**Project Deliverable 3 – Software Design and Project Review**

**System Overview**

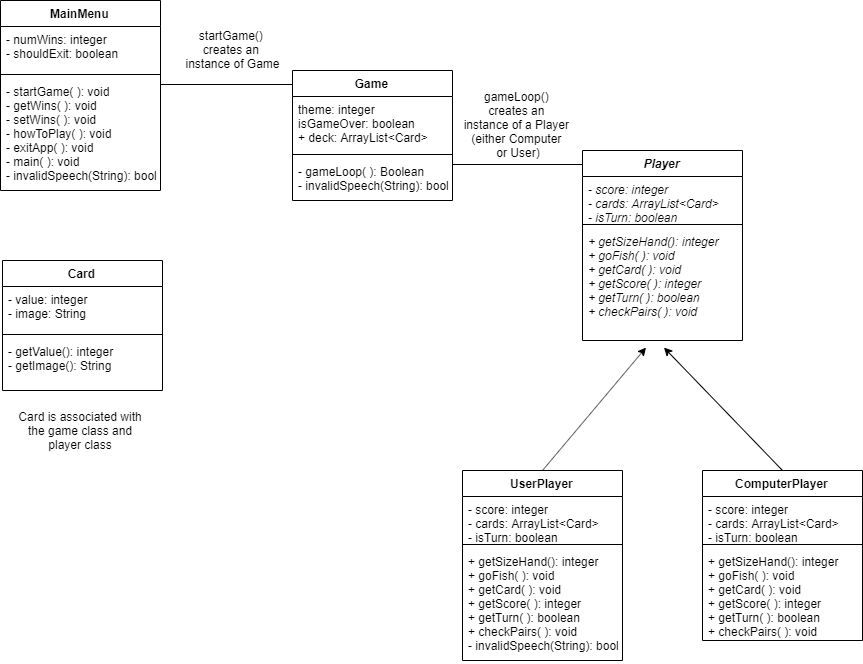
This version of Go Fish is designed for both kids and adults to enjoy on a mobile app on Android devices. The unique voice activation feature makes it possible to play the game hands-free. Different themes, such as animals, colors, and food, are included for an enhanced learning experience for kids.

**Technology**

The main technologies that will be used for the development of this system include, but are not limited to a Java development environment, as the game will be programmed in Java with a heavy foundation of object-oriented programming. The main environment we will be using is Android Studio, as the app is being developed from an android perspective. The specific organization of various files and code in our development process will be organized using GitHub. This includes version control for not only our code, but also any draw.IOs, code, and project deliverables. We are also using Paint.NET for mockup graphics of screen displays. Finally, as for the APIs required for the system, in order to deliver the specified requirements of voice recognition and computer voice response, two APIs will be needed: one for speech to text conversion and one for text to natural speech conversion. Tentatively, the two APIs being considered are Amazon’s *Polly* and Amazon’s *Transcribe*, both offered by AWS. The reason we have chosen this API is due to the ease of access and since GMU allows students to make a free AWS Educate account. *Polly* deals primarily with text to speech and *Transcribe* deals with speech to text. This last technology may be subject to change.

There are a few relevant assumptions to make note of. The first is that the user will need to have an Android device to use the app. A secondary assumption is that the app will receive the needed permission from the user to access the microphone for the user to be able to play the game through speech recognition.

Finally, there will be no changes to the requirements and the architectural styles based on the technology decisions made thus far.

**Class Diagrams/Descriptions**

|  |  |  |  |
| --- | --- | --- | --- |
| **ENUMS** | | | |
| <<enumeration>>  **Default** | <<enumeration>>  **Animals** | <<enumeration>>  **Colors** | <<enumeration>>  **Food** |
| ACE  TWO  THREE  FOUR  FIVE  SIX  SEVEN  EIGHT  NINE  TEN  JACK  QUEEN  KING | CAT  DOG  SNAKE  MONKEY  PIG  BEAR  SHARK  BIRD  DUCK  SPIDER  SHEEP  LIZARD  BEE | RED  ORANGE  YELLOW  GREEN  BLUE  PURPLE  PINK  BLACK  WHITE  GRAY  BROWN  RAINBOW  TAN/BEIGE | APPLE  BANANA  ORANGE  PEAR  WATERMELON  RASPBERRY  BLUEBERRY  COCONUT  POMEGRANATE  PINEAPPLE  STRAWBERRY  LEMON  CHERRY |

\* Each ENUM literal will be constructed with an associated string (e.g. “eight”) and integer value. These will be used to represent the different types of cards associated with each theme.

