|  |  |  |  |
| --- | --- | --- | --- |
| **Coventry University**  **FACULITY OF Engineering and Computing**  **Coursework Cover Sheet**  *Please ensure that you complete all relevant sections legibly*  ***First Copy:*** *Attach top copy to the front of your assignment.*  ***Second Copy:*** *Keep safety as your receipt* | | | |
| **Module Code**  **210CT** | | **Student Card ID Number**  Please print in BLOCK CAPITALS  Surname…………………LIN.……………………………  Other names………………ZIQIAO….………..........  Signature………………….……...…..…...  **6826257** | |
| **Module Title**  **Programming, Algorithms and Data Structures** | | | |
| **Deadline date**  **Part 1 – 27th Oct 2015**  **Part 2 – 17th Nov 2015** | **Actual word**  **Count N.A.** | | **Tutor**  **CY Cheng** |
| **Coursework Title/Number**  **Assignment** | | | |

This assessment is all

my own work and has not

been copied in part or in whole

from any other source, except for any

clearly marked up quotations. It

complies with the university regulations

on plagiarism, which I have read

and understood.

|  |  |  |
| --- | --- | --- |
| **FEEDBACK ON MARKED WORK:** *Lecturers will complete this section when work is marked.*  *NB. All marks notified during the year are provisional until confirmed by the end of year Assessment Board* | | |
| STRENGTHS  WEAKNESSES  ADVICE ON HOW WORK COULD BE IMPROVED AND FURTHER COMMENTS | | |
| If you require more feedback, please contact your tutor or see module web.  See assignment sheet for assessment criteria for this assignment. | **MARKED AWARDED** |  |
| Less any late penalty | **-** |
| Adjusted mark if penalty |  |

**Marker’s Signature** …………………………………………………………… **Date** ……………………………………………………….

|  |
| --- |
| Students are reminded that reference must be given for any previously published  work used to gather information to help write assignments, including internet  sources, but these sources should not be copied directly. |

**Second Marker Additional comments Signature** ………………Nil…………………………… **Date** …………………………

Table of Contents

[CLASS DIAGRAM 3](#_Toc433812701)

[VARIABLE DESCRIPTION 4](#_Toc433812702)

[CLASS DESCRIPTION 6](#_Toc433812703)

[METHOD DESCRIPTION 7](#_Toc433812704)

[REFERENCE 11](#_Toc433812705)

# CLASS DIAGRAM

|  |
| --- |
| **Game** |
| - null\_char : char  - grid\_width : int  - undo\_list\_size : int  - randomFillChar\_array\_size : int  - alphabet : char[]  - randomFillChar : char[]  - clone\_randomFillChar : char []  - grid : char[][]  - clone\_grid : char[][]  - score : int  - clone\_score : int  - counter : int  - clone\_counter : int  - undo\_grid : ArrayQueue  - undo\_score : ArrayQueue  - undo\_randomFillChar : ArrayQueue  - undo\_randomPosition : ArrayQueue  - gameover : boolean  - victory : boolean |
| + Game()  + display() : void  + slideUp() : void  + slideDown() : void  + slideLeft() : void  + slideRight() : void  + randomFill() : void  + gameover() : boolean  + undo() : void  + reset() : void  + gridIsEmpty() : boolean  + gridIsFull() : boolean  + init() : void  + convertLetter(char) : int  + mergeTile(char) : char  + cloneState () : void  + moveValidate() : boolean  + addToUndoList() : void  + generateCharForRandomFill() : void  + shuffle(char[]) : void  + start() : void |

|  |
| --- |
| **Start** |
|  |
| + main(String[]) : void |

<<use>>

|  |
| --- |
| **ArrayQueue** |
|  |
| - maxSize : int  - undo\_list : Object[]  - front : int  - rear : int  - count : int |
| + ArrayQueue(int)  + ArrayQueue()  + empty() : boolean  + full() : boolean  + length() : int  + enqueue(Object) : void  + dequeue() : Object  + pop() : Object  + toString() : String |

