VERSION 1.1
DATE: 05/01/2020

OBJECT-ORIENTED PROGRAMMING PROJECT REPORT

MONOPOLY EXTRA PLUSUBTRACT

MEMBERS: NGUYỄN VĂN ĐẠT
VÕ ANH VIỆT
TRẦN TẤN TÀI
NGUYỄN ĐỨC ANH TÀI

OBJECT-ORIENTED PROGRAMMING PROJECT REPORT

<<Building a game that get ideas from board game and game console system >>

OVERVIEW

This is a report about our Monopoly project. With many efforts, we only can develop a simple Monopoly remake version which certainly is removed many complex features. In this report, we will introduce our game through 5 sections:

I) GAME RULES AND FEATURES

Brieftly, Monopoly is a game, in which the player' s goal is to remain financially solvent while forcing opponents into bankruptcy by using and developing pieces of property. Each player will roll a dice by turn and go following the total number they have on the dice. Our game is a "gift" for a group of 4 friends. In the first version, 4 is the number of players which our game supports.

Initially, we must say that this game needs 5 devices to run. The game will run on a computer and four mobiles for each player to handle actions in game. A 4-digit of pin must be provided by mobile clients to make sure that you do not come into wrong room. Then, each player will choose their favorite character. Do not worry, we will not let you choose the same character as your friends. After that, our game will random the order of 4 players and start the game's flow.

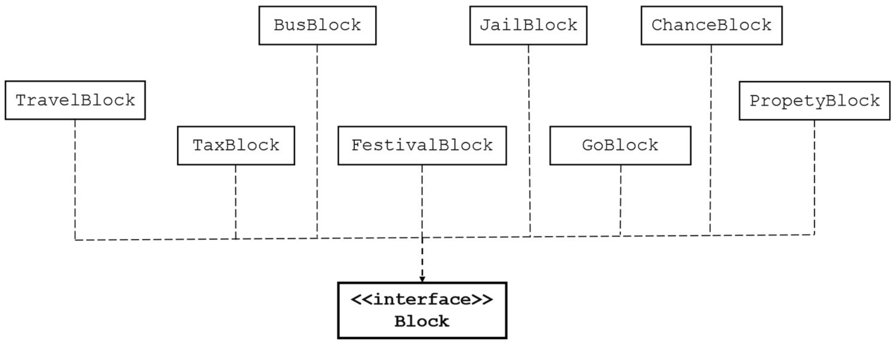
Monopoly board game contains 32 blocks with some special blocks. Each type of block has different functions. Go block is your first position and you can receive fixed money when each time you go through it (in case you are not going to jail). Jail block is the place which you are imprisoned, and you must roll a lucky number or pay money to go out. In festival block, you can have a chance to organize festival and receive a double rent fee in the block you choose. Bus block is the best destination for you to test your dignity because you will be moved to a random block in board in the next turn. Chance block also does the same thing. You may be received or charged money as well as go to some special blocks. Property block and Travel block are worthwhile properties to buy and own, you can sell them in dangerous circumstances. If you go to Tax block, you must pay the bank 10% of your total assets (include your property and travel plus money).

In game, when your turn comes, the action will depend on your position in board. If you are not in some special blocks such as jail block or bus block, you can have a chance to roll a dice. Your character will jump to the next block which is equal to your position plus the value in the dice. Especially, we have the feature called "lucky" numbers – 1 and 6. If your dice have the same value with the lucky number, you can continue go in the next turn. However, you must be careful! If 1 or 6 appears three times consecutively, it means that you are treating, and you must go to jail block. We hope that you do not encounter this situation that is you may go to a block being bought by another player. Moreover, you do not have enough money in hand to pay them rent fee. Bankruptcy? No, we create a chance for you to sell your property to pay money. If this thing cannot help you, we must sorry that you are loser. All your money and assets will be substituted for your debt. The last player who survives will be our winner.