The MainMenu class includes the attributes and methods that will be used for the main menu of the game. The attributes include the number of total wins and a Boolean variable of if the menu, and eventually the app, should be exited. The methods include startGame(), which is called if the user picks the start option from the menu. The method takes an input for a theme from the user and creates an instance of the Game class and will call the Game class’s gameLoop() method. The getWins() method returns the number of total wins of all cumulative games. The setWins() method alters the number of total wins based on the result of the game. The howToPlay() method displays the rules of the game to the user. Finally, the exitApp() method exits out of the app.

The Card class includes the attributes and methods used by the game and player class (such as in the ArrayList of Cards, as discussed later). The class includes a getValue() method, which gets the type of card. The class also has a getImage() method, which is to get the image associated with the card for graphical purposes.

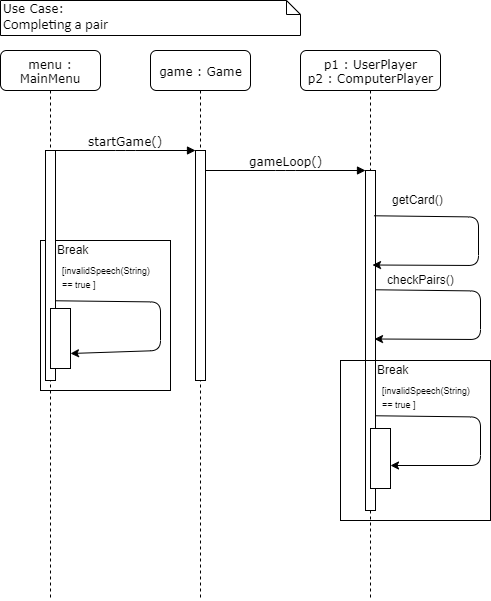
The Game class is the class that composes of the main game loop. Internally, it will keep track of which theme the user selected with an integer in order to correctly display the correct theme. The Game class also keeps track of the main deck, which is an ArrayList of cards, and includes a Boolean variable to keep track of whether the game is over. The actual gameLoop() method is the method that initializes the two players (computer and user) and loops through the game logic by making use of Player methods on the computer and user player objects. Essentially this will continue looping until the game is over. This is the same logic flow described by the P2 deliverable and will include calls to methods such as getCard( ), getScore( ) and more until the game is over. Throughout the loop, within each method call, the screen will change. After the loop ends, the screen will display who won, and the method call will end.

The player class is an abstract class, which has the subclasses of UserPlayer and ComputerPlayer, which both will be used as instances in the main game loop. An abstract class was chosen for Player, as some methods, such as getScore() will have the same implementation for both types of subclasses, however other methods such as getCard() will have unique implementations within each subclass. The player class includes the player score of each player, an array list of cards of the player’s hand, and a Boolean to check whose turn it is. The methods include getSizeHand(), which is needed to see if the game had ended, goFish() for when the opposing player does not have the card which is being asked, and getCard() for when the player needs to ask for a card (e.g. goFish() will be called in getCard() if opponent does not have desired card). The methods also include getScore(), which is called at the end to see who wins, getTurn() to check whose turn it is, and checkPairs() after each turn to see if a player has a pair.

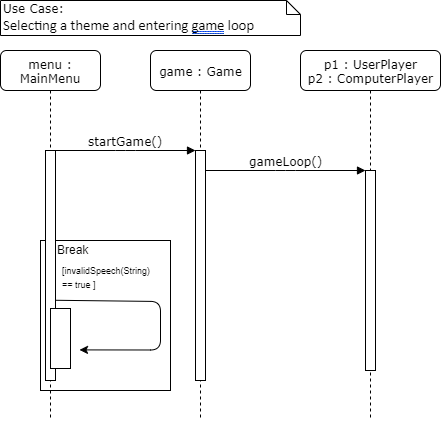
Certain classes will also have an invalidSpeech() method which returns a Boolean value if the user says a command that is invalid or asks for a card they do not have.

**Sequence Diagrams**

Below is a sequence diagram for completing a pair. This invokes methods such as Player.getCard() and Player.checkPair(). This also has a break/alternative result if the user asks for an invalid card



A second sequence diagram is used to demonstrate selecting a theme and entering the gameLoop. This invokes methods such as MainMenu.startGame() and Game.gameLoop(). This also has a break/alternative result if the user asks for an invalid card.



**User Experience**

Below are the user experience considerations for the system that have remained unchanged:

Usable – Voice commands and the menu must be simple enough for kids to use it. Help menu is included to give further information about the app to the user.

