**Project Deliverable 2 – Architecture**

**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.

**Requirements**

**Functional Requirements**

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirement** | **Description/Rationale** | **Priority** | **Estimate** |
| Go Fish game mechanics | Handling standard Go Fish mechanics, such as dealing cards at beginning of game, checking for pairs, asking for cards, having a working computer AI player, and knowing when a player has won | 1 | 2 weeks |
| Android app Implementation | Implementing the Go Fish game into a viable Android application | 2 | 2 weeks |
| Voice input | The system will receive voice input and respond according to it | 3 | 1 week |
| Main menu | Creating a menu including a play button, a how to play tab that explains how to play Go Fish and how to use the voice recognition, a themes drop-down box, and a quit button | 4 | 1 week |
| Themes | Different themes for the cards, such as colors, animals, and food | 5 | 1 week |
| Graphics | Pictures for the cards, avatars, and background | 6 | 2 weeks |

**Nonfunctional Requirements**

|  |  |  |
| --- | --- | --- |
| **Requirement** | **Measurement Criteria** | **Constraints** |
| Accuracy of voice recognition | Commands should be repeated at most 5 times | Users would have to speak clearly enough for the voice recognition to be able to understand |
| Permissions - accessing device’s microphone | User must give permission for microphone access | The game will only work on devices that have a microphone |
| Learnability - learning how to use the app to play Go Fish with different themes and voice recognition | Users will be able to learn in less than 10 minutes (assuming they know the rules Go Fish) | App would need to be made simpler, possibly including a “Help” button |

**\*Neither the functional nor non-functional requirements have changed.**

**System Architecture**

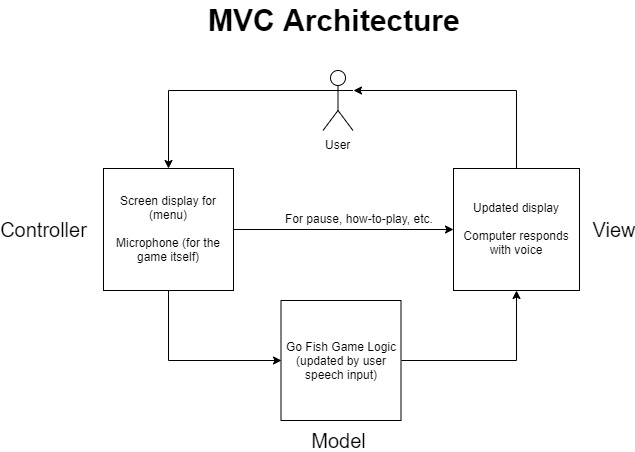
The architectural patterns that we plan to apply to our system include a call and return architecture and a model-view-controller architecture (MVC). The call and return architecture provides an overview of the whole system architecture and a general idea of the overall system architecture. On the other hand, the MVC architecture is used to model the architecture for specific instances of interactions between the user and the system. In this situation, the model is the Go Fish game, the view is the UI/app view, and the controller is the microphone and speech to text API.

This decision to include the MVC architecture was necessary as we felt the need to have an architecture that provides more details on the interactions between the components of the system (model, view and controller) with the user. The reason this is relevant to our specific system is that the system relies heavily on external user interaction which must be processed and applied to the model (Go Fish game). Furthermore, the MVC architecture is a common one used for mobile app and web development.