II) JAVA CORE GAME

BLOCK BOARD

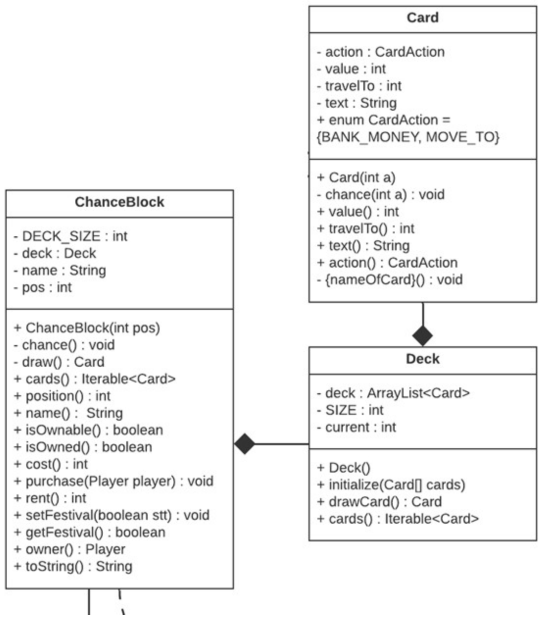
Firstly, monopoly's game board is a 32-block board. Each block has unique attributes as well as different methods. In this list of blocks, some of them which are Go, Jail, Festival, Bus, Chance, and Tax are special blocks. However, we realize that all of them have some same methods such that `name()`, `position()`, `cost()`, etc. So, interface class `Block` is created as a template for classes `TravelBlock`, `TaxBlock`, `FestivalBlock`, `GoBlock`, `PropertyBlock`, `BusBlock`, `JailBlock`, and `ChanceBlock` to implement. Then, some of necessary attributes and methods in each block also will also be declared for features to work. (picture 1)



Picture 1. All blocks are created based on Block interface

CHANCE BLOCK

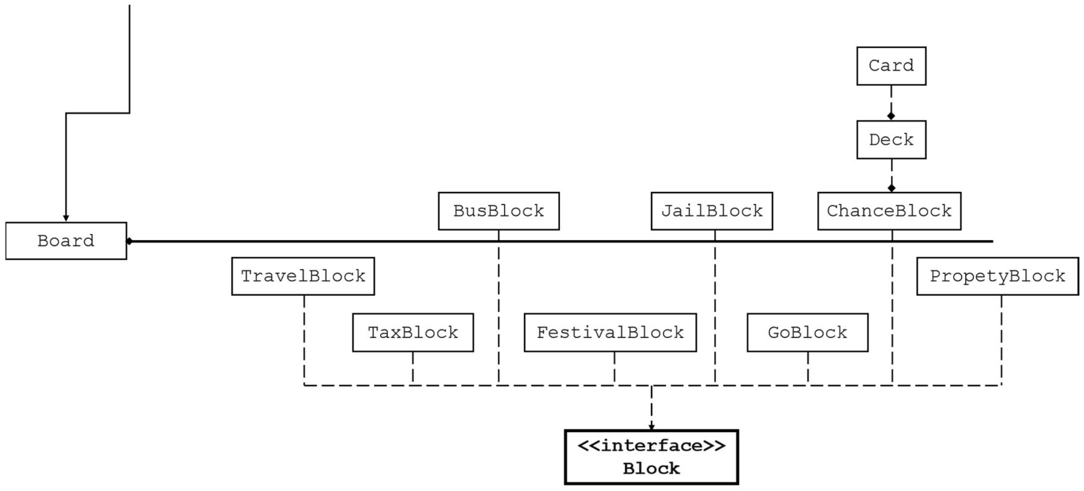
In ChanceBlock, now it is only an "empty" block, we will add a deck of cards for it. Class `Deck` will be the "container" of class `Card`. Meanwhile, class `Card` contains many different chance cards which bring many surprise things to lucky players. When players go into the chance block, they can have ability to draw a card in this deck. In `ChanceBlock`, its constructor will create an object of deck and call the method `chance()`. `chance()` will create an array of object Card and pass them to this deck through the method - `deck.initialize(cards)`.Then, when player go into, they will use the method `draw()`, it will return a card took from method `deck.drawCard()`. (picture 2)



