

Software Development (202001064)

Programming (202001066)

Programming Report – UNO

Muhammad Rafi Albab, s-2424894, muhammadrafialbab@student.utwente.nl

Miguel Beleza Leong Seixas e Sousa, s-2551586,
m.belexaleongseixasesousa@student.utwente.nl

January/2023

Overall Design

1.1. Class Diagrams

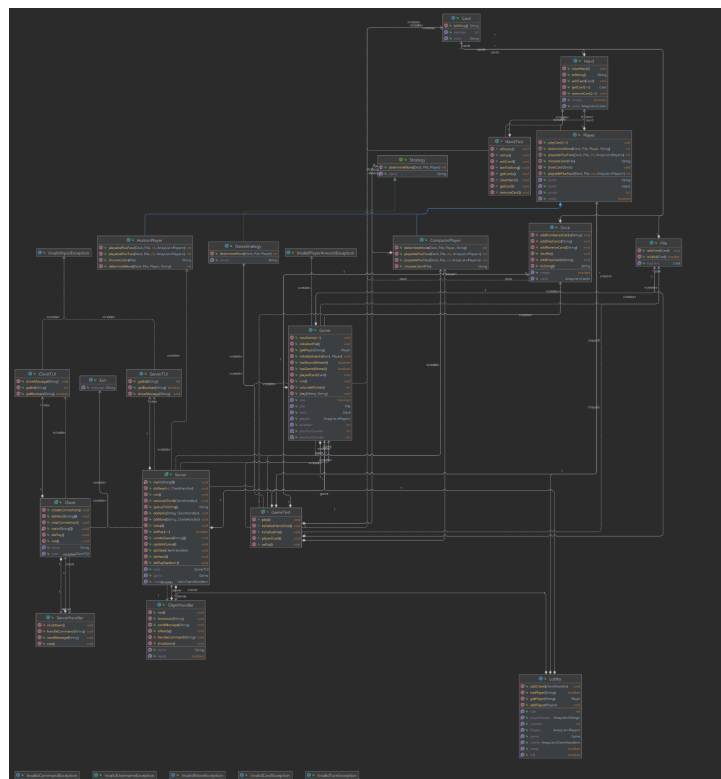


Figure 1: Class Diagram for UNO game

In Figure 1 above the class diagram of the UNO game is shown. The class diagram contains the methods of each classes and its attributes. The class diagram above shows the complete interactions between each classes. The most relevant classes from the class diagram is game as it controls most of the game. Other classes are also important such as player, hand, deck, and pile. The game runs with a deck of cards that is distributed to a hand of each player and the cards of those player would be played into the pile. (We included the image file in the git repository so you can have a better look at the class diagram)

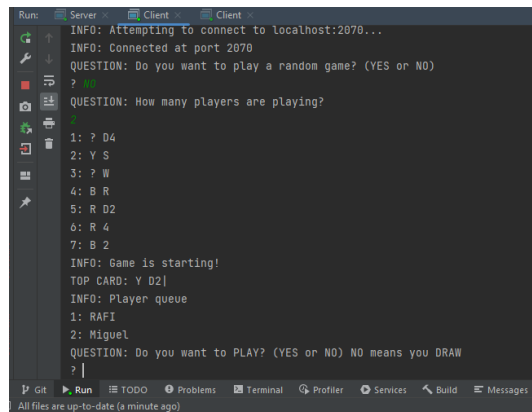
1.2. Requirements Implementation Map

The basic implementation of UNO should be able to play UNO with at least 2 people over a network with a client & server application, in our game this first implementation runs fine with some exceptions which will be talked about in the later part of this section. The second basic implantation is to be able to connect to a server, play a game and announce the winner in the end, this implementation is made and runs perfectly. Third basic implementation is the server should be able to host at least one game with 2 players, following the rules of UNO, and determine the winner in the end. The third basic is also implemented correctly and will run as expected. The UI of the game is the fourth implementation. The game uses a TUI as a UI displays what the user needs to do whether it is connecting to the server or playing the game, it takes input from the user and throws exceptions when needed. The fifth basic implementation that we needed to create was a to make the program to be to make our program to be MVC structured application. Our program managed to implement Model and Controller to a good extent however view was not implemented very well which will be explained further in the Model-View-Controller Implementation part of the report. Exceptions are the sixth implementation that needs to be implemented as a basic implementation. Exceptions in the program are thrown and caught with respect to the error made by the user such as `InvalidInputException` and `InvalidPlayerAmount` exception. Computer player is implemented seamlessly in the local side of the program. It is also able to make moves with respect to progressive uno which will be talked about as the eight implementations. Locally the computer will be able to determine its own move and choose its own color when playing a wild card. Although the computer player works well in the local side of the program, when the Computer Player implementation was moved to the network with a client and server application it doesn't work as well. The computer player has massive issue when implemented with a network side. This issue is that the computer player will share the same hand as the other player when the other player is a human player. This error is quite odd because when having two human players this error doesn't happen. The computer player however can determine its own move like it should be even on when using a server instead of running it locally. The last implementation is to have at least one additional feature which was stated earlier was progressive UNO. This implementation is chosen because without it doesn't feel like an UNO game. We managed to perfectly implement the progressive UNO feature locally. The progressive UNO feature is able to forward plus two cards and plus four cards as expected. However, in the server side when the protocol is used the progressive UNO feature couldn't be implemented. The implementation was difficult when moving over to the protocol side. When this was being implemented too much of game must be changed to make this work in turn causing a lot of bugs making the program not being able to run.