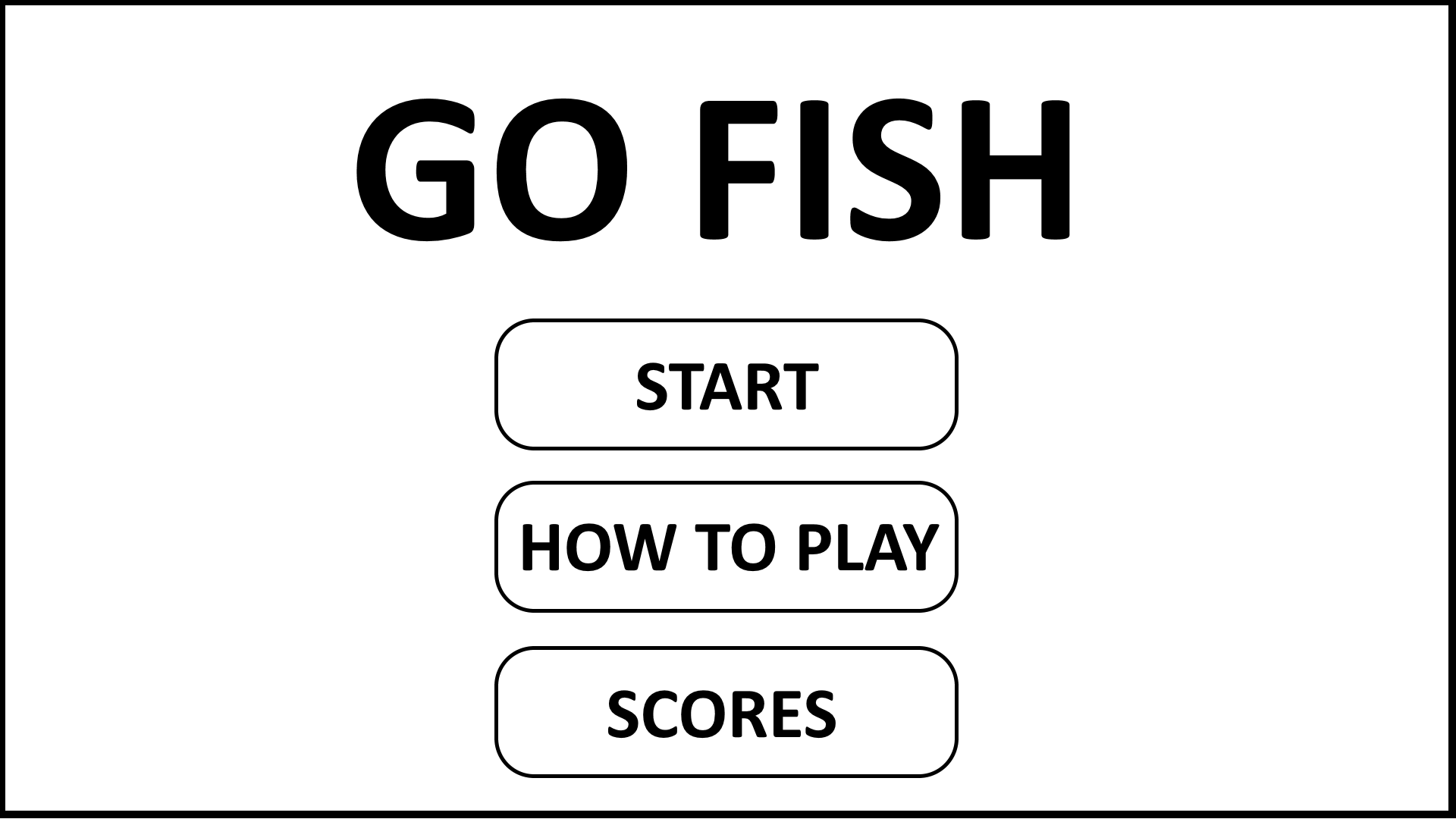
Desirable - Kid-friendly graphics and congratulatory messages are included to reward the user for winning or getting a pair.

