

SOFTWARE PROGRAMMING

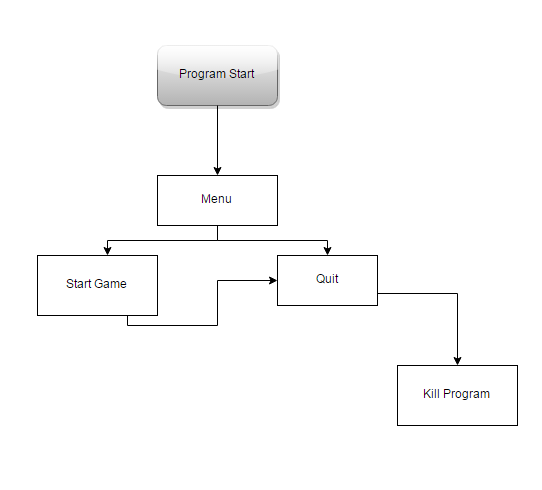
HANGMAN

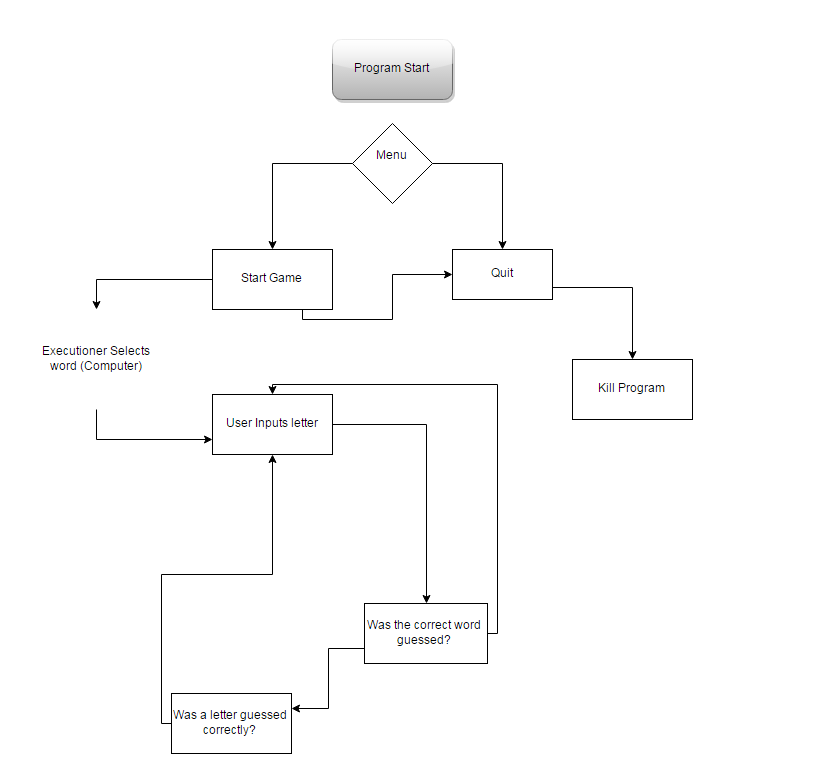
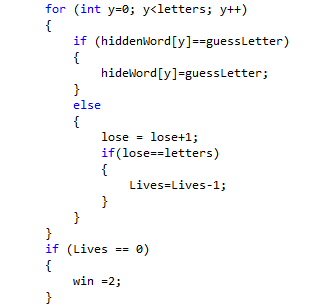


