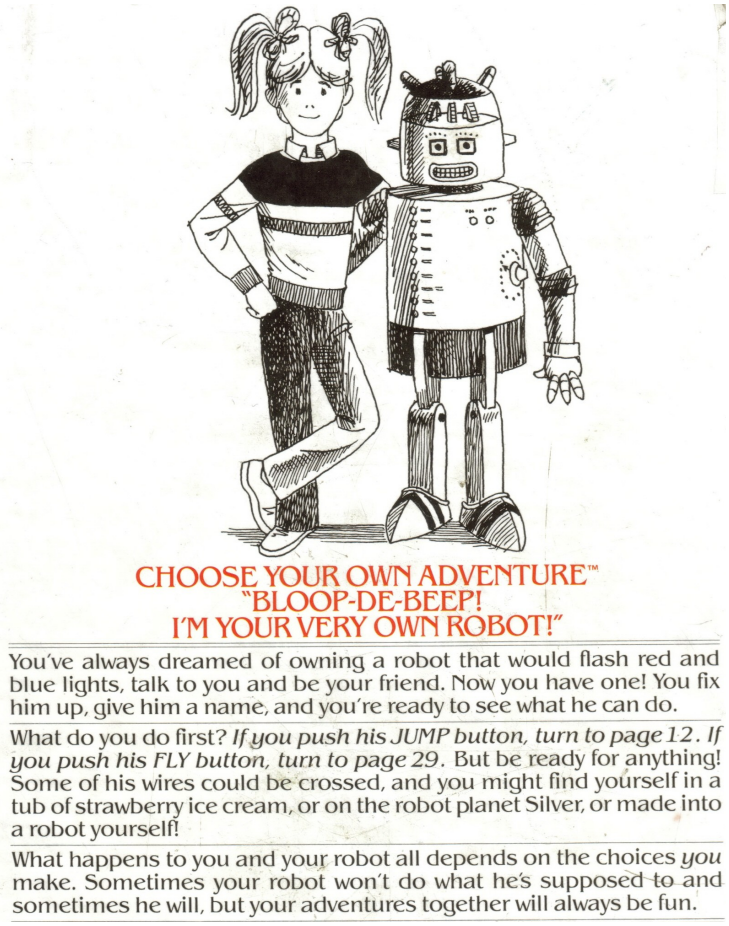
CS101 Homework 2: Choose Your Own Adventure Game

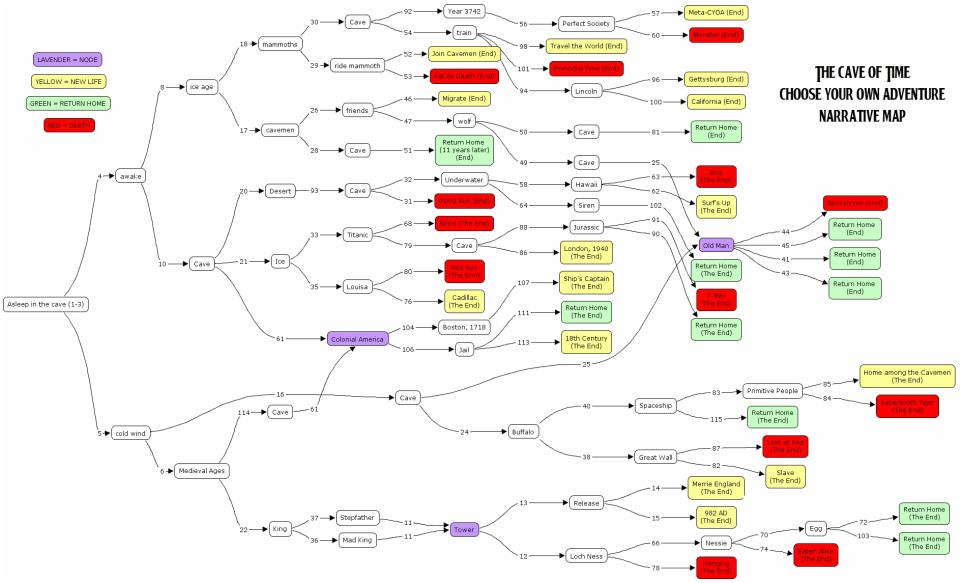
Prof Tejada
Design Document due: 11:59pm Friday, February 1th
Game and Report due: 11:59pm Friday, February 8th