1.3. Model-View-Controller Implementation

There are 6 packages present in our project, the controller, exceptions, model, server, test and util. The controller class was where most of the functionality relied. Here the OOP classes are present and their connections. The exceptions package contains all used exceptions, although some of the protocol exceptions were not fully implemented due to time constraints. The model contains our Game model, which is the heart of the whole project. All classes run through Game. The server package contains all server related files, including protocol implementation, server files and client files. The test package contains our JUnit tests that we used to test functionality and the package util contains the `TextIO` used for local functionality. Our project structure is not correct, since we did not incorporate a view package and did not create a Board class, which would be used as the mechanism to display all components of the game to the server. Instead, our Game class runs all displaying and viewing functionality, which ended up bringing a lot of difficulty when implementing the server, client and protocol.

1.4. User Interface



```
Run: Server Client Client
INFO: Attempting to connect to localhost:2070...
INFO: Connected at port 2070
QUESTION: Do you want to play a random game? (YES or NO)
?
QUESTION: How many players are playing?
?
1: ? D4
2: Y S
3: ? W
4: B R
5: R D2
6: R 4
7: B 2
INFO: Game is starting!
TOP CARD: Y D2
INFO: Player queue
1: RAFI
2: Miguel
QUESTION: Do you want to PLAY? (YES or NO) NO means you DRAW
?
|
```

Figure 2: Picture of The TUI Interface

The TUI of the project of a client is shown above in Figure 2. The TUI contains the cards and their indexes and will be the input for a move after the client enters YES to PLAY?. This TUI is made with respect to the protocol made for our mentor group. The top card is shown as can be seen from Figure 2 to be Y D2 which means yellow draw two and this top card is randomly generated from the deck. The card shown for index one in Figure 2 shows a (?) which means that is a wild card and D4 which means draw two. In index 3, there is a (?) but instead of a D4 it shows a W which means it is a normal wild card that is able to choose color. The rest for the normal cards the first letter represents the color, as an example Y would be yellow and the second would be the number and special cards such as R for reverse, S for skip, W for wild card, and D(number) for draw the number of cards.

Testing

Testing is an important part of the project, to test all the methods work properly and that the game will run accordingly a proper testing would need to be done. In the game a lot of methods are implemented to play the game and to progress within the game. For our testing the first test that is done is with a simple eye test, with respect to the TUI, the game is made sure to be running properly and the game can be ran locally. When the simple eye test is finished, the bugs are then tested with JUNIT test to test the methods that contribute to it. The classes that the JUNITs are made are for the following classes: Game, Hand, and Pile.

2.1 Unit Testing

The strategy of testing the classes is to test the most important classes that contributes to the logic of the whole game which are the classes as follows: Game, Hand, Pile, Player, and Deck. However testing player might be redundant because the player most prominent attribute besides it being a computer player or a human player, which would not be possible to be tested with a JUnit test is the hand of the player. Testing deck is a very similar to testing pile and when pile is tested and the game works accordingly deck would also follow to work accordingly. Therefore the tests for JUnits that are being run are the classes Game, Hand and Pile.