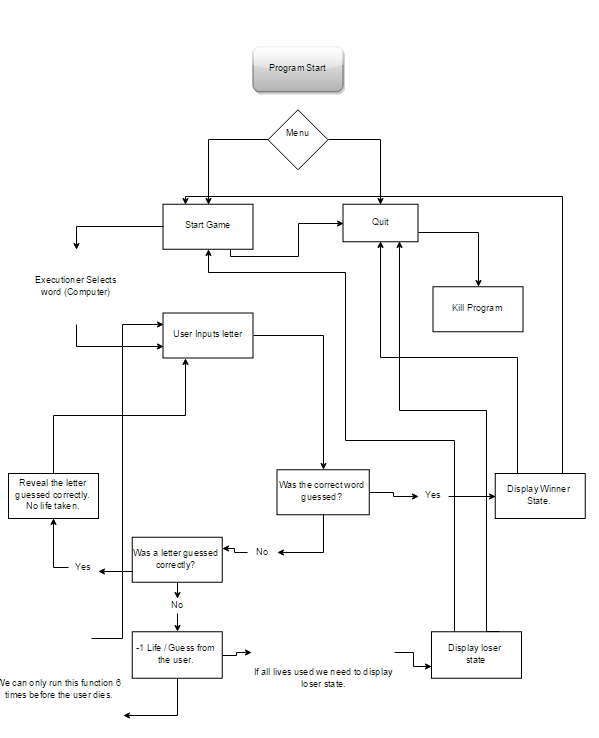
January 17, 2015

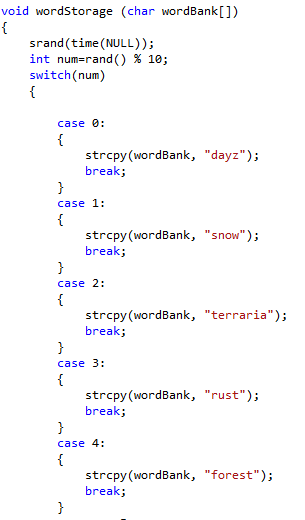
Steven Reynolds

In this report I am going to talk about the C++ project Hangman. I am going to go through the initial problems that I had and reflect these in a Initial problem design chart, and then I am going to talk about how I managed to get passed these difficulties by displaying them in a Top Level Flow chart that shows how functions are working together with one another by displaying where statements are storing information.

After doing some extensive research into coding a hangman game, I created an Initial Problem chat that would allow me to plan out how I wanted to create this program. First of all I created the fundamental basics of a game, this included a Menu, the game, and a way for the user to quit out of the program. Trying to do this in a manner that is quick and easy for the user is the hardest part because you want the user to be able to navigate around the program quickly and easily. In most games when you want to quit or return back to the main menu you normally have to confirm yourself a couple of times just in case you miss-click or select the wrong button however I am using a different system that just allows the user to easily quit by using a function called Exit(It almost works the same as the die function in the programming language PHP) The good thing about exit is you just need to return a variable/number and it works for example when the user quits the game it will automatically run (exit(1)). What this is doing is quitting the game and returning the value 1 as I specified in the brackets.

After reviewing my IPC, I then had to break down the branch of how the start game was going to work. This was probably the hardest part to do because I am using an enumerator for the menu system and trying to exit out of that was very hard. As you can see in the diagram, the start game branch now has a lot of different side branches coming off it that all need to work together without breaking and by doing it step by step actually helped me solve the initial problem. On the second IPC to the left we can see an alpha stage of the game of how the statements are linked together but we still need to be error checking the words and also deducting the amount of lives (or guesses) that the user has left. Now this is very simple, as you can see on the right the code snippet shows if the user guesses a hidden word it will subtract Lives=Lives-1; By doing this we are actually using multiple functions/variables in this procedure by using the variables lose, letters, and lives and simply by changing the values of them we can correctly display the users lives, At the bottom of the snippet you can see an if statement that works out if the lives value is equal to 0 we then display the lose screen. However we are still not error checking enough to tell the program once this has happened do this or go to this state. We now need to do a full IPC with error checking and testing to make sure everything is displayed correctly to the user and making sure that the word and or letter is revealed when the user guesses correctly and making sure they do not lose a life.

In comparison with our first IPC chart, we can see that we now have a lot of statements and a lot of error checking going on, at the bottom left of the IPC is the error checking for user lives, if the user has more than 0 lives or guesses then carry on, if not then display the error screen.

With everything now implemented I can go into detail on what everything does, When the user starts the game, the computer will choose a word out of the function (void wordStorage (char wordBank[])) which has a random number generator inside of it. The user will then come to execution screen in where they can input there guesses which then gets fed through to a function that checks if the user has guessed the word correctly, if not has a letter been guessed correctly, If it has carry on, if not deduct a life and go to user input again.

The way the ASCII art is being handled is quite cool because I have them in a void which has its own switch statement. Inside I have cases from 0-6 and every time the user gets a letter wrong the case will go down (6-5-4-3-2-1) The reason I did this was because I wanted to keep the case switch statements for the menu and the program clean, as that was going to be very code heavy and having ASCII art in the way would confuse me on what I am currently working on. On the other hand I did include the ASCII art for the main menu and the win and lose states inside the program simply because this was a last minute job.

In the end, I believe I have created a basic/intermediate hangman designed game which has good design, good programming and commenting for future reference. I feel that if I pushed myself further I would of like to of made some levels so you could choose how hard you wanted the words and try to incorporate something more fun that would keep the user entertained for more, not just looking at console lines all the time.