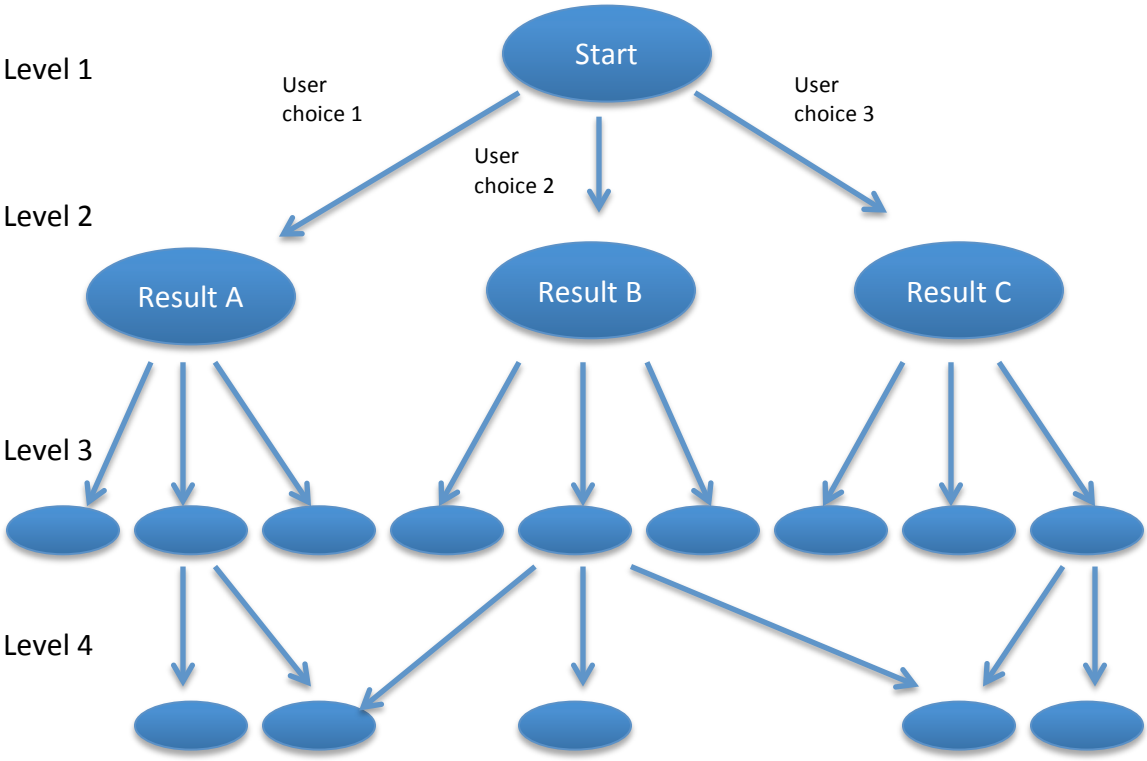
1 Introduction

When I was a kid I enjoyed reading Choose Your Own Adventure books, such as, this one called "Your Very Own Robot." I see now that it was helping me to think like a computer scientist, in terms of control structures -- branching and loops. The story starts out on page 1, but at the end of the page the reader can make a choice about what the main character should do next. There can be several options given to the reader of what decisions the main character can make. The reader makes a choice by turning to the page paired with the main character's action, e.g. "If you push his JUMP button, turn to page 12. If you push his FLY button, turn to page 29." I could read the book over and over again and have a different adventure each time.