Picture 2. ChanceBlock contains Deck and Deck is the place to hold Card

BOARD CLASS

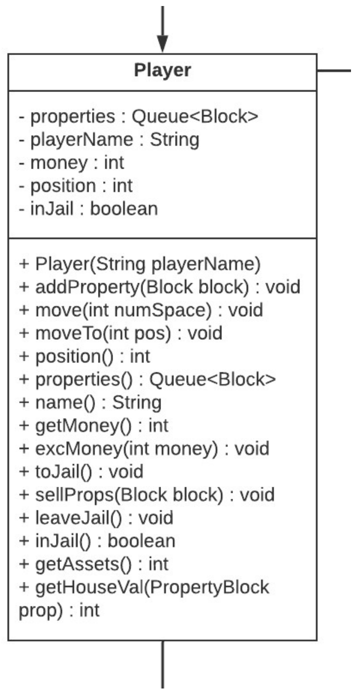
We want a "big" class to control all of "small" blocks. Therefore, we aggregate all them to `Board` class. `Board` class is used to create all 32 objects of blocks for game board through method `makeBlock(int pos)`. In game, TravelBlock is a property in which players cannot build houses, they only try to own them as much as possible because rent price will increase by formula: 200 * (number of `TravelBlock` you own). So, `makeTravel()` will group all other travel blocks to each group for each object of `TravelBlock`. We can count how many travel blocks which is owned by player thanks to this method. (picture 3). This is very difficult for us to update the information of each block when we want to change name, price, or position. So, we save all information in file **config.ini**. Then, Board class will use library ini4j to read this ini file whenever starting to initialize.



Picture 3. Board handle all "small" blocks

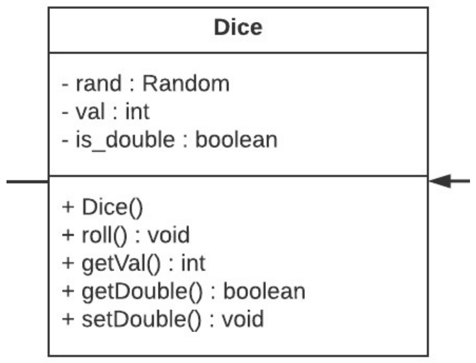
PLAYER CLASS

In a game, players are the thing which cannot be lost. Player in monopoly needs some attributes like name, money, some assets, position, as well as some methods such as `move()`, `moveTo()`, `excMoney()`, `addProperty()`, etc. A full Player class's features which is initialized by us show bellow. (picture 4)


Player
- properties : Queue<Block>
- playerName : String
- money : int
- position : int
- inJail : boolean
+ Player(String playerName)
+ addProperty(Block block) : void
+ move(int numSpace) : void
+ moveTo(int pos) : void
+ position() : int
+ properties() : Queue<Block>
+ name() : String
+ getMoney() : int
+ excMoney(int money) : void
+ toJail() : void
+ sellProps(Block block) : void
+ leaveJail() : void
+ inJail() : boolean
+ getAssets() : int
+ getHouseVal(PropertyBlock prop) : int

Picture 4. Player class

DICE CLASS



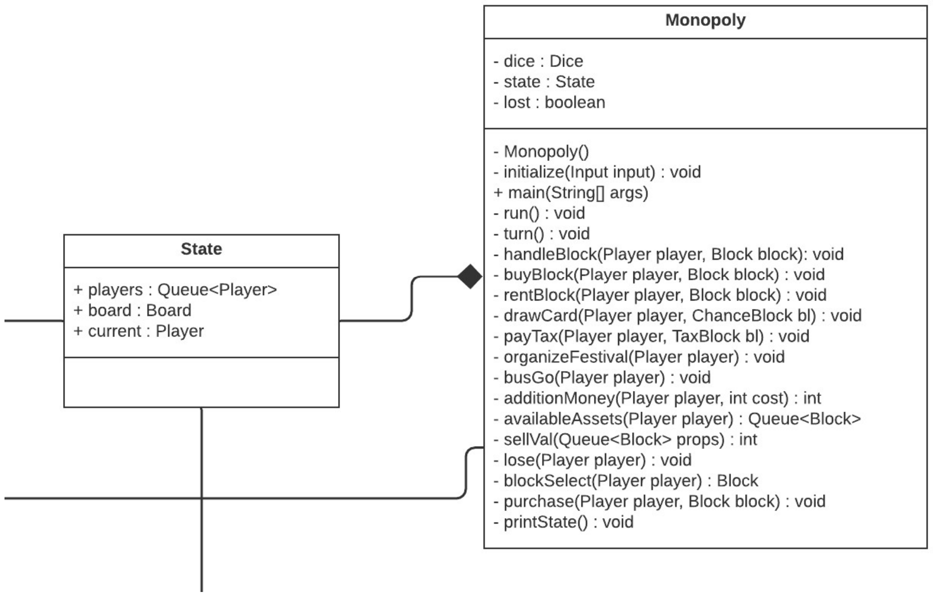
Picture 5. Dice class for player roll the dice