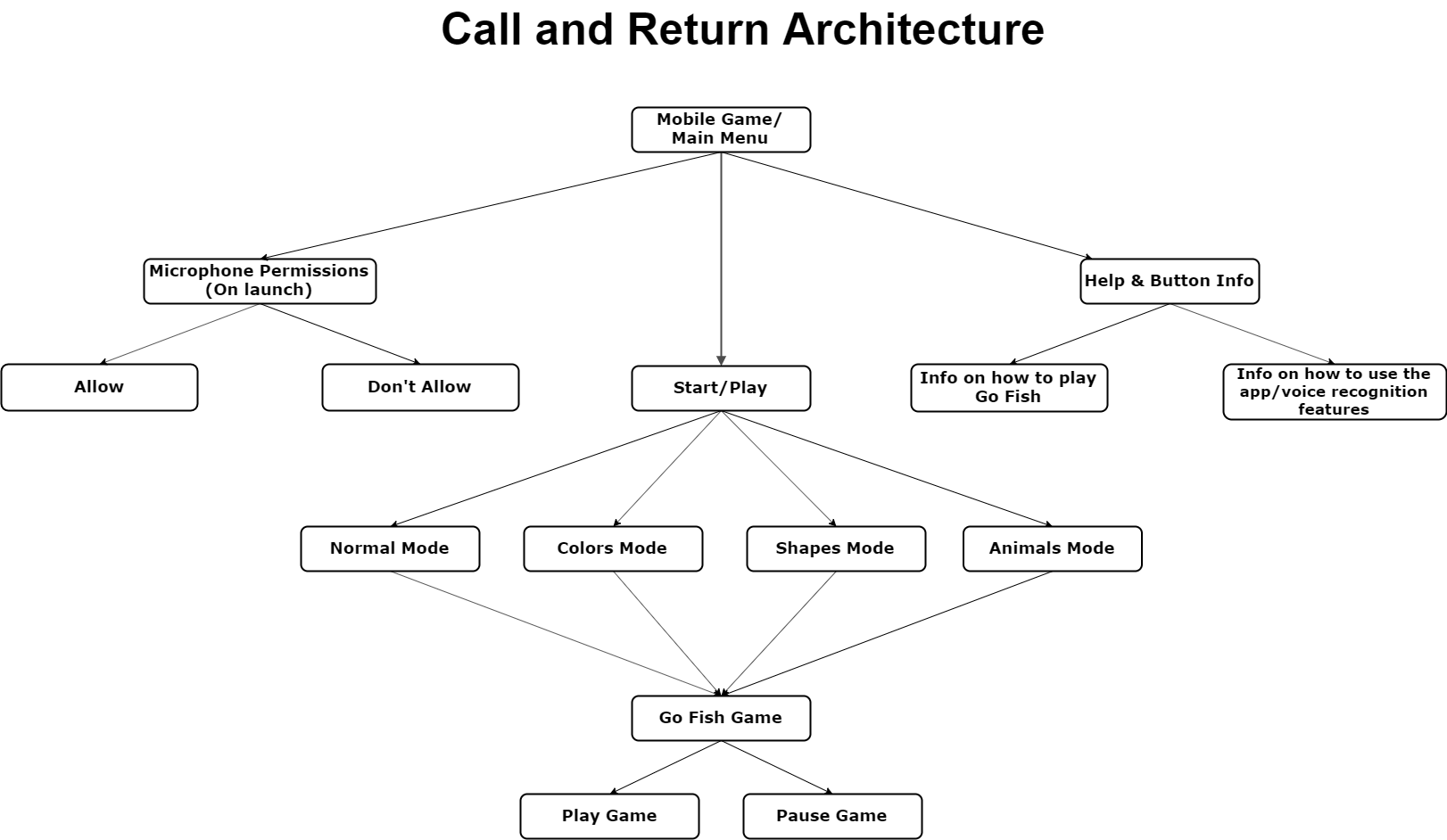
The call and return architecture is also justified as it shows a general domain of the overall system (from the main app to sub features). By putting the two architectures together, we are able to have one architecture that describes the overall system (call and return) and one that can be applied more for individual instances of interaction between the user and the app (MVC). Inspiration for architectural patterns and visual representations were sourced from various resources [1], [2]. Visual representations are later provided.