This is a graph of all the player choices possible for the book "The Cave of Time." Every branch in the graph represents a user's choice, and every bubble is the result of the user's actions.



For this homework you will create your own Choose Your Own Adventure game, where your story must be at least 4 levels deep – meaning the player can make 3 choices between the start and end of the game. This graph below represents the program flow for a game. Each bubble or node with out-going arrows represents a part of your program that will ask the user for input, then based on that input decided which bubble or node to transition to next. Each node without out-going arrows represents an end of the game. When the game ends the user should be given the opportunity to play your game again from the beginning. Your game must have at least 15 different nodes and at least 20 different arrows or user choices. Unlike a Choose Your Own Adventure book your game can have some randomness. Your program should allow for randomness somewhere in your graph, so that the player may have the option of "rolling the dice," allowing a random choice by your program decide where the player goes next.



This programming assignment should be performed **INDIVIDUALLY**. You may re-use portions of code from previous examples and labs provided in this class, but any other copying of sections of code is **prohibited**. We will use software tools that check all students' code and known Internet sources to find hidden similarities.

2 What you will learn

After completing this programming assignment you will:

1. Write a non-trivial C program from scratch by combining knowledge from previous examples and modifying them appropriately
2. Select appropriate control structures including for loops, while loops, conditionals / selection mechanisms (i.e. if statements) to solve a problem
3. Use the random number generation capabilities of the standard C++ library.
4. Combine and create the above skills to develop a "choose your own adventure" game.

3 Adding Randomness

We can just use the random number generation capabilities of the standard C++ library (or any another programming environment) through the `rand()` function. Most random number generators will produce a number that is **uniformly** distributed over a certain range. A uniform distribution means that theoretically every number over the range has equal probability of being produced. `rand()` returns an integer number in the range 0 to `RAND_MAX` (which is a constant defined in `cstdlib` which is also where `rand()` is declared/prototyped as well).

One issue with `rand()` and most computers generating random numbers is that computers do only what they are told and so it is sometimes hard to be random. Most computers generate pseudo-random numbers (sequences that look random but are not necessarily). And in fact, because of this, every time you start your program you will be the SAME random sequence. To get a DIFFERENT sequence for each program run, we need to "seed" the random number generator with a different starting value. You can do this with the `srand(int seed)` function in `cstdlib`. During testing we often provide the same seed to make sure our results are similar as we update the code. When we are satisfied our code works we can provide a unique seed each run (the easiest way to do this is to pass srand() the current time which should be different for each run). Thus, as you develop your program place the following line of code as your first statement after declaring your variables.

```
srand(1000);
```

Then when you think your code is working, change the line of code to read:

```
srand( time(NULL) );
```

You will need to include `<ctime>` in addition to `<cstdlib>` to use this functionality.

4 Requirements

Your program shall meet the following requirements for features and approach:

1. Prompt the user to input their choices.
2. Receive input from the user.
3. Check that the input is a valid choice.
4. Use the input to decide the next statements to be executed.
5. Add some randomness to the game by using `rand()`
6. Player can repeat the game as many times as they wish.

5 Design Document

Write down the answers to the following questions **BEFORE** beginning. You MUST do this **BEFORE** you begin to program or write code.

1. What data needs to be maintained for a single game? What type?
2. What type of user input do you need? How will your program use this input to decide what statement to execute next?
3. What data needs to be maintained for the total player experience (i.e. across several games)? What type?
4. Is the number of iterations for the total Choose Your Own Adventure known before entering the loop? What kind of loop should then be used?
5. Where will you add randomness to your game?
6. Draw your own graph to represent the program flow for your game. What are the nodes? What are the user choices?
7. Blackboard submission (Due Friday February 1, 11:59pm)
    a. Zip your Design Document and submit the zipped file to Blackboard.

6 Procedure

Perform the following.