Next, it is apparent that player will roll a dice in their turn. So, class `Dice` will play a main role for this feature. The private attribute rand will be the `Random` type and its object will be created in `Dice`'s constructor. Two remain private attributes: val - saving total of dices' value, is_double – checking whether two values are duplicated or not. `Roll()` method will assign value to these variables. (picture 5)

MONOPOLY CLASS

Finally, main class for our game is `Monopoly` class. Class `State` which is an essential class contain list of players in game, a game's board, and player playing at that time. The process of our game will run in order: `Monopoly() –> initialize()` (input player, random the order of players' turn) `-> main() -> -> run()` (loop until have winner) -> turn() (One player plays each turn) `-> printState() ->` have winner - `END`

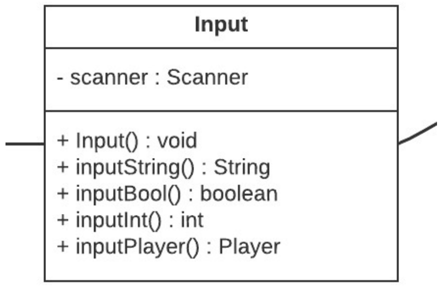
In `turn()`: `dice() -> handleBlock()` (player will go and operate the feature in destination) `-> buyBlock() / rentBlock() / payTax() / toJail() / organizeFestival() / busGo()` (we also add some additional abilities in each circumstance which can occurs – `additionMoney(), purchase(), sellVal()`, etc). (picture6)


Monopoly
- dice : Dice
- state : State
- lost : boolean
- Monopoly()
- initialize(Input input) : void
+ main(String[] args)
- run() : void
- turn() : void
- handleBlock(Player player, Block block): void
- buyBlock(Player player, Block block) : void
- rentBlock(Player player, Block block) : void
- drawCard(Player player, ChanceBlock bl) : void
- payTax(Player player, TaxBlock bl) : void
- organizeFestival(Player player) : void
- busGo(Player player) : void
- additionMoney(Player player, int cost) : int
- availableAssets(Player player) : Queue<Block>
- sellVal(Queue<Block> props) : int
- lose(Player player) : void
- blockSelect(Player player) : Block
- purchase(Player player, Block block) : void
- printState() : void
State
+ players : Queue<Player>
+ board : Board
+ current : Player

Picture 6. Monopoly - the "heart" of game

INPUT CLASS

For convenience, we also have class Input to handle all the input in game. For instance, `inputBool()` for player input their decision (`yes`, `no`) to buy house, dice, etc, `inputInt()` for player input the number of houses they want to build, and `inputPlayer` for player input their name in `Monopoly.initialize()`.



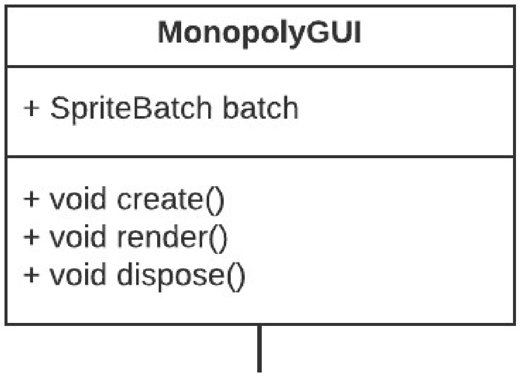
Picture 7. Input class to handle all input in game

Full class diagram is provided in Report folder in our github repo.