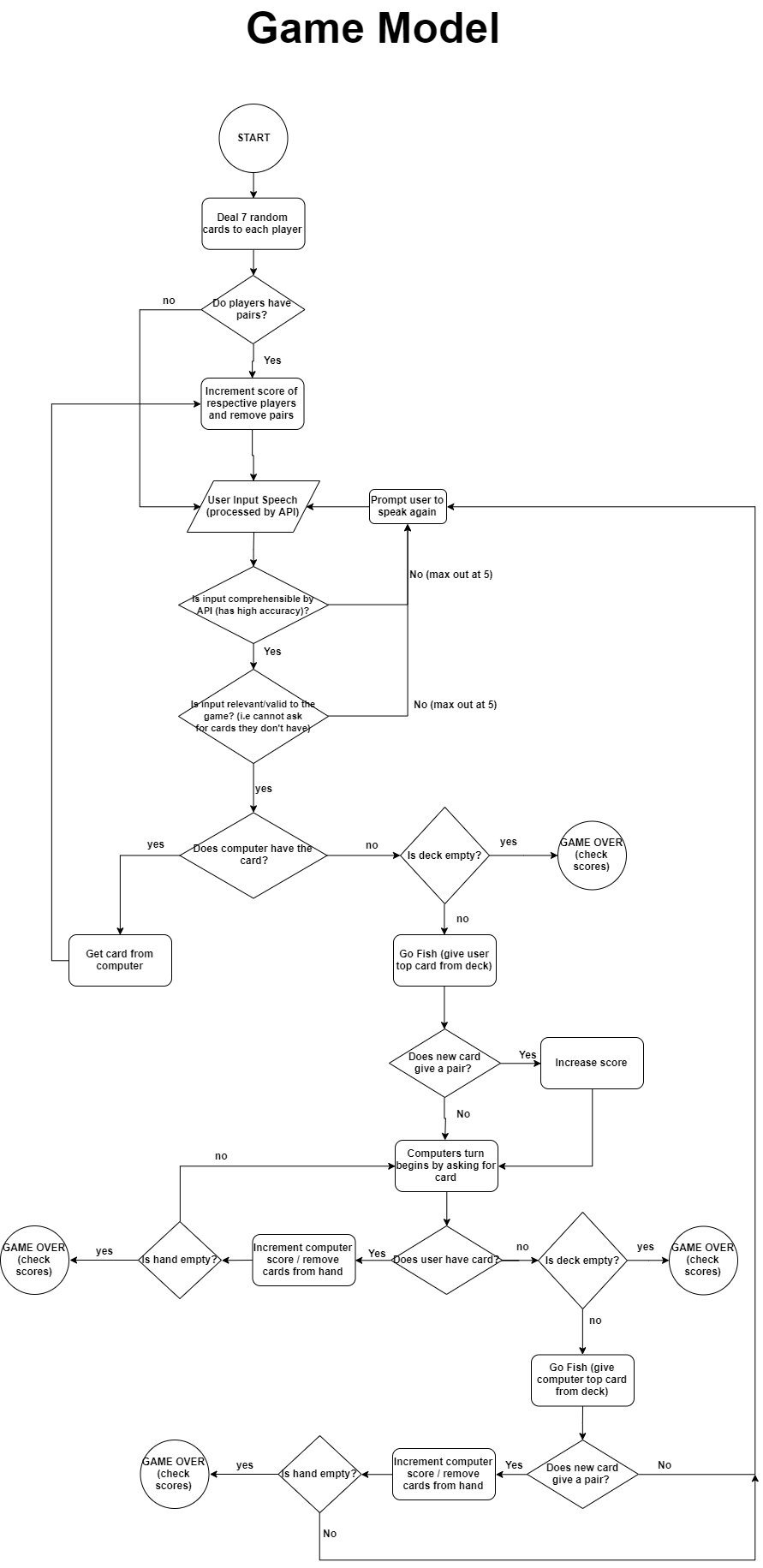
As for the reflection of the non-functional requirements within our architectural decisions, the learnability requirement is achieved by updating the display appropriately based on input without causing confusion. Furthermore, the user can also request to access the help menu. Also, the microphone permission requirement is achieved by the user allowing the controller to have access to the microphone for input. Finally, the voice recognition accuracy requirement is achieved by the model updating view accordingly based on input. For instance, if the controlling API processes the speech but with low accuracy, then the model will recognize that the request is violating the established non-functional requirement and will therefore prompt some sort of message onto the view requesting the user to speak again. These are just a few relevant ways that the architectural patterns we have chosen to reflect the non-functional requirements.

On the next page is a visual representation of the MVC architecture:



Below is a visual representation of the call and return architecture:





\*The model on the left is a supplementary visual to describe the general flow of the actual Go Fish game model referenced in the architectures above. Note that this is not an actual architecture. This is just a draft and subject to change.

**User Experience**

Usable – Voice commands and the menu must be simple enough for kids to use it. A 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.

The user experience considerations did not have an impact on the architectural decision.

**Team Retrospective**

One thing that is going well so far is meeting face-to-face at least once a week. Also, doing the deliverables on the Mason Outlook on Word so everyone can edit and see the changes in real time is going well. Also, everyone is accessible, so communication outside of group meeting times are going well.

One thing that has not gone well is trying to meet face-to-face more than once a week consistently. Everyone’s schedule each week varies so it is difficult to find a time where each member is available.

We decided that if a team member cannot meet face-to-face and we need more time to work on something, we can voice or video chat everyone and communicate through that.

**Contributions**

Kevin managed this deliverable.

|  |  |  |  |
| --- | --- | --- | --- |
| **Ahmad** | **Ivan** | **Ecem** | **Kevin** |
| * Team Retrospective | * Architecture | * User Experience | * User experience |
| * Architecture | * System Overview | * Requirements | * Team Retrospective |
| * Flow chart | * Requirements | * Architecture | * Architecture |
|  |  | * Flow chart | * Flow chart |

**References**

[1] En.wikipedia.org. (2019). *Model–view–controller*. [online] Available at: <https://en.wikipedia.org/wiki/Model–view–controller> [Accessed 11 Oct. 2019].

[2] Studytonight.com. (2019). *Understanding Basic Game Architecture | Studytonight*. [online] Available at: <https://www.studytonight.com/3d-game-engineering-with-unity/game-development-architecture> [Accessed 11 Oct. 2019].