For the Hand class all the methods are tested. A new hand is created in the test class and all the methods are checked with a @BeforeEach which adds 3 cards into the hand. The first test would be to test the method getCards(). In this test getCards() should return an ArrayList of cards in the hand and this tested by asserting True when getCards() is called it should be an instance of an ArrayList of cards and another assertTrue that the size of the hand is 3. The next method to be tested is getCard(), this should get the card based on the index and to test if this works, getCard() is called with the corresponding index and get the color and number and asserting equal that the card color and number is the same as the indexed card based on when the card is added. Next method is addCard(), this method adds a card into the hand. To test this method a card is added to the hand, and it is asserted as equal to the index of the size of the hand -1 which is the last card of the hand. Next method is to check whether the hand is empty, isEmpty(). This test is straight forward and it asserts false that the hand created earlier is empty and creating a new

hand isEmpty() should be true. The next method to test the clearHand() method, this method is tested by asserting true of hand isEmpty() after clearHand() is called. Last test of the Hand class is testing its toString() method, to test the method this method tests with assertTrue that after calling the toString() method the string contains keyword of the card such as "R" for red and "9".

Some of the methods for game are tested, because some methods like run() would almost be impossible to test since it is basically running the whole game and the test would be the same as a visual inspection of the UI. First @BeforeEach is created to setUp() the game and this is done by adding players into the ArrayList of players. In game the method to be tested is initializeHands() which would initialize the hands of each player and the asserting Equals for the hand to be equal to 7 and that it is not empty. Second method to be tested is initializePile(), this would initialize the pile to have a random top card.

All the methods of the Pile class like the Hand class all methods are tested. For the pile tested no set up needs to be done, only creating a new pile inside the class is done. First method is asserting equals when getCards() is called and its size should be equals to 0 since a new pile should have 0 cards. Second method to be tested is the addCards() method. And to test this method a card is added using the method. First to test it is asserted the size of pile.getCards() to not be 0. Then check the index 0 of the card if the color of the card is the same as the color of the card added. Next method to be tested is getTopCard(), to test this method a new card is added and that card should be the top card. With this information it is asserted as equals the getTopCard() and the object that is added as top card. Last method of the class is isValid(). This method should test whether the card being played is an acceptable move. And to do this a new card is created using addCard(new Card("RED", 0)). Then a bunch of cards is tested with the isValid() method, cards would return true after calling isValid() statement are Red cards with appropriate numbers or a card with number 0 with appropriate Color meaning that the color and number should be part of the uno game (Example invalid color: "BLACK", "MAROON").

2.2 System Testing

The approach taken for the system testing was a procedural one, after every addition the system was tested for the added functionality. Firstly, after finishing the locally functioning game, we started by adding the client and server classes, the connection between server and multithreaded clients was then tested. All tests were done with a localhost server and local clients. We tried to run tests with multiple machines but were not able to connect them, so decided to do all testing locally. It was quite odd that we were incapable of connecting multiple computers to one, but we believe it to be because of an issue that will be mentioned later.

The initial tests of connecting multiple local clients to a localhost worked without any problems. The protocol was then implemented within the client, server and handlers. Here we encountered our first and biggest issue. Once the broadcasting method was added to the handlers and we tested it, instead of broadcasting an individual message to all clients, one client was receiving all broadcasts. This is where we wasted a lot of time, even with the help of TAs, nobody was able to figure out the issue behind the broadcast. We ended up researching for what the problem could be and looked at other examples of server and clients. We eventually noticed that our buffered reader and writer were static variables and since an individual client handler oversees the broadcast, the buffered writer out could not change between the different out connections and therefore not broadcast.

With a functional broadcast, we could move on to implementing and testing all the protocol commands. These were done one by one with 2 clients as human players. Here, the biggest complications came with changing the game class such that it worked with the server methods. Due to having fully developed a locally functioning game, there were a lot of while loops and scanner inputs which had to be removed and rerouted to the server.

Here is where we started developing a lot of difficulty debugging the code. It became increasingly harder to find the cause of errors and fixing one would just lead to another. We were able to correctly route human players and tested them by running 1v1 games but were incapable of routing the computer player fully. Although a computer player can be created and run, the hands of the players become duplicated when playing with a computer player. After a few days of testing, we were incapable of finding the cause of the issue and decided to move on to rerouting the incorporated feature (progressive uno).

We rerouted the code so that the server kept track of the plus two and four cards when placed, such that the total pluses were calculated when a client had to take. It was whilst testing the feature that we realized the way we had structured the implementation in the local functionality made it impossible to be implemented with the server and client. This was because the whole implementation was done through the play() method in Game. Unfortunately, this is as far as we were able to get. We successfully implemented and tested a game between two clients as human

players but were unable to successfully test all functionality of a computer player and also our features. However, on the local version of the game all functionality was successfully tested, although a few bugs are still present.

