Assignment A1

Due Date: February 5

Purpose

This project is meant to help you refresh your memory regarding object oriented techniques in C++, such as inheritance, virtual functions, and polymorphism.

Problem

Gambling is one of the country's greatest pastimes, now more than ever. One very popular method of gambling involves slot machines, once called one-armed bandits for the lever that the player pulled; now one simply pushes a button.[1] There are many different types of slot machines, but they all work in the same general way. A player inserts one or more (virtual) coins into a (virtual) slot, then presses a button to initiate play, in which several reels of "lucky symbols" spin around until they finally come to rest in some configuration. Depending on which symbols are visible on the front of the machine, the player may receive a payoff. But probably not.

To entice players to continue to feed coins into the machines, casinos take advantage of human psychology. They program the machines to give out lots of small prizes (1, 2, or 5 coins), knowing that the player will invariably feed all of these back into the machine hoping to win the jackpot. Jackpots are quite rare, but the machines are programmed to give these out from time to time too, so that players will feel they have a chance to hit it big. Casinos cluster hundreds of machines in a single room so that players will frequently hear others win and, occasionally, hear somebody win a jackpot. This keeps the player interested, and the casino rich.

You have been hired by ABCDE Corporation (Awesome Bob's Casino and Donut Emporium) to write a slot machine program. The program will allow a player to play a few different slot machine versions, with different payouts. You can try out a classic 3-reel (free) slot machine game at

`https://www.freeslots.com/Slot3.htm`

as just one example. You can watch your virtual balance go down the drain.

Input

The program should first prompt for the number of quarters (credits) to start with. Since the machine is run by software instead of constrained by hardware, there can be many built-in options. Your program should thus prompt the player if 3, 4, or 5 reels are desired. This stays constant for the remainder of the game. At each round, the machine should prompt for the number credits to play. The winnings are based on the number of reels and the number of credits played. Three symbols in a row is all that is needed to win; if playing more than three reels, matching more than three symbols yields higher winnings. Subtract credits from the total at each round; the program should stop and/or ask for more credits when the balance is 0. Credits should be added when the player wins. Finally, when the player inputs a 0 for the number of credits to play, the game should stop.

Output

The program should first display some sort of table showing how much can be won in terms of the number of credits played. For example, if there are three hearts, then the winnings might be

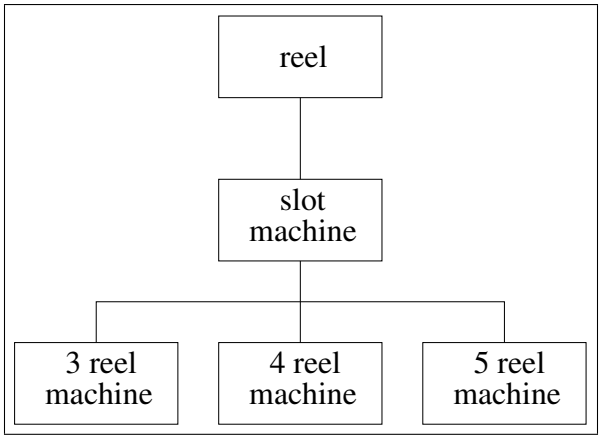
[1] What fun is that??

3× the number of credits played. Keep this simple; see Wikipedia or other online source for some guidelines as to winning percentages. The key at any casino is to have lots of small winners and few big winners. This keeps players coming back hoping to be the big winner.

After each round, the program should display the symbol on each wheel when it comes to rest. The winnings or a message stating a loss should be displayed at the end of each round. The current number of credits should be displayed as well.

Specifics

You should use C++ object oriented ideas as much as possible. To incorporate inheritance and polymorphism, you **must** use the design shown below. It may not be how you might design the game, but this will force you to think about how to write the program with as much code reuse as possible, and to write code to someone else's specifications (like you will do in industry). In other words, you want to use the power of inheritance so that the base class (reel) functionality is used as much as possible in the derived classes.



- To simulate the reels spinning, use a random number generator. The function `rand()` is sufficient for this purpose. Make sure you "seed" it properly so you don't always get the same set of numbers.
- The "lucky symbols" should be represented by Unicode characters. Choose six or so symbols from `https://www.utf8-chartable.de/unicode-utf8-table.pl?start=9984`. You do not have to write a graphical interface or show the reels spinning or anything fancy like that. Note that a reel may have a multiple number of the same symbol, which increases the odds of getting a match with that symbol.
- The payoff should be more-or-lessrealistic: the player should win occasionally. The amount of the winnings should be (non-linearly) higher the more credits are input and/or a different number of reels is played. More succinctly: the easier it is to win, the lower the payoff should be. This can be programmed as follows: If a wheel has several hearts and only one star (these are two possible Unicode symbols), then there would be a lower probability of getting three stars (on a 3-wheel machine) than three hearts. Thus, if the player got three stars, she would get a higher payoff than if she won with three hearts. As noted above, see online sources for more payoff information.

- Your code should be adequately commented. Be especially careful in commenting how all of the classes "fit" together.
  - Include an introductory comment as specified in class.
  - Each class should have a comment describing what it does in general.
  - Every variable and data member should be commented.
  - Every function/method should be commented with a short general description of what it does, a description of any parameters (which could mean "None"), and a description of the output/result. This means that each function/method comment should be at least three lines long.

Notes

- The key to a good grade is to reuse code as much as possible. If you find you are writing almost the same function three times, for example, then it's time to rethink your approach. Inheritance and virtual functions can really help here.
- You can earn up to 90 points if you do not use virtual functions. You can earn up to 100 points if you use virtual functions that are non-trivial and the number of reels is not stored anywhere within the class structure.
- The odds/methods of winning are not that important, as long as your approach is reasonable. We are not creating a true slot machine.
- You are free to use any C++ features and libraries, as long as they are standard in C++17.
- Turning in:
  - If you're writing one file, name your file *lastname*`A1.cpp`.
  - If you're writing multiple files, name your files *lastname*`Main.cpp`, *lastname*`A1.cpp`, etc, as in `gousieMain.cpp`, `gousieA1.cpp`, etc. Zip them together using the same convention: *lastname*`A1.zip`, as in `gousieA1.zip`.
  - Turn in the project via Canvas before 11:59:59 PM on the due date.
  - A printed version of your source code (yep - actual paper!) is due at the **beginning** of class on February 6^{th}.
  - Write or print and **sign** the Wheaton Honor Code Pledge on what you turn in: "I have abided by the Wheaton College Honor Code in this work."
- This may seem confusing/complicated/arduous, but it is really not a very long program. Take some time to think things through before beginning to code.
- Remember to save all of your work until your project is returned.

Lotteries are for people who are bad in math.
– Unknown