<<use>>

|  |
| --- |
| **RuntimeException** |
|  |
|  |

<<use>>

|  |
| --- |
| **EmptyQueueException** |
|  |
| + EmptyQueueException () |

|  |
| --- |
| **QueueFullException** |
|  |
| + QueueFullException () |

# VARIABLE DESCRIPTION

null\_char : char

It represents the null value of char. In Java, we can not compare char with null directly. Thus, I use this ‘\u0000’ to indicate the null value of char. It is a constant value.

grid\_with : int

A constant value of grid width is 4.

undo\_list\_size : int

A constant value of the undo list size is 20.

randomFillChar\_array\_size : int

I use an array to store the shuffled value and its size is 50. It is a constant value.

alphabet : char[]

This array stores the 26 letters of the alphabet. It is a constant value.

randomFillchar : char[]

This array stores the shuffled value ‘0’, ‘A’ and ‘B’ with ratio 40 : 9 : 1 respectively.

clone\_rondomFillChar : char[]

This array uses to clone the previous state of randomFillChar array before moving the grid then save it into the undo list for the undo function.

grid : char[][]

It uses to store the grid.

clone\_grid : char[][]

It uses to clone the previous state of grid before moving the grid then save it into the undo list for the undo function.

score : int

It uses to store the score after merging tiles.

clone\_score : int

It uses to clone the previous state of score before moving the grid then save it into undo list for the undo function.

counter : int

It uses to store the randomFillChar array index.

clone\_counter : int

It uses to clone the counter before moving the grid then save it into undo list for the undo function.

undo\_grid : ArrayQueue

It uses to store the previous state of grid for the undo function.

undo\_score : ArrayQueue

It uses to store the previous state of score for the undo function.

undo\_randomFillChar : ArrayQueue

It uses to store the previous state of randomFillChar array for the undo function.

undo\_randomPosition : ArrayQueue

It uses to store the previous state of randomFillChar array index (counter) for the undo function.

gameover : boolean

It uses to store the state of gameover.

victory : boolean

It uses to store the state of victory.

# CLASS DESCRIPTION

Game class

The class contains all the variables and methods as well as the whole game logic. A constructor to initial the variables. It reads the user’s input and give response back to the users. For example, input ‘2’, ‘4’, ‘6’, ‘8’ and ‘0’ are represent the slideDown(), slideLeft(), slideRight(), slideUp() and undo() method respectively to control the game movement and ‘q’ to quit game. Furthermore, I would add a reset() method to reset the game easily rather than restart the game.

Start class

This class is just used to invoke the game to start a new game.

ArrayQueue class

This class is used to support the undo function. For the undo function, I chose to use array to implement a Queue to handle the undo. I would add a pop() method to support it, because the Queue only has enqueue to add the element to rear and dequeue to remove element from front. Due to the fact that I decided to add a pop() method to complete the Last In First Out (LIFO) rule that I need to implement on the undo function.

RuntimeException class

This class is used to be extended to customize my own error classes.

EmptyQueueException class

The customized error class extends from RuntimeException class to show message “Queue is empty!” for the queue is empty situation. It used in enqueue() method.

QueueFullException class

The customized error class extends from RuntimeException class to show message “Queue is full!” for the queue is full situation. It used in dequeue() and pop() method.

# METHOD DESCRIPTION

Game class

Game() method

The constructor method for the Game class to initial the variables it needs.

display() method

This method is used to display the grid and score on the command prompt.

slideUp() method

The method uses to slide up the grid.

First, clone the all the information before moving the grid. Such as grid 2d array, score, randomFillChar array and counter of randomFillChar array.

Second, use nested loop to loop through the grid 2d array from up to down and from left to right. Check the cell is empty or not. Check the cell is last cell or not to avoid comparing the last cell. Because we don’t need to compare last cell, we are comparing the cell from up to down. If all are true, then go to the next stage. We use third for loop to compare the cell one by one from up to down until it find the same char to merge. If the cell is empty, we break the loop to jump the compare the next cell. If found the same char, we empty the one that we found and change the one that we use to compare with other to the next letter. For example, ‘A’ change to ‘B’ and ‘B’ change to ‘C’. Finally, break the loop for finished on comparing and jump to next cell.

