**Groupwork: The 0re0**

The group comprises of Antoine Cattaneo, Brandon Filmer, Philip Kwedza and Tino Motsi. The roles initially assigned as follows:

Brandon: Card and Rubbish Class; GameFinal Class;

Phillip: Player Class; GameFinal Class;

Antoine: GameFinal Class;

Tino: Deck Class;

Tino made the Deck class which was in charge of the creation, shuffling and distribution of the cards mainly, with some other methods including the addition of Shannon entropy determining method to name one. The class was faulty and returning a lot of errors which Brandon and Philip then ironed out. Philip created Player class, which was in charge of handling the player’s cards, controls the decision-making and playing of the Computer player(AI) as well as keeping the individual’s players’ scores at the end of each round. Brandon then made the Card class which returns the different ranks of the card which the computer players/AI uses to make decisions, it also converts the card strings into images the user can see in command prompt; Brandon also made the Rubbish class which serves as the pile where the play happens, it also implements most of the rules set out by checking if the moves are valid. Antoine, Brandon and Philip then worked on the GameFinal class which essentially was the client program and linked all the individual classes, it also implemented a vast amount of the rules for the actual gameplay. We commented in each of the sets of code as to who did what and where exactly each person worked. We all divided the work equally and did the documentation.

**Extension of Background and Motivation: The 0re0**

In the group proposal we highlighted the history of the game and where it comes from, we also touched on some of the very basic rules of the game. In this extension we will shed light on these rules and the attributes given to each card.

Firstly, since the game has a ‘largest number out ruling’ each card has been assigned a specific ‘weighting’ which allows us to tally up the number each player has at the end of each round excluding the final. The weight is as follows: Joker = 50; ‘8’ of any suit = 40; ‘2’ of any suit = 25; ‘7’ of any suit = 14; an Ace of any suit = 11; the King ‘K’, Queen ‘Q’ and Jack ‘J’= 10; any other card’s weight is equal to the number on the card e.g. a ‘5’ of any suit = 5 or a ‘3’ of any suit = 3. We have also assigned each of the card ranks to the cards to try and make the game a bit more structural. An ‘8’ is the highest rank because of its ability to determine the suit of the pile. A Joker and ‘2’ are both ranked 2, as these are the picking cards. The King ‘K’, Jack ‘J’ and ‘7’ are all ranked 1 as these cards affect the players’ turn either in getting one extra turn or skipping another player’s turn. This will make it possible for the computer to play strategically to make the game as challenging and as realistic as possible.

During the gameplay some of the cards’ special attributes come in to play. Starting with the jokers, any of the Jokers make the following player pick up 5 cards from the deck, unless the player has an Ace which in this case will serve as a block or a 2 of any suit which then means the following player after that has to pick 5 (from joker) + 2 (from the 2 played). This process will be repeated until a player has an Ace or has picked the cards. With that being said, a 2 of any suit will make the following player pick 2 cards from the deck, but it can only be played on a joker or card of the same suit. An 8 of any suit can be used to declare a changed of suit of the pile, provided that it is played on a card of the same suit or another 8. A 7 skips the next player’s turn, provided it is played on the same suit or another 7. A king gives the player who plays it another turn to play and they have to play either another King or another card of the same suit. Lastly, the Jack ‘J’ reverses gameplay by one move, meaning the player before gets an extra turn and the game flows normally after that.

**Problem Statement**

We had to overcome certain obstacles in order to make the game as interactive, realistic and challenging as possible to the user, and to give the user as close to a real-life gaming experience as far as the GUI is concerned. The problems we faced included: creation different classes to handle the different aspects of the game i.e. the deck, the pile, the rules etc. Linking all these classes and implementing a high-quality GUI which will work effectively in order to deliver a good quality game the user can enjoy their experience.

**Approach**

This project implemented the version of the Crazy 8s game that we have outlined in the previous sections of the documentation. We have a few milestones as follows: Creating the individual classes namely: Deck class which will handle all the creating, shuffling and distributing function of the game; Player class which will take care of keeping track of each players’ cards, their moves and in the end how many points the player has; Cards class which basically works with the ranking and weighting of each card, this class keeps all this information and returns it when necessary and Rubbish (Pile) class, this is the class we created in order to keep some of the rules of the game play, but to make it manageable we only put about half of the rules in this class. The next milestone was creating the GameFinal class which then serves as the client program and also has the other half of the rules left over from the Rubbish (Pile) Class, therefore incorporating all the individual classes and making them into a functional game. The next step was to test out this game and to find and debug any kinks or anything that could prevent the game from running smoothly. The final step would have been to implement the high-quality GUI which would give our game that attractive look and this would open up the game to anyone with basic computer skills.

**Timeline of events**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Week 1: 23 - 29 September | Week 2: 30 September – 6 October | Week 3: 7 - 10 October |
| Card, Deck, Rubbish and Player class |  |  |  |
| GameFinal class |  |  |  |
| Testing and debugging |  |  |  |
| Implementing the GUI |  |  |  |
| Project Document |  |  |  |

**Software Development Life Cycle**

Throughout the journey of getting to the finalised product we used a system whereby we completed one stage, in this case one class, at a time. Not moving on to the next one until we had completed the one we were currently working on. After we had completed one stage of our program there was no going back with the exception of the final debugging stage. Each stage was heavily reliant on the information from the previous stage although we had one project plan for all the stages.

With that being said we adopted The Waterfall Model which is one of the oldest and easiest of the structured SDLC methodologies. How we did it was: We created the Deck class then went on to the Player class, then went on to the Card and Rubbish class. After that we went on to make the final game class which relied on the four individua classes working first before we could create it, we had to perform some debugging to the individual classes as well. We then went on to create the GameFinal class which we had to finish before we could even start looking at possible GUIs.

Diagrammatic representation of our process:

Deck Class

Player Class

Card Class

Rubbish Class

Debugging of all the individual Class

GameFinal Class

Testing the final game

Implementing the GUI

\*At each of the stage transitions represented by the arrows there is a form of debugging done.