Kevin Gottfried

Go Fish

The program created, plays the game of Go Fish as defined by the requirements document given to the development team. It includes three types of computer opponents to play against using a command line interface. Also included are basic instructions for the game and statistical records of games played. The development process followed a streamlined Waterfall method where a design was made based on the requirements document and coding responsibilities were divided among the team members. With biweekly team meetings, all members of the team stayed up to date on the progress of the rest of the team and any questions that arose could be addressed. Once all the parts of the game were coded it was all integrated into the complete game. Testing was then done on the game and bugs were worked out as they were found. Once testing was completed the final game was polished by cleaning up the UI and improving comments in the code.   The team stayed organized by utilizing a shared Google drive, where we added any documents, code, and other files for easy access.

The Go Fish game that we developed is run by a central game manager. This game manager handles all the implementations and instances of any players, UI, stats, and anything else involved in the game. There are 4 major components to our software. The game manager, a UI for ease of human computer interaction, a human player, and a computer. There exists a player superclass in which are instances of a player in the game are a subclass to. There exists three AI classes and a human player class. Each of these classes differs slightly in their logic, but still implement ideally the same functionality. The game manager starts a game and allows the player to select their function of the game (stats, play, help) and then allows them to pick the difficulty and then manager then respectively creates a computer to play. Each player is given a hand and the human player starts. The user is prompted to ask for a card and the computers hand is checked for an equivalent match of that card. The computer then returns all possible cards and proceeds to take its turn. During the computer's turn, a random integer is created and then a card is retrieved from that corresponding index of the hand represented as a linked list. Similar to the human player, the computer automatically asks and takes any possible cards from the human's hand and lays down any sets that it may have found. Being the central processing unit of the software, the game manager was the first unit that needed to be created and work properly for the rest of the components to be built around the logic of the processer. After meeting and talking about our approach and specifications regarding the components and requirements of each component, Robert created the game manager and a skeleton of certain functions in the manager that each player will need to utilize. This allowed Fabian and I to design the superclass that is the Player and then myself to design the human player and the AI player classes as well. Sean separately built an implementable UI that created an ease of interaction between the human and the computer, mainly through print statements and prompts. The UI is utilized in every aspect of the game which creates a simplistic and consistent method of user interaction and development. A large majority of the time was spent on specification and testing. We spent a week and a half meeting before class to outline and continue delving into the specifics and requirements of all the subcomponents to completely visualize the flow of the system. This made the development aspect simpler, and with a good game manager, it allowed the other classes to fit right into place. Some, but not all the units were tested individually, due to complexity of building a testing unit for these components. When the integration and system testing occurred, it took a lot more brute force and effort to individually test the computer components with the system. There was a lot of debugging and error checking that occurred during this step of the process, and this took up most the effort in my case. More time was spent debugging and testing, than was spent on developing and writing the system. I spent most of my time tweaking and adding code to fit and exceed the expectations and requirements that we set into place.

I think we all communicated well as a whole. We all could express our concerns and needs as they were. We planned and established meeting times efficiently, which is not always easy to do with students and varying schedules. The specifications and requirements aspect seemed to be the easiest task to accomplish, mainly because of our wide knowledge and experience of the game. It was easy for us all to express opinions and thoughts about the different components of the game because we understood the fundamentals and flow of how the game is played. The development aspect was made a lot easier from thorough requirements and a thorough understanding. Integration was simple because of the modularity that we designed into the system, however testing the system took a lot of resources and time, mainly because to test each component, multiple full games needed to be played. The biggest issues that were found regarded the computer classes and the methods that we utilized to check for sets and ask for cards. The transfer of cards in and out of a player's hand took a lot tweaking to get perfect as well. Overall, almost, if not all, issues were debugged during testing and the system was successfully integrated. There did not seem to be any major hiccups during the development cycle that disallowed us to continue as planned.