Third, after we merged all the available pairs but they are not in the right position. Because they may have empty between cell. We need to bring all the char to the user’s input direction after merging.

Finally, check the move if valid or not, if it is valid call the randomFill() method to put a char to a random empty cell. Also, we add the cloned previous state to undo list.

slideDown() method

The method uses to slide down the grid.

Actually, the concept is the same as slideUp() method description, just need to adjust the start point and the end point of the nested loop.

slideLeft() method

The method uses to slide left the grid.

Others same as above.

slideRight() method

The method uses to slide right the grid.

Others same as above.

randomFIll() method

The method uses to random fill an empty cell of grid after successfully merging. In this method, we use the Random class to generate 2 number between 0 to 3, 0 and 3 are includsive. Use while loop to fill until the position is empty.

gameover() method

This method uses to check the game is game over or not.

We use the looping method is same as the slideUp() method’s second paragraph’s description. But when we found the cell can merge then change the gameover variable to false and immediately return it.

undo() method

This method uses for undo. First, check the undo lists size is greater than zero then pop the object out and replace original variable. If not print the “no moves can undo!” message.

reset() method

The method uses to reset the game. Just reinitialize the variables and regenerate the randomFillChar array to return game state to the start point.

gridIsEmpty() method

The method uses to validate the grid is empty or not.

gridIsFull() method

The method uses to validate the grid is full or not.

init() method

The method uses to put two random char to random position of grid when the game first start.

convertLetter(char) method

The method uses to return next position of alphabet array. Receive the char and find it from the alphabet array, if the char is ‘Z’ change victory variable to true else return next position of array.

mergeTile(char) method

This method uses to return the next letter of the current one by calling the convertLetter (char) method and calculate the score.

cloneState() method

This method uses clone the state of grid array, score, randomFillChar array and counter of randomFillChar array for preparing to add into the undo lists.

moveValidate() method

The method uses to validate the move is valid or not, the valid one is the move modifying the grid, if the move does not modify the grid, it considers invalid.

addToUndoList() method

The method uses to add the previous cloned state to the undo lists. Check the undo list is equal to undo\_list\_size 20, if true use the dequeue() method of ArrayQueue to delete the front element. Then, use enqueue() method to add element to the rear. If false directly add them into the undo lists.

generateCharForRandomFill() method

This method uses to generate ‘0’ 40 times, ‘A’ 9 times and ‘B’ 1 times then add them into randomFillChar array and shuffle it.

shuffle(char[]) method

This method uses to shuffle the randomFillChar array. I use the Fisher-Yates method, a fast shuffling algorithm, we loop over an array. We swap each element with a random element past the iteration point.

start() method

The method uses to print out the menu, handle the user input and call the display() method to display the grid. Game over and victory message also print out here.

Start class

main(String[]) method

This method uses to invoke the Game class.

ArrayQueue class

ArrayQueue(int) method

Constructor method for ArrayQueue class to initial the variables.

ArrayQueue() method

Constructor method for ArrayQueue class to initial the array queue size.

empty() method

The method uses to check the empty of the array queue.

full() method

The method uses to check the full of the array queue.

length() method

The method uses to return the length of the array queue.

enqueue(Object) method

The method uses to add element to the rear of the array queue. Throw exception if array queue is full.

dequeue() method

The method uses to remove the element from the front of the array queue. Throw exception if array queue is empty.

pop() method

The method uses to pop the element from the rear of the array queue. Throw exception if array queue is empty.

toString() method

The method uses to print out the elements in the array queue.

EmptyQueueException class

EmptyQueueException() method

Constructor method for display “Queue is empty” message.

QueueFullException class

QueueFullException() method

Constructor method for display “Queue is full” message.

# REFERENCE

[https://en.wikipedia.org/wiki/Fisher–Yates\_shuffle](https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle)