C#-programming project – simulating a Slot Machine

This project concerns a somewhat larger and more complex C# program compared to what we have seen so far. The program is supposed to be able to simulate a classic gambling machine: a **Slot Machine**

Project Details

An introduction to Slot Machines will follow later. First, some details about the project are given:

1. **We work on this project TODAY**, and today only. We start at the beginning of class, and the project ends at the end of class today.
2. **There is no hand-in of a program or report**. This project is only supposed to be a "learning tool", to assist in the learning of the curriculum. In the next class, a "walkthrough" of a solution for the project is given, but you do not need to hand anything in to the teacher at the end of the day.
3. The project consists of a number of tasks you need to solve, in relation to the given program. The first tasks are (hopefully) fairly simple, but the tasks will increase in difficulty. The last tasks are quite hard even for the fastest students… Be aware, however, that **the goal is not to finish all the tasks**; the goal is to work concentrated and efficiently on the project during the day, and get as far as possible. Hopefully, at the end of the day, everybody will have learned something , even though some got further than others.
4. You are encouraged to **work in pairs of two!** It is much better to have somebody to discuss and interact with, than just sitting all alone with the project.
5. There is absolutely **NO POINT in trying to cheat** in any way, like Googling for code, copying code from other groups, etc. Since there is no hand-in and no marks, there is nothing to gain from cheating other than cheating yourself from learning something new…
6. **The teacher is available for help all day**. As usual, I will not monitor you closely, but if I get the feeling that somebody is doing something else than working on the project, I might take action…
7. Working on the same project for a whole day is tough! You must **manage your time yourself**, and take some breaks when you need them. Again, if I get the feeling that some are spending more time on breaks than on the project, I might take action as well…
8. Again, remember that **the goal is to LEARN**. So don't be too frustrated if you spend a long time on e.g. a single task, if you are learning something by solving it. This is not a competition about solving most tasks. However, if you are stuck and don't know how to proceed, then ask for help!
9. There is no "I'm done, can I leave early?" option for this project! The project is open-ended to a degree where it is highly unlikely that anyone will finish all tasks. So everybody works until the end of class.

Introduction

A **Slot Machine** is probably known by almost everybody, but in any case, the basic operation of a Slot Machine is as follows:

- A Slot Machine contains three wheels, each containing a number of different symbols, like a star, the number 7, the word "BAR", etc.
- For a certain small amount (e.g. 1 krone), the player is allowed to make the wheels "spin". This is done by pulling a lever or pressing a button.
- The wheels spin for a while, and end up showing three symbols, one on each wheel
- Certain combinations of symbols now pay a prize to the player. The specific combinations and prizes vary from machine to machine, but in any case, the prize (maybe 10 kroner or even 100 kroner) is paid to the player. Often, the player will use the winnings to keep on playing…

This is the basic operation of a Slot Machine. You can find Slot Machines in a casino, a bar, etc.. Almost all on-line casinos also offer various Slot Machines. These will obviously be software simulations.

The provided C# program

This project also concerns a software simulation of a Slot Machine. Fortunately, you are not supposed to write all of the software – a C# program containing a basic Slot Machine simulator has been prepared for you. You can download that C# program from the class website; it is called **CasinoSimulator**. You can find it in the row marked "C# Programming Project #1", under Projects.

The program is as mentioned somewhat larger and more complex than what we have seen in class so far. An overview of the program follows below. Note however that it is only an overview; understanding the details of the program is actually one of the tasks you have to do yourself!

The program contains three classes: **Program**, **SlotMachineManager** and **SlotMachineSimulator**

Program: This class does as usual only contain the ***Main*** method, which is the method that is executed when the program starts. It is quite simple, and essentially just creates a **SlotMachineManager** object and calls the method ***RunSimulation*** on that object.

SlotMachineManager: This class is mainly responsible for getting the actual simulation up and running, and to communicate with the player, e.g. asking the player if (s)he wants to play again after a single game. The primary method in the class is ***RunSimulation***, which starts the slot machine simulator, adds some credits to the simulator, and then lets the player play. There are some private helper methods in the class as well.

SlotMachineSimulator: This class contains the actual slot machine simulation. It is a quite large class, with many instance fields, constants, public and private methods. The three central methods are:

- ***Spin***: Perform a single spin of the machine. This means that all three dials spin. The winnings (if any) are calculated, and added to the credits in the machine
- ***SpinDial***: Spins a single dial. The result is according to the probabilities for each of the symbols, as defined by the constants in the class
- ***CalculateWinnings***: Given the symbols on the three dials, this method calculates the winnings according to the winning table. Again, the specific winnings are defined by the constants in the class

There are obviously more details in the classes, but you will explore them in the tasks.

Tasks

This section contains the tasks you should perform in relation to the program. The tasks get increasingly difficult, so you can probably not complete them all. The tasks fall into three categories:

- **Read and understand the program**. Here you are given some questions about the structure of the program. Answering them requires that you look inside the given classes, and see if you understand how the code works.
- **Modify the program**. Here you are supposed to make some fairly simple changes to the program. However, the changes do get increasingly difficult. It is very important to test the program after you make a change, even small changes. If you change many things at once, and the program suddenly does not work anymore, it can be very hard to figure out what exactly caused the problem.
- **Add new functionality to the program**. Here you are asked to change the program in a significant way, which will probably require that you add new code to the program, or change the existing code significantly. Again, be sure to test very often… Some of these tasks will definitely be quite difficult to solve.

