DOBBLE

Qazi Ahmed Ayan
M. Lal

Oct. 6, 2024

Definetly not doing this in a PDF form because adding another menu option was not worth the headache yup

GENERAL RULES

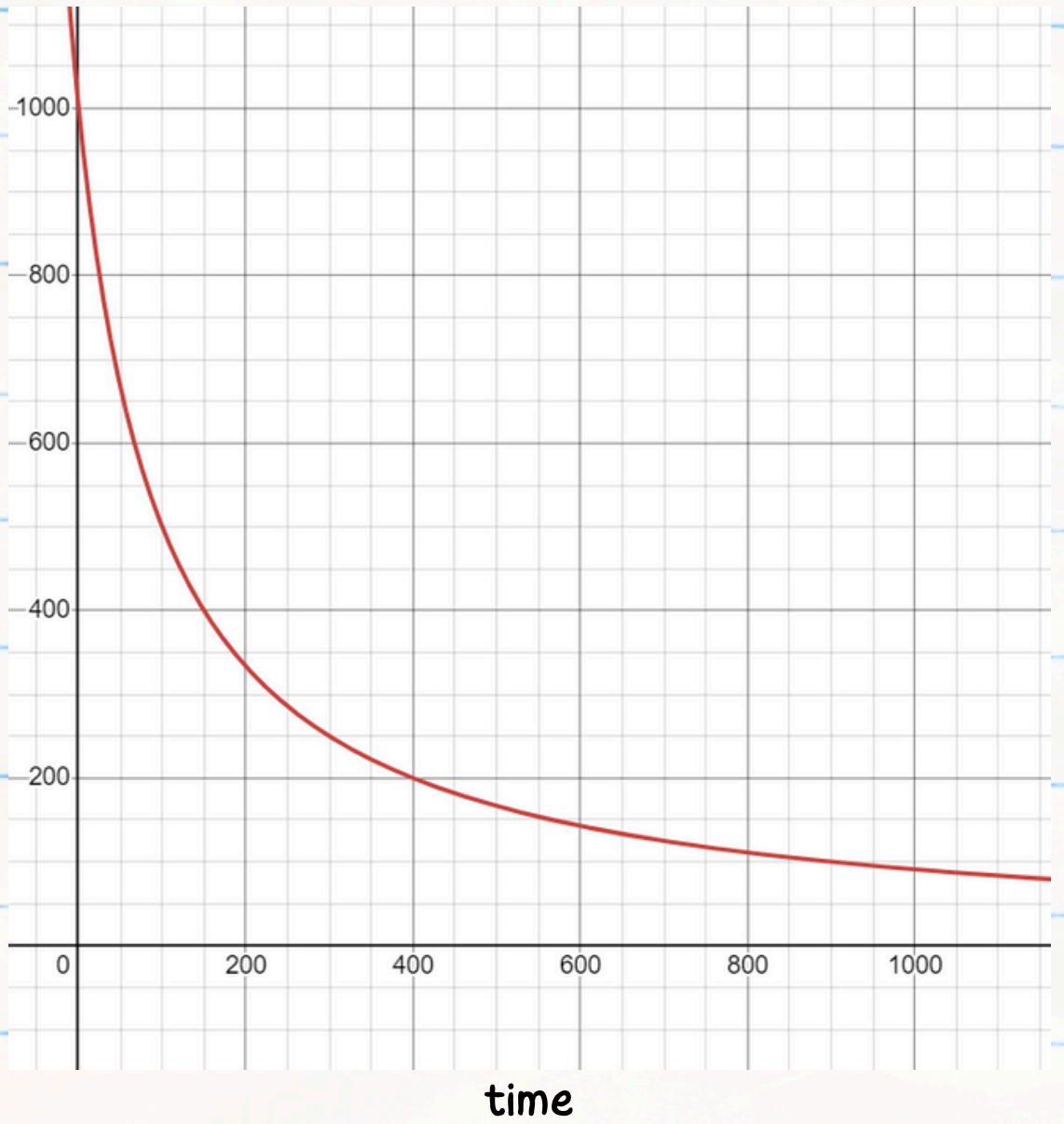
The game **Spot-it** (originally known as **Dobble**) works by having the user find **a common image** between two **randomly** selected cards. Although the original card game has **8 images** and **no timer**, this version has **a running timer** and uses **3 - 4 images**, given you, the user, are playing against yourself (ha imagine having no friends).

So cute, he doesn't even know what a for-loop is

CALCULATING SCORE

This super fancy equation is used to calculate the score:

$$y = \frac{100}{(x + 100)} \cdot 1000$$



This helps make it such that the score decreases somewhat gradually but never hits 0, and the maximum possible score (at 0s) is 1000.

GAME MODE #2

The Second Game mode allows the user (you!!) to **input their own values**, such as the names of their friends (if they have any that is), with the option to have **three** names/items per card or **four** names/items per card.

magical bunny provides a tip!
You can write 'quit' to quit any game mode and return to the start menu!

GAME MODE #3

The third game mode is timed and says whether you passed or failed based on the selected game.

Difficulty	Time/Rounds
Easy	25s /3 rounds
Medium	15s / 5 rounds
Impossible	10s / 10 rounds

REFLECTION

GENERAL SUMMARY

This project was an incredibly insightful and fun way to transition from 'thinking in Python' to 'thinking in Java', as well as learning many new coding concepts along the way. I personally believe projects are the best way to learn and build skills, and this proved that. Some of the things I learned were:

- How to use git/github
- How to use arrays & arraylists
- How to fix InputMismatch Errors in numerous ways
- How to better use methods
- Enhanced for-loops as well as switched loops
- Simplifying incredibly complex logic
- How to run into incredibly niche errors that one person from stackoverflow faced before I was born (kidding !!)

All-in-all, this project was incredibly useful in developing my skills as both a programmer as well as a student, and despite the many minor bugs and issues, how to actually enjoy the process.

REFLECTION CONT'D

BIGGEST CHALLENGE

The biggest challenge I faced was debugging and implementing the various different ideas I had into one cohesive bit of code. In the end, the code can still be considered 'spaghetti code', but given more time, I'm sure I would be able to better refine what I have. This also provided insight into approaching problems in the future in a much more methodical way, as discussed in the next question.

CHANGES FOR THE FUTURE

This project taught me **A LOT** about programming in general, and how to tackle problems. I approached the project blindly and added to it gradually, without considering any future ideas and implementations. This made it such that I was often left with incredibly redundant and confusing code which could have been cleared up if I had spent time planning and making a design/functionalities doc to stick to. As such, for future projects, I intend on outlining my personal requirements ahead of time and spending a lot more time actually planning and preparing compared to this project.

REFLECTION CONT'D

CURRENT BUGS

Ahh the least favourite section.
As of right now, there's one major logic error that I am unable to fix due to time constraints. If you quit during the third option it displays "You Won!" as well as the score. This can be fixed by rewriting some of the methods and general logic to be inclusive of the user guesses, but that would take up a lot of time to refactor and rework the code.
Another bug that could be fixed with more time is the fact that in game mode two, users can enter blanks. If the code were to be reworked to have inputs taken in using a loop and assigning values that way, this could be prevented with a simple while loop. It would also fix the fact that users can input multiple of the same name/items.
In addition, there are a few general errors which stem from quirks of Java. These include:

1. Time being 'rounded down' because of the way Java functions
2. The CYAN colour not being compatible with a black background (hence the splashscreen has alternating coloured backgrounds, given it didn't function with Cyan and it looked odd otherwise