (káró)  
  - 3: Spades (pikk)  
  - 4: Clubs (treff)  
  The values of YY goes from 02 -> 14 (e.g. 2, 3, …, 10, J, Q, K, A)
* **Int colour:** The suit of the card, the value of X (see above).
* **Int value:** The value of the card, YY (see above).
* **String name:** Natural name (e.g. 3\_12 -> Spades Q)
* **Card(String s):** Constructor from the form X\_YY
* **Card(int colour, int value):** Constructor from the suit and the value of the card
* **addName():** Adds natural name to the variable name

**CardAdapter (currently only client side):** Helps creating text-based UI, only a temporary solution. It is for the creation of a ListView with Card elements. It uses R.layout.card\_list as a template.

**Player** **(only server side):** It stores the information of the players. When a client connects the server, a corresponding player instance is created.

* **Int position:** The position of the player in the game. A positive value will be assigned when the game starts.
* **Int score:** The current score of the player. It updates after every game.
* **Final String ip:** The IP address of the player’s device.
* **Final String playerName:** The name of the player (it is chosen when the client connects to the server)

**PlayerAdapter (only server side):** Pretty much like CardAdapter, helps create a ListView with Player elements. It uses R.layout.list\_item\_player as a template. Only a temporary solution.

**Game flow, ConnectionActivity:**

* Server starts listening (via SocketLsiteningThread)
* Clients enter a name and an IP search starts (in ConnectActivity. buttonConnectClicked), where the client sends messages (“NAME.” + <OWNNAME>) to everyone in subip range (e.g. own IP: 192.168.0.78, message sent to everyone in 192.168.0.0 -> 192.168.0.255). After every message, there is a 50ms sleep in order to slow down network traffic and avoid junction.
* When the server receives “NAME.XXX” from a client, it gets detected in ServerCom.parseReceivedMessage. A new player is registered using ConnectionActivity.putToList(Player newPlayer) function. It gets added to players list and the UI refreshes, showing the newly-come player as well.
* When on the server startGame button is clicked, the startGameOnClick function gets called which checks for the validity of player selection (while adding position to players in ArrayList), then a new Intent is created to fire up a GameActivity activity. It also puts the info of players into extras (name and IP). A “START” message is also sent to every player. Finally the serverCom.closingServerSocket is called, which stops listening. (Technically it sets running to false so after the next message SocketListeningThread exits infinite loop. This extra message is sent by clients right after receiving “START”.)
* After client received “START” it replies with a random message to help shuts down the server’s SocketListeningThread. Then an intent is created to fire up GameActivity, serverIP, ownName, Sending- and ReceivingPort is handed over via extra. Finally the ClientCom is shut down, setting running to false. (This runs on the same thread as SocketListeningThread, so after that this can be broken out of infinite loop, no need for waiting to a message from server.)

//TODO: folyamatábra

**Game flow, GameActivity:**

* After GameActivity is started on server and all the info of players is gathered from intents, the dealer() function is called. It sends the number of the current game to the players in the form of “NUMBER.X”. It creates 52 instances of Card (corresponding to an entire deck) and assigns a random float to each of them. Then the cards get sorted, then dealt. Each card is sent to the client with the message “DEAL.X\_YY”. Then giving() function is called, which determines whether giving takes place.
* Client gets the number of current game and 13 cards. Upon receiving the 13th card, in the Clientom.parseReceivedMessage function the gameActivity.startRound() gets called, which determine whether giving takes place. If no startGame() gets called. If yes, isInGiving is set true, createListView gets called, which shows the 13 cards to the player. After this the user selects the cards, then clicks the button, firing onOkButtonClicked function, which checks for the validity of selection (e.g. 3 cards have been selected), then passes them to server in the form of “GIVING.X\_YY”.
* The server receives the cards, they get detected in ServerCom.parseReceivedMessage. Upon receiving 12 cards, everything gets reset and sendGivingCards() is called. It determines the receivers and sends back the cards in the form of “GIVING.X\_YY”. (Technically not the cards but the messages from players gets stored in String givingCards[4][3] (e.g. “GIVING.X\_YY”), and the server sends back these messages.)
* When the client receives the given cards (in ClientCom.parseReceivedMessage), it adds them to their own (gameActivity.ownCards). If all 3 have been received the gameActivity.startGame() is called which checks for the Clubs 2. It is found, it notifies the server in the form of “CLUBS2”.
* The server receives “CLUBS2”, sets gameActivity.startingPlayer to the index of sending player (via using the playerFromIp(String ip) function), then calls gameActivity.game() which sets isPlaying true, hasBeenHearts false. Then it calls startTurn(true) (where true means it’s the first turn), which sends “CALL.CLUBS2” to the calling player. From now on, every turn is started via startTurn() function, only which alwas notifies the calling player with “CALL.”. This message can get two modifiers: “CALL.CLUBS2”, which means it’s the first turn, you have to call Clubs 2, or “CALL.HEARTS”, which means at least one card with Hearts has been played, so you can call a card with Hearts.
* When the client receives “CALL.(XXX)”, gameActivity.yourCall(Boolean clubs2, Boolean hasBeenHearts) is called. It alerts the player to play a card and sets canPlayCard true, so the user can select a card to play. When the user pushes the ok button, the onOkButtonClicked() function fires, and the isInGame branch gets executed. It checks whether you can call this card (using the youCall branch of canPlayThisCard(Card card) function). If the user’s selection is valid, the card is removed from ownCards and the playCard(Card card) function is called, which sends the card to the server in the form of “PLAY.X\_YY”. After that the UI is refreshed via using the createListView() function.
* When the server receives a card, the gameActivity.playedACard(Card card) function is called. The card is added to currCards ListArray<Card>, which are the cards played in this turn and showsUI() function is called to update the server’s UI (//TODO). If it wasn’t the 4th, the next player is alerted via the alertPlayer() function. It sends a message to the next client in the form of “PLAY.X” where X is the suit (colour) of the first card.
* When the client receives “PLAY.X”, gameActivity.yourPlay(int colourOfCall) is called, passing the suit of the first card as argument. The function yourPlay sets canPlayCard true, so the user can select a card by clicking the button, which fires the isInGame branch of onOkButtonClicked. It checks the validity of selection via canPlayThisCard(Card card), where the non-youCall branch is executed. If the selection was valid, the playCard(Card card) function is called, which sends the card to the server in the form of “PLAY.X\_YY”.
* If the server receives all 4 cards in a turn, then endTurn() is called, which counts the points, determine who got them and adds them to the temporary int currPoints[4] array. The server notifies the client to start the next turn, using startTurn(false).