Read and understand the program

The **SlotMachineManager** class:

1. How many instance fields does this class have?
2. How many public methods does the class contain?
3. How many private methods does the class contain?
4. How many parameters does the method ***AskPlayerToPlayOrQuit*** take?
5. How many parameters does the method ***AskPlayerToEnterANumber*** take?
6. Which method in the class contains a repetition statement?
7. What is the purpose of the repetition statement?
8. Under what conditions can you exit from the repetition statement *(Hint: read the comments)*?
9. Why is the local variable ***playAgain*** set to **true** just before the loop starts?
10. Would it be possible to use a for-loop instead of a while-loop?
11. In total, how many times does the method ***RunSimulation*** make methods calls on the **SlotMachine-Simulator** instance field?

The **SlotMachineSimulator** class:

1. How many instance fields that are NOT constants does this class contain?
2. What is the purpose of the constants starting with ***prob***…?
3. What is the purpose of the constants starting with ***winning***…?
4. Do you think these constants have good names? Can you think of better names?
5. When you create a new **SlotMachineSimulator** object, what is the initial value of the credits instance field?
6. Where in the class code is the number of credits left in the machine changed?

Modify the program

The **SlotMachineManager** class:

1. It is very easy to run out of credits quickly, when you only start with 10 credits. Modify the program to start with 25 credits instead.
2. Many players forget to keep track of the number of credits left. Inside the while-loop in the ***RunSimulation*** method, add code that prints a warning (like e.g. "WARNING! You have less than 10 credits left!"), if the number of credits gets below 10.
3. The class contains a single instance field ***theSimulator***, which is of the type **SlotMachineSimulator**. However, since the instance field is only used within the method ***RunSimulation***, we can make it a local variable inside that method instead. Make this change: delete the instance field, and instead create a local variable of type **SlotMachineSimulator** inside the ***RunSimulation*** method. For convenience, you can call the local variable for ***theSimulator***, so you don't have to change the rest of the code. Is the call ***theSimulator.Reset()*** now necessary?

The **SlotMachineSimulator** class:

1. The message that gets printed after a spin always report the winnings in the same way, like *"You won X credit(s)"*. This is a bit boring… Modify this, such that the winnings is reported like:
    - When winning nothing: *"Sorry, you did not win anything"*
    - When winning 1 credit: *"You won 1 credit"*
    - When winning more than 1 credit: *"You won X credits, congratulations!"* (X should of course be the actual number of credits won).
2. In the method ***CalculateWinnings***, there are five cases that can produce a winning – that is why the if-else statement has five if-statements. For the last two cases, we use the private helper method ***CountSymbols***, since it is easier than writing a logical expression in those cases. However, we could use that helper method in all cases. Change the code to use ***CountSymbols*** in all five cases, not just the last two *(Hint: The three first cases cover the situations where all three dials show the same symbol)*
3. A code reviewer finds that the method ***Spin*** contains too many lines of code, which makes it hard to understand. The reviewer suggests to move some parts of the code into separate methods, which are then called by ***Spin***. The reviewer suggests three new methods:
    - ***PrintMessageBeforeSpin***: Print the message that is printed before the dials spin
    - ***SpinAllDials***: Spin all three dials
    - ***PrintOutcomeOfSpin***: Print the message that reports the outcome of the spin to the player

    Follow the reviewers idea: make the three new methods as described above, and call them from the ***Spin*** method.

Add new functionality to the program

NOTE: Since this is meant to be the hardest part of the project, the level of detail is no longer so high. You will not be told in detail what to do and where to do it – instead, the new functions are described in more general terms, and it is up to you to work out the details. If you are in doubt about how to proceed with a task, you may need to study the given code more closely, to be sure you understand how it works.

1. Instead of always starting with a fixed number of credits, it would be better to ask the player how many credits (s)he wants to play for. There is a method ***AskPlayerToEnterANumber*** available in the **SlotMachineManager** class, which could be useful for doing this. Change to program to ask the player for the starting number of credits.
2. For testing purposes, it will be very useful to be able to run a simulation where many spins are done without any user interaction. Therefore, a method ***RunLongSimulation(int noOfCredits, int noOfSpins)*** is needed. A call like ***RunLongSimulation(1000,5000)*** will thus set the starting number of credits to 1000, and then perform 5000 spins. Add such a method to the **SlotMachineManager** class, and run some tests. Hopefully, the runs can give you an impression of how much the slot machine pays back in winnings.
3. When you run a long simulation, it is somewhat irritating that the program prints messages for every spin. Update the ***RunLongSimulation*** method to include a parameter that specifies if the simulation should run in "silent mode" (no messages printed during the simulation), or "normal mode". Of course it should be possible to see the end result of the long simulation (the number of credits left in the machine), even when running in "silent mode".
4. Add functionality to keep track of detailed statistics of how often each winning combination came out during a long simulation. The statistics should be displayed on the screen after a long simulation has been performed, like "The combination # # # came out 259 times out of 10000"
5. If you are good at math, you can calculate that with the given probabilities and winnings, the slot machine will on average pay out about 96 % (try to run some very long – at least one million spins – simulations, and see if you get close to this number…). So in the long run, players lose about 4 %. In order to make the machine more spectacular, the producer decides to include a fourth symbol on the dials: a star (*). If the machine shows three stars, it will pay out 10000 (ten thousand) credits! However, the probability that a dial shows a star is only 2 %. Compared to the existing machine, the 2 % are "taken" from the probability for the "7", such that the probability for a "7" is now only 8 %. Change the program to include the fourth symbol as specified above, and see if you can see any effect on the winnings (you might be surprised…).