Group Members Contribution

Name	Tasks
Craig Reid: 1806394	Created the puzzle to solve in the round "puzzleTo Solve()", created the" select puzzle at random method "selectPuzzleAtRandom()", add the audio and implement the effects method, implement bot's functionality, implement method for the queue, created the read from files methods, implement main() and miscellaneous task, link all methods, debugging and assist with the report.
Sackasha Griffiths:1800733	Created the circular linked list, the wheel class, implemented the method for the wheel "spinWheel()", clear round method, create LoadContestant() method exception handling, create user manual and assist with the report

Kenneal Stephens:1602964	Created the Game data method, created the display method, created the destroy method, create the initialize method created the menu, contestant class and method assist with the report

Data Structures used

Wheel:

A circular linked list was constructed for the wheel because of project requirements.

Contestant:

A circular linked list was constructed for the contestant, after analyzing other data structure we have

conclude that a circular list was best due to the operation that needed to be carried out, it is noted that players are selected in a circular manner.

Puzzle:

A circular linked list was constructed for the puzzle based on the method we chose to use, we decided to select puzzle at random and removing it from the list while returning the data to be used in the round.

reason for deleting the selected node is because , when select nodes at random there is a possibility of selecting the same node twice, and that is not what we wish to accomplish in our program.

Another reason for using the selected data structure is that; when deleting a node, we only need to check if there is only one item in the list, while other data structure may require us to check if it is the first and also if it is the last node in the list; hence making it more efficient when removing a none and linking the previous node to the next node, because every node is linked to a previous and a next node.

Guessed letters:

A Queue was used for the letters, vowels and numbers that has been previously used in the given/ current round, this data structure was implemented as the results of project requirements.

Big O Analysis

```
        public boolean circulyList(Card cardObject, Contestant player, Puzzle puzzle) {
                if(Full()) {
1                       System.out.println("List is full");
1                       return false;
                }else {
1                       Node nextValue = new  Node();
                        if(cardObject != null)
1                               nextValue.setCardData(cardObject);
                        else if (player != null)
1                               nextValue.setPlayerData(player);
                        else
1                               nextValue.setPuzzleData(puzzle);
                        if(Empty()){
1                               head = nextValue;
1                               tail = nextValue;
                        }else {
1                               tail.setNext(nextValue);
1                               nextValue.setPrev(tail);
1                               tail = nextValue;
                        }
1                       tail.setNext(head);
1                       head.setPrev(tail);
1                       return true;
                }
        }
```

1+1+1+1+1
= 5
The big O of (circulyList()) is big O of (5): Constant

```
        public Node selectRandomPuzzle(int selectedIndex, int TotalPuzzles) {
1               int index = 0;
1               middleIndex = (TotalPuzzles/2);
1               boolean greater = false;
1               Node current = head;
                if (selectedIndex > middleIndex) {
1                       greater = true;
1                       current = tail;
1                       selectedIndex = TotalPuzzles-(selectedIndex+2);
                }
                if(head == tail) {
1                       head = null;
1                       tail = null;
                }else {
Log                     while(index <= selectedIndex ) {
                                if(!greater) {
                                        if(index > 0) {
1                                               current = current.getNext();
                                        }
                                }else {
1                                       current = current.getPrev();
                                }
1                       index++;
                        }
1                       Node temp = current;
1                       temp.getNext().setPrev(temp.getPrev());
1                       temp.getPrev().setNext(temp.getNext());
1                       current = null;
1                       return temp;
                }
1               return current;
        }
```

1+1+1+1+1+1+1+1+[(Log)(1)]+1+1+1+1+1+1+1
= 7+[(Log)(1)]+6
= 13+Log
= Log
The big O of (selectRandomPuzzle ()) is big O of (Log): Logarithmic

```
        public Node SelectNextNode(){
1               Node current = head;
1               Node Data = new  Node();
                if(!Empty()) {
1                       Data = current;
1                       head = head.getNext();
1                       tail = current;
                }
1               return Data;
        }
```

1+1+1+1+1+1
= 6
The big O of (SelectNextNode ()) is big O of (6): Constant

```
        public void resetRoundTotals(int numOfPlater){
                if(Empty())
1                       return;
                else{
1                       Node current = head;
N                       for(int i = 0; i < numOfPlater; i++) {
                                if(current.getPlayerData().getRoundTotal() > 0) {
1                                       current.getPlayerData().setRoundTotal(0);
                                }
1                       current = current.getNext();
                        }
                }
        }
```