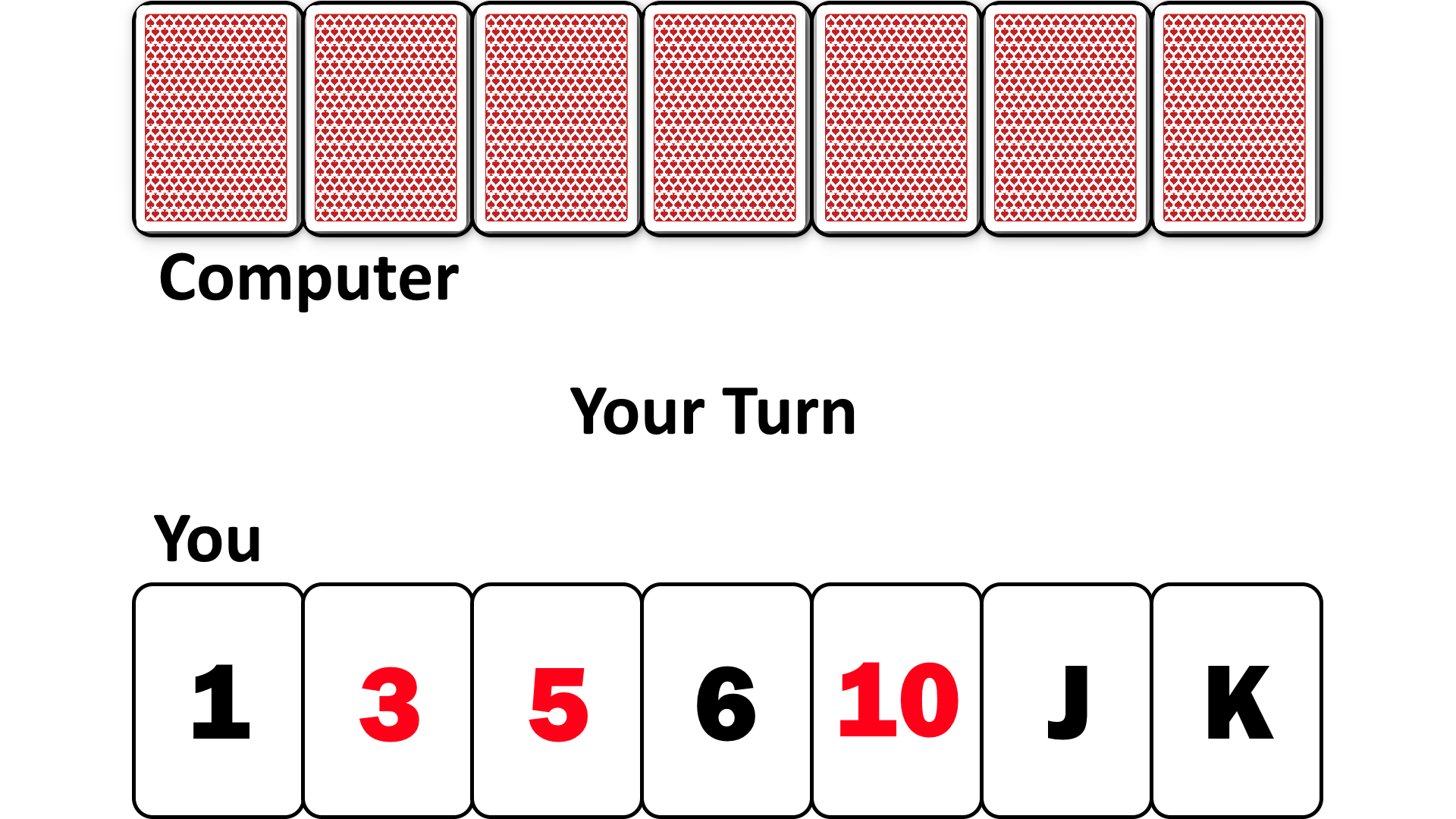
Valuable - Playing the game can be a learning experience for kids to learn words and pictures, as well as practice their speech and communication skills.

Error handling – The system should be able to handle all the errors properly for the user to have an efficient experience with the app

Some impacts that these user experience considerations have on the system design are that the displays/graphics are clear and not cluttered, and objects, such as buttons on the main menu and information while playing a game, are presented with large text. The voice recognition is also forgiving if it does not understand the input by asking for the user to repeat their statement at least five times.

Below are the preliminary screen mockups created:

**MAIN MENU SCREEN**

**GAME SCREEN**

**Team Retrospective**

**a)** We have been doing well meeting at least once a week as a team to work on deliverables and discuss our progress and plans. The documents are shared so we can all edit and work on them at the same time. Communication has been going very well as we also keep in touch outside of meeting times. Another thing that has gone well is the GTA meeting as there were no problems that arose during the meeting.

**b)** Trying to meet more than once a week every week has been a challenge since we all have different schedules. It is difficult to pick a time where every member of the group is available. Although it is not required, another thing that has not gone well is that not everyone can meet with the GTA during the GTA meeting. Usually some members are missing during the GTA meetings. We have also not begun formal implementation of the game/app yet at the time of writing this deliverable.

**c)** During times when we cannot all meet face-to-face but still need to communicate, we can call or video chat. If everyone is not able to make it to a GTA meeting, a representative is sent and will report back the main details of the meeting to the rest of the team. We also need to carefully plan out alternatives dates to meet up, such as on the weekends, so as that we do not fall behind with implementation or other project components.

**Contributions**

Ahmad managed this deliverable.

Ahmad:

* Technology
* Class Diagrams
* Sequence Diagrams
* Team Retrospective

Ivan:

* User Experience
* Class Diagrams
* Technology

Kevin:

* Class Diagrams
* Team Retrospective
* Sequence Diagrams

Ecem:

* Class Diagrams
* Team Retrospective
* Technology
* Sequence Diagrams