Metrics Report

Academic Skills Chapter (Student 1)

4.1 Time Management and Procrastination Avoidance

For this course week 1 to week 6 in terms of programming went well. We managed to do all the exercises needed to pass the course on time although the last week was slightly hectic. Toughest thing that we did during the whole 6 weeks was in my opinion was threading in week 6. From week 2 to week 4 in the last academic skill assignment I mentioned that I was doing well in terms of keeping up.

With regards to continuing to manage my time and adapt it became better sometimes and sometimes and it gets worse than how I managed my time in week 2 to week 4. I managed to manage time by setting my own goals per week on what has to be done. In some weeks I manage to complete that goal sometimes ahead of time and sometimes barely making it and sometime extending my goal to sometime around the week after. I thought this was a good start on keeping Time Management and Procrastination but it turns out maybe I should set a daily a goal instead of a weekly goal instead to be able to manage my time better as it was not very consistent.

How the project went in terms of time management and procrastination is not ideal. We finished the game logic and completed the additional feature and got everything working way ahead of time. However, with regards to implementing it with the client and server there was so many challenges for us. There was a case where we cannot proceed due to one slight mistake that took us 24 hours and three TA's including our mentor couldn't help. The slight mistake was something minor but I managed to figure out but maybe it was too late that our bufferedReader was not supposed to be static which made the out(bufferedReader) overwritten and therefore printing to the same client twice, although when debugging we found that they were running in different sockets.

4.2 Strengths and Weaknesses

My strength based of my Learning Journey during this module was collections. I was able to get through them relatively fast in time and I thoroughly enjoyed them. I was able to also do well on the other topics that are mostly related to OOP. During around week 6 and week 7 I found that my biggest weakness is in the threading topic. This is due to it being quite complicated for me although I do understand now. I think that the check point meeting went really well, I liked the result after those checkpoint meetings knowing what I have to do next and where to improve. But what I think was lacking was mentor meetings during the time on projects. Although we have it with our mentor because it was not a mandatory thing, it could be better structured and be mandatory so that more effort and it is better to prepare us for this project.

4.3 Checkpoint Meetings

In this course I had more than 2 checkpoint meetings with my mentor, I did this in order to keep track of my progress and have a second opinion on where my progress is currently at. With those meeting it helps me with the metacognition cycle of where I reflect and access on my progress and current abilities at that given time. I was also able to self-assess every two week and get feedback from my mentor based on my self-assessment during the mandatory meetings and also additional meetings that I had.

Academic Skills Chapter (Student 2)

5.1 Time Management and Procrastination Avoidance

As the module started, everything was going very well. During the second, third and fourth week, we met all deadlines on time and I did not feel too much pressure or stress to meet deadlines. Having done the first module and had some java programming in high school, I felt that the first few weeks were a lot less stressful than the later parts of the module, understandably so. As the weeks continued we managed to finish all programming and system design deadlines on time until week 5.

Week 5, because of the holidays, was when we fell a bit behind but were still able to complete week 5 and week 6 on. However, the first week of January we made sure to catch up with everything, so that we could meet the week 6 deadline. The project was going very well. In terms of time management, we finished the first and second deadline ahead of time and as exams and deadlines approached I put more and more time into everything in order to make sure we could calmly finish all our deadlines.

The project however turned out to need a lot more time than we expected. Although my time management is not the absolute best, I do think I have put the correct amount of time into everything, but ran into certain issues, which we could not surpass. There were errors which took us days to find the cause of, even some where TAs could not help out.

I do think we should have better used the help of TAs, there were some issues where a new pair of eyes could have easily helped and saved us a lot of time. But both Rafi and I can get stubborn and want to find the cause of an error ourselves. We should have better recognized when we needed help.

5.2 Strengths and Weaknesses

I came into this module with a lot of strengths regarding my programming. Having had java in high school for 2 years and had done module 1 calmly, I made it through quite easily during the first half of the module. I think my strengths definitely lie in the topics of week 1 through week 5, since I had already been exposed to them previously. As my learning journey went past week 5, I started learning some new topics and discovered a whole new side of java in networking. This is definitely where my weakness lies since I had never been exposed to the topics, I found it quite interesting how the IO streams work using the available java classes. However, I think I could have better used the checkpoint meetings and the opportunities provided by the course. I spent a lot of time just trying to learn and do things on my own, which I think ended up harming our success in the project.

5.3 Checkpoint Meetings

I think I could have used the checkpoint meeting much better. Although they went very well and I got great insight from them, I do not think I maximized the resources that were being provided. Overall I think my self-assessment over the course was good and that I correctly identified my strengths and weaknesses and how I could improve them. But I did not recognize how the mentors could have helped me improve even more. Both in terms of how I managed my time and stress but also the help they could have provided for assignments and the project.