Who wants to be a millionaire?

1. Introduction

The game I have decided for is based on the German version of „Who wants to be a millionaire?“. There are 15 questions. Each questions is connected to a certain amount (in $): 50, 100, 200, 300, 500, 1000, 2000, 4000, 8000, 16 000, 32 000, 64 000, 125 000, 250 000, 500 000 and 1 000 000. All questions must be answered correctly to win the million (the last question).

There a similar coding blocks for all 15 questions:

- The user can choose between four answer possbilities (A to D), he can pick a lifeline (E) or quit (F).

- Lifelines (if available):

    - 50:50: Two wrong answers are being removed. After that, the user has to choose one of the two remaining answers.

    - Phone a Friend: The user can call a friend. That means, he chooses the name of the friend who then returns the correct answer with the probability of 60 %. After that, the user can decide whether he follows the advice of his friend, choose another lifeline or quit.

    - Ask the Audience: The audience will be asked. It returns the correct answer with a probability of 75 %. After that, the user can decide whether he follows the vote of the audience, choose another lifeline or quit.

    - Each lifeline can be used only once in the whole game.

If the answer is correct, the game continues with the next question.

If it is wrong there are three possibilites depending on which question has been reached:

- Between 0 $ and 500 $: User won 0 $, game over.

- Between 1000 $ and 16 000 $: User won 500 $, game over.

- Between 32 000 $ and 1 000 000 $: User won 16 000 $, game over.

2. Design and Implementation

Every question is designed as object of the class Question. Attributes are for example the question itself, the four possible answers, and also the probabilities for the weighted random choices of the lifelines Phone a Friend and Ask the Audience.

The questions (the objects) are connected to each other as a simply linked list. So, when the user got a question right, immediately the next question will be presented to him.

Part of the algorithm are five functions:

- pose_question():

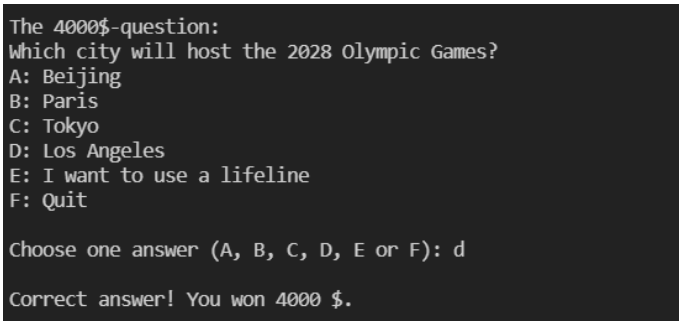
  This is the most important function. The questions are being posed. It presents the answer possibilities and specifies if the answer was correct or not.

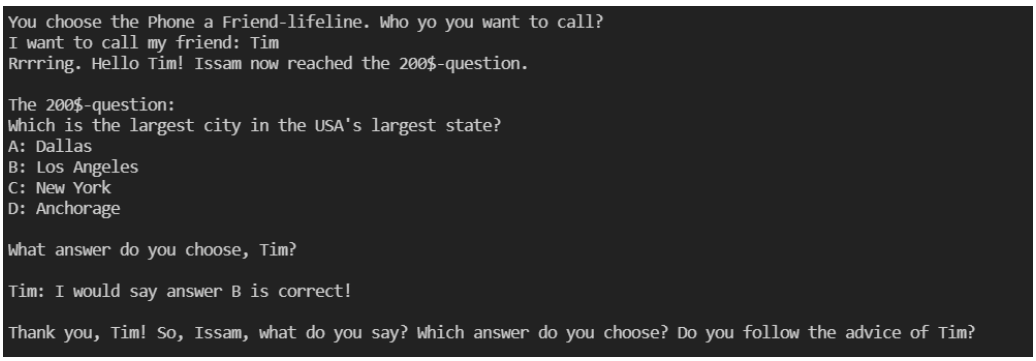
- lifeline():

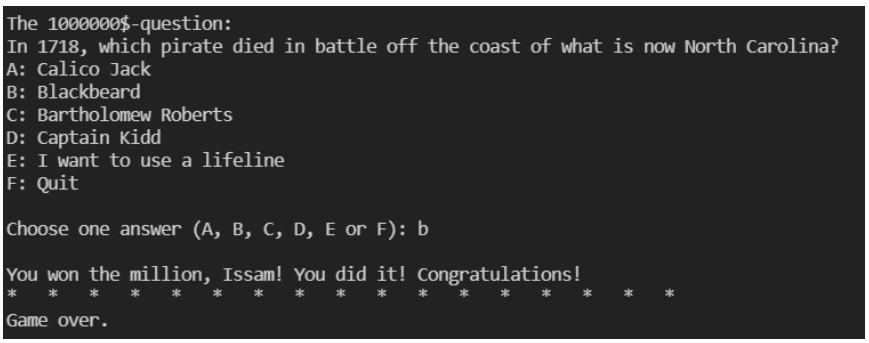
  Here a counter for the lifelines is enforced, as each can only be used once. Also it initiates the chosen lifeline.

- For each lifeline a function is implementd which ensures the execution of the specific lifeline.

I had some difficulties implementing the code:
At first, I had problems initating the linked list. It didn‘t work. So I did some research on tutorials about how to code a quiz. Most of them used for-loops. In the end, I replaced methods with functions. The simply linked list started working well. Another difficulty is incorrect input. In my code there are plenty possibilities for incorrect input. You have to be very precise and careful. Every input of the user has to be processed correctly.







3. Conclusions

What I learned:

A program like this can become large very fast, you need to know ways to keep it as compact and efficient as possible.

Also I learned why „software engineering“ is called like that. It is as if you are an engineer constructing a buidling.

The game was a very good exercise to improve my skills in coding linked lists.

Features I would want to add in the future:

- more dialogues
- more questions which then will be randomly chosen
- a high score table
- build a graphical user interface