[Q2_24062219]

Marcus Francis TIPLER,
Computer Science at Cardiff University

Marcus Francis TIPLER

Cardiff University | ComSC – Abacws Building

GOALS

The objective was to create an engaging event driven object-oriented java puzzle game that challenges players to solve a magic square using input and output functionality. This program has to be a **fully functional prototype** of a game that shows fluency in selecting and using basic components of Java.
The aim of the program was to make a comprehensive and understandable game whereby: the user can choose the size of the solvable array and the program will make and shuffle a magic square matrix and allow the user to use inputs and outputs to remake the array in to a magic square.

SOLUTION DESIGN

Firstly, I started a new **Kanban Board** to stay organised and I opened a **GitHub Repository** for version control. The program needed to be modular, so I decided to take a "*more methods is better*" approach. Secondly, I made a simple overview of the methods the game would require. This gave me an overall idea of what methods I needed and it told me that I would need to import from the following libraries: "*Random*" and "*Scanner*".

I then started to look for existing solutions for shuffling algorithms and I found a method called the "*Fisher Yates Algorithm*". However, upon further inspection I decided to abandon this method to make my own solution. With the *Fisher Yates* algorithm I discovered I would still require to interpret the user input, and I decided to use the same method to shuffle the matrix in the first place. This removed the need for the *Fisher Yates Algorithm* all together. I also made the choice to allow the user to wrap the values around the puzzle in the design concepts. This is better for user experience.
Following testing: I realised that there was an exponential increase of the size of the matrix when high values are requested for the array size, so I decided to limit the input to a 33∗33 array.

COMPLETED OVERVIEW

The completed solution consists in a **class file:** "*q1a_24062219.class*", and the **Java program** itself: "*q1b_24062219.java*". From this point on, these shall be known as "*1a*" and "*1b*".
1b extends 1a in the sense that 1b can access methods in 1a and parse data in to it. 1a is the compiled program that creates the completed magic square based on it's given input. 1b is the command line accessible file that displays the storyline, shuffles the completed magic square and gathers the users input until the game is considered finished (when the array is indeed calculated as a magic square).

Overall, this class represents a simple implementation of the Magic Square Puzzle in Java. The **constructor** takes user input for the size of the matrix, and the "*generate*", "*print*", "*verify*", and "*loop*" methods perform their respective actions related to the puzzle game. The program is designed in **modular** way, with each class or method having a single responsibility. This makes it easier to read, understand, and maintain the code along with the consistent naming conventions, indentation and the clear and concise commenting throughout the program. **Try catch blocks** are scattered around the program at every potential failure point to robustly handle errors and a **global scanner** is used which prevents potential leaks in the code. The program **displays colours** to highlight important information to the user for a better experience, and this also reduces eye strain in my experience. The game also uses static variables for the purpose of readability, this allows the reader to understand their purpose in a clear way.

Originally, parsing values in to methods was not made simple. I had to find an alternative solution to send values back in forth, and for that I discovered **Public Variables**. That and importing 1a to 1b wasn't very well documented as I wasn't made aware that I needed to compile the **extension**. I was hesitant on using them as I'm aware that global variables in programming languages such as C or Python aren't generally a good idea. As I'm new to Java I discovered that Public Variables don't act in the same way as Global Variables from other languages. I also discovered that having a single scanner significantly reduces the problems with reading values and prevents from having more leaks in the program. The final issues I came across were during the testing stages of my program, where we discovered that the user input was inverted for Left and Right, and that the user could input an unfathomable number in to the array size causing the program to crash or in some cases, the computer to reboot.

I decided to improve the User Experience from that of the Performa that was given to us: the way the user interacts with the matrix normally prevents the user from wrapping the values around if the direction is out of bounds, however I modified this to allow the user to make the move of their choice, their–for improving the user experience and the game' overall play-ability.

TEST METHODOLOGY

For testing my program, I decided I would make all of my friends test my game once it was completed, however this came with its own problems. Although helpful, many of my friends we're too honest to try and break the program. Those that did attempt to break the program found many small and yet unexpected issues. Such as: the array size being set to "*999*" crashes the program on MacOS and with a Windows based operating system (in a virtualised environment) caused the virtual environment to crash and reboot. Changes were made to improve on its functionality.