III) JAVA GUI

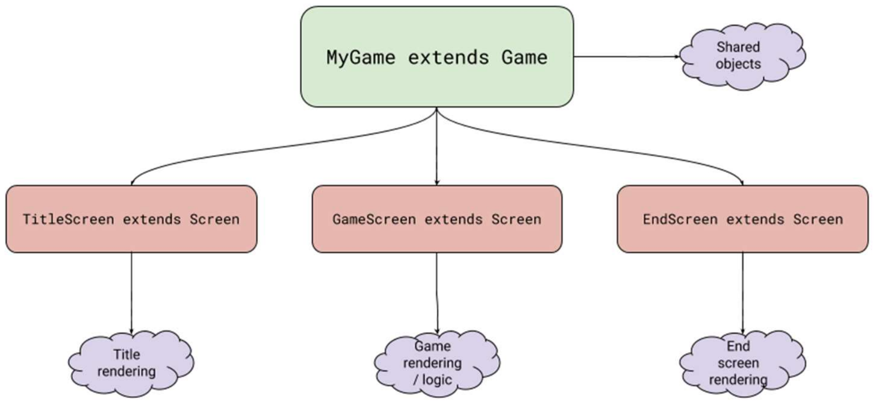
MONOPOLYGUI CLASS

For graphical user interface, Monopoly Extra Plusubstact use famework LibGDX to handle. Our game contains 3 main screens which are WelcomeScreen, WaitingScreen and MonopolyPlay. All of them will be controlled by MonopolyGUI class (Picture 8).



Picture 8. MonopolyGUI class

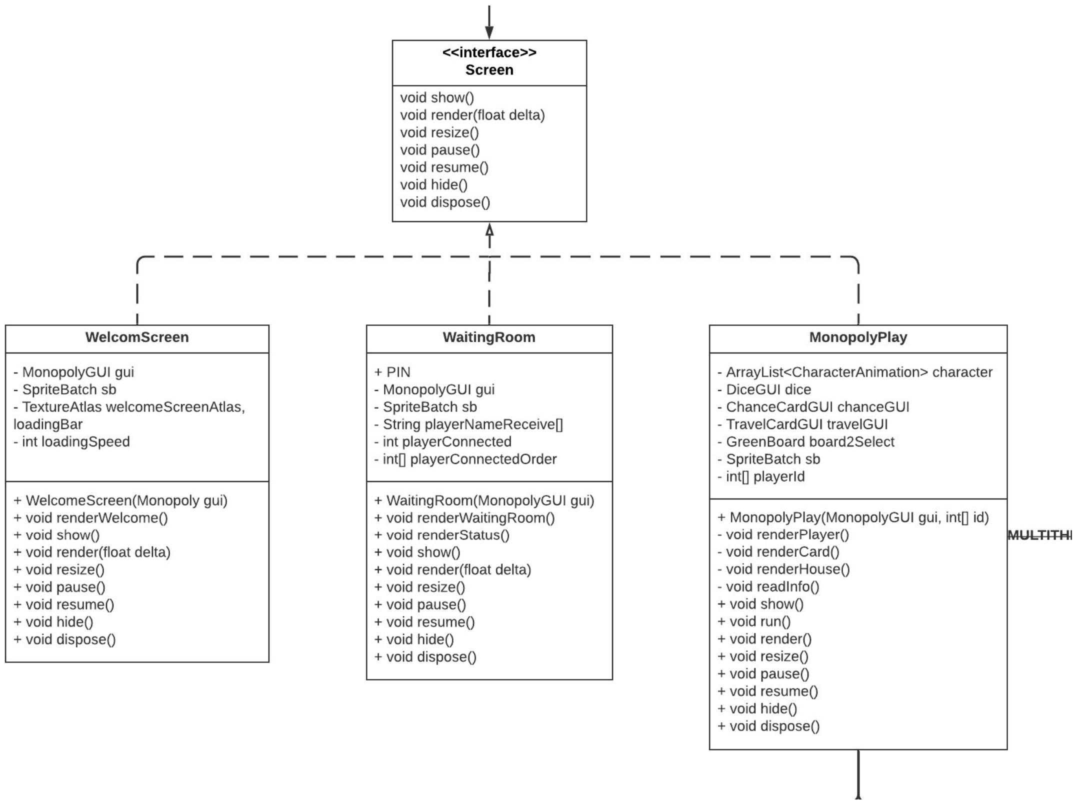
To be more specific, this class extends Game class of LibGDX and controls Screen class by the model below:



Picture 9. LibGDX Game-Screen model

WELCOMESCREEN, WAITINGROOM, AND MONOPOLYPLAY CLASSES

WelcomeScreen, WaitingRoom and MonopolyPlay are our screens which are all implements the interface LibGDX's class Screen



Picture 10. Screens class in game

In Screen class, `show()` method will be called when the screen appears for the first time. Then, `render()` will execute continuously until we change to a new screen. When this screen is not displayed, method `hide()` will do its work. In `dispose()`, we need to dispose things game constantly need like textures, music, sounds or assetManager which holds libGDX objects. This action will free all the resources which are kept in memory.

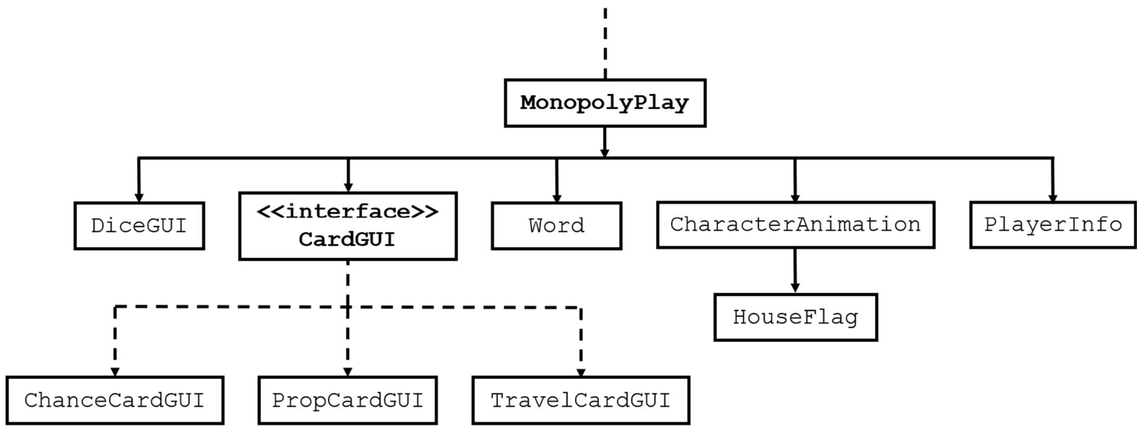
Welcome screen will render a loading bar, and our game's logo. When the bar full, Welcome screen will hide and WaitingRoom appear. At that time, mobile app will connect to our game and choose character. If we receive enough four players, you need to press the button play game. Monopoly game screen will appear and start run the game logic as well as GUI game.

COMPONENTS IN MONOPOLY GAME

We divided our game screen into small components which are `DiceGUI`, `CardGUI` (include `ChanceCard`, `PropertyCard`, `TravelCard`), `CharacterAnimation`, `GameBoard`, `Word` and the place to display `PlayerInfo`.

1. `DiceGUI`: the class handle the dice's image.
2. `CardGUI`: this is an interface for 3 types of card to implement.
3. `CharacterAnimation`: the class renders the position of character as well as their status. In this class, we contain a subclass called `HouseFlag` which is render player's assets such as properties, and house built in them.

4. `GameBoard`: the monopoly game board include 32 blocks and its background.
5. `Word`: render all the words type on screen.
6. `PlayerInfo`: we design 4 player avatar and their money displayed at four edges of game screen.



Picture11. discribe

IV) REACT NATIVE – MOBILE CLIENT

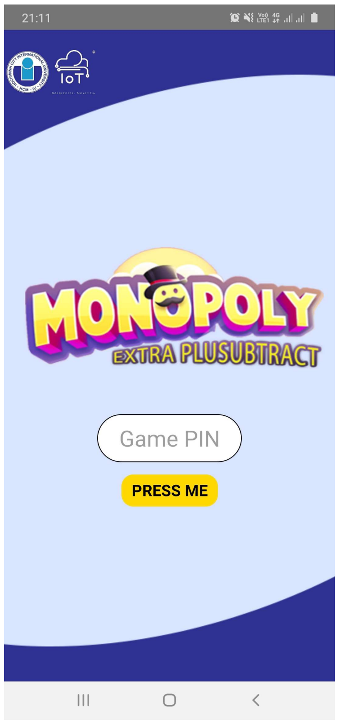
HANDLE INPUT

In many of computer game, player must input value or parameter from keyboard, but the original goal of MEP is building a system like game console. Player can input any thing from another device, not the computer keyboard. So, smart phone is the best choice because of these devices are very popular nowadays.

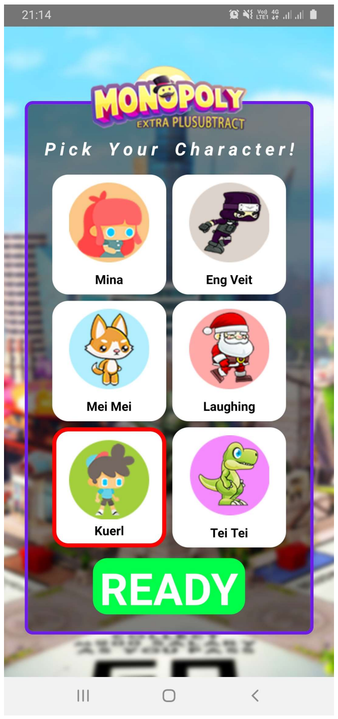
REACT NATIVE

React Native is a JavaScript framework that allow us to build any mobile app in Android or iOS (in the first version of this project, we just build it in Android because of some limitations of this framework). There are some of other framework or programming language that they can program a mobile app, but our team choose because this language is easy to approach with us. Flutter is also a good choice that time, but it will take much time to study it.

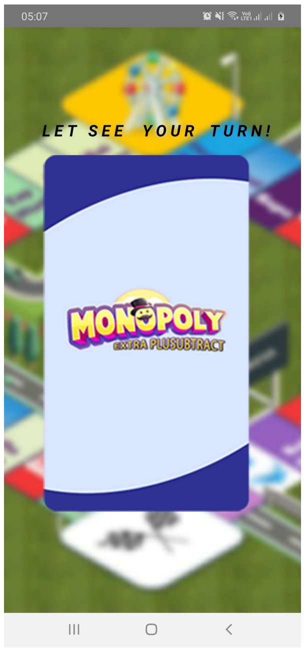
IMPLEMENT MOBILE CLIENT



Picture11. Login screen



Picture12. Choose character screen



Picture11. Get the turn and every input screen

V) MQTT PROTOCOL

COMMUNICATION BETWEEN THE GAME AND MOBILE CLIENT

Follow the original goal of Monopoly Extra Plusubtract (MEP), four players must control the game by a mobile client. So, the information will be transferred between mobile and java game by a real-time way and they will not use any cable for user experience.

At first, socketio used to handle this problem because this is one of the best ways to build a real-time task (original MEP protocol [3]) (more information about socketio [4]). Despite of what socketio can do, it also has a weakness. Socketio built by the Client – Server architecture. So, an intermediate server is required. Managing topic for sending something from message between five clients will not be visual.

Mqtt become the most effective alternative this time. MQTT means Message Queue Telemetry Transport, is an OASIS standard messaging protocol for the Internet of Things (IoT). It is designed as an extremely lightweight publish/subscribe messaging transport that is ideal for connecting remote devices with a small code footprint and minimal network bandwidth. MQTT today is used in a wide variety of industries, such as automotive, manufacturing, telecommunications, oil, and gas, etc [5]. Using publish/subscribe architecture through broker [6] and level topic system solve two of socketio problem at the same time.

PAHO MQTT (JAVA) AND MQTTJS (REACT NATIVE) LIBRARY

In MEP game client, Esclipse Paho Mqtt is used and Mqttjs with mobile clients (more information [7]).

TOPIC DESIGN

Mobile Client	Game Client	Notes
onConnect		PIN: confirm room
	onConnect/PIN	1: confirm client connected
onConnect/playerId		playerId: choose character
	PIN/onConnect/order	Order: order of turn
PIN/gameplayM/playerId/...		Confirm block feature
	PIN/gameplayP/playerId/...	Ask for confirm block feature
PIN/gameplayM/playerId/select		Select on bus/festival
	PIN/gameplayP/playerId/select	Confirm select on bus/festival

VI) REFERENCES

1. https://happycoding.io/tutorials/libgdx/game-screens#
2. https://libgdx.com/

3. https://github.com/Kuerl/socketio_demo_mobile_javaapp.git
4. https://socket.IO
5. https://mqtt.org
6. https://mphcmiuedu-my.sharepoint.com/:p:/g/personal/ititiu19047_student_hcmiu_edu_vn/EXCzgBkT24VOqqj8rDG8TwoBFuDnZNn9ghtoZQzGUCEmMg?e=4eoYGm
7. https://github.com/Kuerl/mqtt_protocol_demo_mobile_javaapp.git