**Group Members & Contributions**

|  |  |  |
| --- | --- | --- |
| Name | ID# | Contribution/s |
| Dinito Thompson | 1801202 | TheWheel.h   * WheelSection * WheelNode * WheelLinkedList   GameFunctions.h   * SectionIdentifier(int, int) * SpinTheWheel() * TheWheelCreator() * ThePlayerBaseCreator() * GamePlayMenu() * MemoryAllocator()   Play.cpp |
| Shanice Facey | 1701438 | GuessedQueue.h   * Queue   GameFunctions.h   * SectionIdentifier(int, int) * SpinTheWheel() * GameContinue()   Play.cpp  User Interface |
| Ricardo Johnson | 1702583 | Contestant.h   * Contestant Linked List   + Search(int)   + isEmpty()   + isFull()   + DeleteContestantList() |
| Rojae Davis | 1608195 | Contestant.h   * Contestant Linked List   + Display()   + Destroy()   + InsertAtBack(Contestant) |

**Data Structures Used & Why**

The Following is a list of all the Data Structures used for each section of this project and the justification for using each.

* The Wheel
  + Wheel Section
    - Data Type:
      * Integer: Even though we were working with money, at no point would there be a decimal generated as each calculation returned a whole number. An integer value was also used to generate a colour for each section when the wheel was spun.
      * String: Used to store the sectionType so we could differentiate between a Reward and a Penalty.
  + Wheel Node/Wheel Linked List
    - Data Type:
      * WheelSection: Used to store every Instance of a “WheelSection” that is created via “TheWheelCreator()” function.
      * WheelNode \* (pointer): Used to linked every Instance of a “WheelNode” created via the “InsertIntoWheel(WheelSection)” function.
    - Data Structures:
      * Circular Linked List:: This data structure proved to be the most optimal in terms of storing different instances of node and allowing us to link the last node to the first node, similar to how a real wheel of fortune would have each card/section linked.
* Contestant/s
  + Contestant
    - Data Type:
      * Integer: Similar to The Wheel, Even though we were working with money and a grand total at the end of each round, at no point would there be a decimal generated as each calculation returned a whole number.
      * String: Used to store the contestantName so we could differentiate between each contestant.
  + Contestant Node/Contestant Linked List
    - Data Type:
      * Contestant: Used to store every Instance of a “Contestant” that is created via “ThePlayerBasedCreator()” function.
      * ContestantNode \* (pointer): Used to linked every Instance of a “Contestant” created via the “InsertAtBack(Contestant)” function.
    - Data Structures:
      * Circular Linked List: Similar to the Wheel, this data structure allowed us to quickly alternate between Contestant in a turn based manner, 1 after the other in a seemingly circular manner just by calling the getNextPlayer() method.
* Guessed Queue: Used to keep track off letters guessed by each contestant per round.
  + GuessedQueue
    - Data Type
      * Array: We used an array of size 50 (# of letters in the alphabet) to keep track of each letter guessed for a round.
      * Integer: Used to traverse the array from front to back and allows us to clear the same array for reuse each round.

**Worst Case Asymptotic Analysis**

|  |  |  |
| --- | --- | --- |
| SectionIdentifier()  = 1 + 1  = 2  = Big O (1) : Constant | SpinTheWheel()  = 1 + 3 + 1 + N ( 6 ) + 1  = N ( 6 ) + 6  = Big O ( N ) : Linear | TheWheelCreator()  = N ( N ( 10 ) )  = N2 ( 10 )  = Big O ( N2 ) : Polynomial |
| ThePlayerBaseCreator()  = 2 + 3 ( 9 ) + 3  = 5 + 3 ( 9 )  = 5 + 27  = 32  = Big O ( 1 ) : Constant | **GamePlayMenu()**  = 1 + N ( 6 + 4 + 1 ) + 1  = 2 + N ( 11 )  = Big O ( N ) : Linear | **GameContinue()**  = 2 + N ( 4 + 2 + 1 ) + 1  = 1 + N ( 7 )  = Big O ( N ) : Linear |
| Search()  *Both TheWheel.h & Contestant.h*  = 1 + 1 + N ( 1 + 1 )  = 2 + N ( 2 )  = Big O ( N ) : Linear | **enQueue()**  = 1 + 51 + 1 + 1 + 1  = 55  = Big O ( 1 ) : Constant | **MemoryAllocator()**  = N ( 2 + 2 + 1 ) + 1 ( 2 + N ( 1 + 2 + 1 ) + 1 )  = N ( 5) + N ( 4 ) + 3  = Big O ( N ) : Linear |