*Blackjack Development – Self Assessment*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Project Rubric | | | | Selected Rating | | | |
| Criteria | Ratings | | | | | | Pts |
| **Conceptual Coverage** | **20 pts [Excellent]** | **15 pts [Substantial]** | **10 pts [Pass]** | | **5 pts [Below Expectations]** | **0 pts [Absent]** | **20** |
| Justification | Clear use of user I/O throughout the code. The user is required to interact with the game, mainly through the player loop functions to give information to the computer. This information is used to perform many interesting computations – pulling a specific card of suit and face value based on a random 52 integer value or changing that value based on user input – aces to 11s, etc. Players can choose to hit or stand, and conditional execution takes that choice into consideration and performs different functions respectively. Output is evident and is vital in showing the user how to play but also the results of their inputs. User and dealer hands are formatted to follow the rules of blackjack and nicely formatted messages follow interesting computations to show the user’s win status and current bet / balance. Rules are outlined at the beginning of the game to make sure the user understands how this version of the game is played.  A visual, in the form of a plotted graph can be seen at the end of every game. The graph serves two main purposes – to aid in driver testing stage of development, as outlined in that document, and to show the user how their balance varied over the course of the game and perhaps show them where they can improve in future games.  Programming concepts are evident throughout the code - Frequent use of for loops throughout the code – used in several ways, mainly to iterate through a player or dealer hand. Many complex nested for loops that contain conditional execution exist throughout the code to make decisions about the status of the dealer and player hand. Vectors are frequently used throughout the code, mainly to store information regarding the status of the player or dealer hand. Matrices are used to store different types of information in separate rows, specifically storing the value and the suit of a card in a single array. 2D arrays are not used in the final version of the game, only within drivers to test and cross reference different data sets. Conditional execution in the form of while and if statements are used throughout the code to execute certain functions in specific circumstances. The biggest use of conditional execution is to keep replaying the game until the player balance reaches 0. Complex functions and even nested functions can be found throughout the code and are used in incremental development to test different mechanics of the code as well to streamline the code and make it easier to update throughout development. Many interesting functions and computations that work with the user input to create a complex and engaging user experience. | | | | | | |
| **Added Value** | **20 pts [Excellent]** | **17 pts [Substantial]** | **13 pts [Pass]** | | **6 pts [Below Expectations]** | **0 pts [Absent]** | **20** |
| Justification | Code clearly outlines the different functions and how they interact with each other. Functionality works flawlessly and executes the code in the desired way. Very interesting behaviour within certain functions – take the intialBet function; It takes a bet and decides whether it is valid or not based on the balance in which the player entered. Another interesting function is findCard. It takes an integer from 1:52 and assigns a suit and face value based its position. Creativity used not only to bring complexity to code but to also fix some issues. Aces in the final version are less functional but far less confusing to edit and far more convenient for the player.  Extensions to the basic functionality of the code are frequent throughout the program – a graph is added to the final output of the game, showing the player’s balance over time. A score system is used to tally the player’s highest score as their highest balance in each game, outputted in the final output. Sounds are added to the final win calculations and plays a specific sound depending on whether you win or lose. A while loop is used to keep repeating the game over and over until the player’s balance is 0. Players can also choose to cash out before going broke. A betting system is an extension of the player’s balance, performing interesting computations with a bet based on the player’s performance. Rules can be chosen to view or not at the start of each game. Player’s can ask for hints as to whether they should hit or stand, based on the sum of their hand. There are 4 different hints that can help the player.  Many learning extensions can be seen throughout the code. Nested functions are frequently used. Further knowledge within plotting is exercised, namely, to label axes and titles of the graph. Conditional execution with knowledge of string types shown. Keywords such as break, and return are used within conditional execution. The biggest addition was the incorporation of Boolean variables in function calling. The randperm function has also been used. Read and play sounds functions have been researched and used. No piece of code has been copied and is entirely my own work.  Questions about code CAN be answered in detail if needed, justification of further development and modification was outlined in the testing and driver documentation. Please see my video in full describing my program in great detail. | | | | | | |
| **Incremental Development** | **20 pts [Excellent]** | **15 pts [Substantial]** | **10 pts [Pass]** | | **5 pts [Below Expectations]** | **0 pts [Absent]** | **20** |
| Justification | Substantial evidence of incremental development, as outlined in the testing and driver documentation. Initial drivers were used to test the 4 main functions of the code – shuffleDeck, cardFinder, hitDriver and firstTurn. 9 prototypes that improve on each other and clearly show their problems / fixes and the different inputs used to test them. Where specific inputs were unable to be tested, justification was given, and written documentation was provided in its place. All initial function and prototypes / intermediate files are thoroughly commented and showcase small increments of development. | | | | | | |
| **Testing** | **20 pts [Excellent]** | **15 pts [Substantial]** | **10 pts [Pass]** | | **5 pts [Below Expectations]** | **0 pts [Absent]** | **20** |
| Justification | Clear evidence of testing throughout different stages of development – as outlined in its respective document. Drivers are used to extensively test the four main functions as well as the expected final form of each function of the program and intermediate files are used to showcase the smaller incremental mechanics added to the program. | | | | | | |
| **Comments and Style** | **20 pts [Excellent]** | **15 pts [Substantial]** | **10 pts [Pass]** | | **5 pts [Below Expectations]** | **0 pts [Absent]** | **20** |
| Justification | Software development practices are evident throughout the code – commenting is frequent and descriptive within the main body of the program as well as the different functions. Where a function is used in multiple different circumstances, commenting is used to distinguish between the two, notably for the cardFinder function where it can be used for any length hand for the dealer or the player. Code style – including naming conventions, organisation of files, layouts / documentation and statements are of high quality and closely follow the MATLAB style booklet. These areas of style are evident throughout all stages of development. A single document can be found in the main directory that contains thorough documentation of the different tests and incremental development stages. A 1.0 version can be found with its respective functions, separate from its prototypes. A final stage of development was pushed to ensure style was as best it could be – mainly to make functions reusable. Many functions were made more general purpose to be called more times and with different parameter sets. | | | | | | |