1+ N(1)+1
= 2(N)
= N
The big O of (resetRoundTotals ()) is big O of (N): Linier

```
        public void deleteFromHead(){
                if(Empty())
1                       return;
                else {
1                       head.setPrev(null);
1                       tail.setNext(null);
                        if(head == tail) {
1                               head = null;
1                               tail = null;
                        }else {
1                               Node delete = head;
1                               head = delete.getNext();
1                               delete = null;
                        }
                }
        }
```

1+1+1+1+1
= 5
The big O of (deleteFromHead ()) is big O of (5): Constant

```
        public void destroy() {
N               while(!Empty()) {
5                       deleteFromHead();
                }
        }
```

5(N)
= N
The big O of (destroy ()) is big O of (N): Linier

```
public boolean EnQueue(Guess guess) {
                if(Full()) {
1                       System.out.println("List is full");
1                       return false;
                }else {
1                       Node nextItem = new  Node();
1                               nextItem.setGuessData(guess);
                        if(Empty()){
1                               Front = nextItem;
1                               Rear = nextItem;
                        }else {
1                               nextItem.setPrev(Rear);
1                               Rear.setNext(nextItem);
1                               Rear = nextItem;
                        }
1                       return true;
                }
        }
```

1+1+1+1+1+1
= 6
The big O of (EnQueue ()) is big O of (6): Constant

```
        public Guess DeQueue() {
1               Node nodeToDelete = Front;
1               Guess Data = nodeToDelete.getGuessData();
                if(Empty()) {
1                       System.err.println("the Queue is empty");
                }else {
                        if(Front == Rear) {
1                               Front = null;
1                               Rear = null;
                        }else {
1                               Front = Front.getNext();
1                               Front.setPrev(null);
1                               nodeToDelete = null;
                        }
                }
1               return Data;
        }
```

1+1+1+1+1+1
= 6
The big O of (DeQueue ()) is big O of (6): Constant

```
        public boolean search(String enteredLetter) {
                if(!Empty()){
1                       Queue temp = new Queue();
1                       Guess current;
1                       boolean found = false;
N                       while (!Empty() && !found) {
6                               current = DeQueue();
                                if(current.getGuessed().equals(enteredLetter)) {
1                                       found = true;
                                }
6                               temp.EnQueue(current);
                        }
N                       while (temp.Front != null) {
6                               EnQueue(temp.DeQueue());
                        }
                        if (found)
1                               return true;
                }
1               return false;
        }
```

1+1+1+N(6+6)+N(6)+1
= 4+N(12)+N(6)
= 4+ 18(N)
= 18(N)
= N

The big O of (search ()) is big O of (N): Linier

```
        public void display(String typeOfLetter) {
                if(Empty())
1                       return;
                else{
1                       Queue temp = new Queue();
1                       Guess current;
N                       while (!Empty()) {
6                               current = DeQueue();
                                if( typeOfLetter.equals("VOWELS")){
                                        if(current.getGuessed().equals("A") |
current.getGuessed().equals("E") | current.getGuessed().equals("I") |
current.getGuessed().equals("O") | current.getGuessed().equals("U")) {
1                                               current.display();
                                        }
                                }else if( typeOfLetter.equals("REGULAR")) {
                                        if(!(current.getGuessed().equals("A") |
current.getGuessed().equals("E") | current.getGuessed().equals("I") |
current.getGuessed().equals("O") | current.getGuessed().equals("U"))) {
1                                               current.display();
                                        }
                                }
6                               temp.EnQueue(current);
                        }
N                       while (temp.Front != null) {
6                               EnQueue(temp.DeQueue());
                        }
                }
        }
1+1+N(6+1)+ 6+N(6)
= 8+N(7)+N(6)
= 8+ 13(N)
= 13(N)
= N
```

The big O of (display ()) is big O of (N): Linier

```
        public void destroyQueue() {
N               while(!Empty()) {
6                       DeQueue();
                }
        }
```

6(N)
= N
The big O of (destroyQueue ()) is big O of (N): Linier