CS 1337 – Homework 1 - Snake Game Extensions

Dr. Doug DeGroot's Class

Due: Wednesday, June 9, 2021, by midnight

How to submit: Upload your homework to the eLearning site (***only*** the .cpp file). Don't zip the program. The project must compile on either Code::Blocks or MS Visual Studio 2015 (or later).

Note: You must work alone on this homework; do not collaborate with other people.

Online Help: I'll try to provide online help through office hours using MS Teams several times to answer any questions and address any problems you might experience. However, there will be no help provided the day before the homework is due. So, don't procrastinate.

Objective: Modify and extend the original SnakeGame program in the following ways:

1. Continue to improve the code wherever you think it needs to be improved for clarity or comprehensibility. This is essentially a continuation of Homework 0. Show me what you know about good programming structure and style. (You have probably already done most of this work in HW0.)
2. Check for possible bugs: Is there a bug in the board drawing code that sometimes makes the snake disappear? If there is, find it and fix it. There may be other bugs in the code; find any and fix them.
3. The Logic routine is too complicated for easy understandability. For example, it addresses multiple, separate concerns. Fix this by *separation of concerns*.
4. The snake growing part of the Logic routine is poorly written – unclear variable names, no commentary, and obscure logic. Fix these issues.
5. Add the capability of having more than one fruit on the board; use max values of 5 to 10 fruits.. Create some variable and set it to a value of 2 to 10 or more, say, to represent the number of fruits that should be on the board. Don't hardwire the number of fruits; instead, ask the user at the beginning of the program how many fruits to create. Experiment with a number of different fruit values.
6. Each time the snake eats a piece of fruit, generate a new piece of fruit at some random location on the board. But be careful where you generate it.
7. If the user fails by running into the snake's tail, don't just quit, but instead report this fact to the user and then and only then quit the current game.
8. Add an option to let the user play another game when s/he loses rather than just exiting the program when the user fails.
9. Add an input option P that pauses the game and waits for the user to hit any key before proceeding.
10. When the program first starts, print out some instructions for using the game to the user.
11. Add an option to the game that makes the player lose the game if the snake hits a wall, but set the default action to die only when running into your own tail.
12. Make the input routine case-insensitive.
13. Add any other features you think are valuable/interesting and note in the comments at the top of the program what you added and why.
14. Use good debugging techniques but leave the debugging code in the program when you submit it; just comment out the debugging code or, preferably, use Boolean switches to turn them off. Make your program talk to you. This is important.
15. As the user's score increases, speed up the program some incremental amount. The more Fruit eaten, the faster the snake should begin to move and the longer the tail will grow. Reset this speed when a new game begins.
16. Allow the user to enter + or – characters that will speed up or slow down the snake's speed. Or perhaps auto-increase the snake's speed and tail length after each piece of fruit eaten.

17. Show the user's current score, the number of fruits on the board, and maybe the high score ever.

Annotations for the code:

1. The main function can be at either the beginning or the end of the program. I don't care which.
2. Add comments at the top of your main.cpp file to include your name, the name of the program, a changelog, and notes on what changes and improvements you made to the program and when. (For example, list what bugs you fixed, what new features you added, what code you improved, etc.
3. Add a "notes:" section and list any ideas you have for future improvement.
4. Comment your code effectively, as we discussed in class. Use descriptive variable names everywhere and good code structure so that the code becomes as self-documenting as possible. But do use commentary to improve readability and comprehensibility.
5. You absolutely MUST use consistent indentation and coding styles throughout the program. Failure to do so will result in a loss of 5 points. And just bite the bullet and use Google-style formatting! 😊 Reparse your code before submitting it.
6. Don't use advanced, complex, esoteric C++ functions. Per the article we read, write dumb code, so dumb that you think even ***I,*** Professor DeGroot, will easily understood your code.
7. If the program does not compile without error or does not work at all, or works but works incorrectly, at least 10 points will be deducted.
8. No late submissions will be accepted – period (except for my one-hour tolerance if *absolutely necessary*). Please meet the deadline.
9. I will help with your code during my online office hours, but only if you have first added debugging code to your code.
10. MAC USERS: For this program (only), you will probably need to use the *ncurses* library.

An Example Program Header:

Here's how you should think about structuring your program's header. Note that this is just an example. But please follow the given format shown below.

```
// SnakeGame-ME - version 02
// Doug DeGroot
// created: 12jan19
//
// comments:
//  from original code by N. Vitanovic
//  original code location: https://bit.ly/29WZ5Ml
//  I cleaned-up/improved only part of his code below. I want
//                        *you* to clean-up/improve/fix the rest.
//  Note: There are surely several bugs and/or intentional errors
//                        in this code. Can you find and fix them?
//
// changelog:
//  12jan19 - cleaned up the code, added more commentary;
//     seeded the random number generator so fruit would spawn
//     in different places on startup;
//     added q and x keys in input to quit;
//     added + and - key input processing to speed up or slow
//     down the snake's movement
//  14jan19 - added code to always show location of snake's head
//     added code to let the user play again if desired
//     I reset the snake's tail-length to 0 in setup
//  17jan19 - added a "pause" command to the input logic
//
// notes:
//  Maybe add a single step command.
//  Make tailX and tailY into a 2D array.
//  Parameterize the option for detecting the snake's hitting a wall.
//  Does the snake sometimes disappear within a wall? Or what about
//      fruits? If so, what's wrong with the code?
//   Do we sometimes generate two or more fruits at the same location?
//     What's the best way to handle this? Detect and Present? Allow?
//     Allow but use special character to note multi[le fruits in a single location?
//     Or what?
//
```