COMP161 - Project 1 - Hangman

Spring 2014

Your first project is to implement a Linux CLI version of the classic word guessing game Hangman.

The Game

Hangman is a simple game[1]. The player must figure out a word by guessing one letter at a time. If, over the course of the game, they guess seven[2] letters that do not appear in the word, then they lose. Typically, each wrong guess corresponds to a part of the picture of a hanging stick figure: gallows, head, body, left arm, right arm, left leg, and right leg. The player gets to knows how many letters the unknown work contains. When they guess a letter correctly, they get to see the location of each occurrence of that letter in the word.

[1] `http://en.wikipedia.org/wiki/Hangman_(game)`

[2] or some fixed number

The Program Requirements

Your program must provide the usual elements of the game:

- Visual representation of the number of wrong guesses made/left[3]
- Visual display of the letters guessed so far
- Visual display of the target word with correctly guessed letters filled in.
- Case-insensitive words.[4]

[3] does not need to be a gallows picture or any kind of picture

[4] The letter h and H are equivalent as far as the words are concerned

You're free to play with the game UI as long as the above requirements are met. The code itself must be well-designed. Each procedure should be properly documented and tested.

Logistics

We'll dedicate two labs and some class time to discussing issues related to the project. The expectation is that you are putting in continual, consistent effort in to this project during that time. This effort needed always mean your programming. It can be problem analysis and program design tasks as well.

Due Dates

1. **Due by end of lab on 3/25** UI Mock up showing a complete run of your program.[5]
2. **Due by class on 4/1** Complete project. Submit as *proj1* via *handin*.

[5] Like `http://en.wikipedia.org/wiki/Hangman_(game)#Example_game` but showing *exactly* what your program will show

Grading

Your project will be evaluated on the following criteria:

- Completeness,correctness and evidence of an iterative development process

  Ideally, your program is a complete working version of the game. To get to that point you should be working through a series of iterations, where each iteration produces a high-quality, but incomplete, version of the game. *This means that incomplete programs should clearly be a finished version of something and not an unfinished program.* This also means that the finished product *should always compile, run, and do something.*

- Usage of good program/procedure design principles

  We have a standard process for designing, documenting, and developing procedures. Lack of documentation and proper unit tests will result in a loss of points regardless of the level of completion or apparent correctness of the program itself.

- Usage of good code style

  Ugly code that works is still ugly code. Human beings read more C++ than compilers do and human beings struggle when reading ugly code. Don't write ugly code. Properly indent your code and follow our standard conventions for writing C++. You should also ensure that your code will not line wrap when printed.

Challenge Problem

If you're looking for a little extra challenge, then consider the following additional program requirement:

- Allow the user to pass a positive valued integer to the program from the command line. This integer indicates the length of the target word. So, if the user launches the program with the input of 5, then the program finds a 5 letter word for them to guess.

- Allow for a command line argument that sets the number of allowable wrong guesses.

- Add some other cool feature to the program.

Completing the challenge problems will not result in bonus points. Do it for the experience and the challenge.