1. Using your answers to the problems above implement your choose your own adventure game as a C++ program, in a file named `game.cpp`
2. Checkpoints: If you are new to programming or having trouble getting started try to write the code in stages and be sure each stage is working:
    a. Stage 1: Start by writing a program to prompt the user for the necessary input values and the read them in from the keyboard into appropriate variables…Check your work by displaying them back to the user.
    b. Stage 2: Wrap the code from the previous step in some kind of loop that will continue asking the user for input if they did not enter a valid choice.
    c. Stage 3: Write code that will use the input to decide what statements should be executed next.
    d. Stage 4: Write the code that simulates a random choice.

e. Stage 5: Wrap all your code in some kind of loop to allow the user to play your Choose Your Own Adventure story again.

3. Comment your code as you write your program documenting what abstract operation each for, while, if or switch statement is performing as well as other decisions you make along the way.
4. Compile and run your program:

```
g++ -o game game.cpp
```

7 Report

Include the following items in your report. **Be sure to have your name and USC ID on the first page at the top.**

1. Your answers from the design document.
2. Your graph of the program flow.
3. A printout of a run through your program.
4. Place your 'game.cpp' file and Report in a ZIPPED folder and submit that ZIP file. Failure to place the file in a ZIP file will lead to point deductions. Then submit the ZIP file via Blackboard.
5. Blackboard submission (Due Friday February 8, 11:59pm)
   a. Zip your Report and your C++ program file together and submit the zipped file to Blackboard.

Please also note that you must submit your code using the USC Blackboard System since the Blackboard System timestamps your submission. You should also verify what you have submitted is what you intended to submit. Please note that **it is your responsibility** to ensure that *you have submitted valid submissions, meaning that your code must run on aludra, which is the machine that it will be graded on*.

8 Late Policy

All homework must be turned in on time. Late submissions will receive severe penalties. If you submit within 24 hours after the grace period, you will receive 80% of your grade. If you submit within 48 hours after the grace period, you will receive 50% of your grade. If you are unable to complete a homework assignment due to illness or family emergency, please see the instructor as soon as possible to get an extension. A doctor's note is *required* as proof of illness or emergency. In general, when you get sick, it's best to see a doctor and get a note just in case you may need it later.

HW2: Choose Your Own Adventure Grading Rubric

Name: _____ Score: _____ / 100

Req. / Guideline	Wt.	Score	10 (Excellent)	8 (Good)	5 (Poor)	2 (Deficient)	(0) Failure
Design Document	1		Well-written answers that demonstrate strong thought processes and reasoning, includes graph	Well-written answers that demonstrate acceptable thought processes and reasoning, includes graph	Major errors in cases or end condition or no graph	Missing cases or end condition or no graph	Missing completely
User Input	2		Well formatted and works correctly, uses loop for input validation, with at least 15 nodes	Acceptable formatting and works correctly, uses loop for input validation	Wrong type of data input or poor formatting or no input validation, less than 15 nodes	Input does not work or no loop	Missing
Code for repeating game	1		Executes correctly based on user input, using loop		Executes incorrectly based on user input, using loop		Missing
Code for branching	2		Executes correctly based on user input, with at least 20 branching decisions, 4 levels deep	Executes based on user input, with at least 20 branching decisions, 4 levels deep	Executes based on user input, with less than 20 branching decisions or not 4 levels deep	Executes based on user input, with less than 20 branching decisions and not 4 levels deep	Missing
Code for rand()	2		Correctly uses the rand() function to make random decision for the user		Incorrectly uses the rand() function to make random decision for the user	Incorrectly uses the rand() function	Missing
Code Organization	1		Well-organized code (indented /readable) AND commented such that another person could easily follow what the code was attempting to accomplish as well as providing insight into your decisions	Acceptably organized code (indented /readable) AND commented in a reasonable manner to allow someone else to reasonably follow the code and provided some insight into your decisions	Poorly organized code (indented/ readable) OR comments were disorganized and provided little insight into the code and implementation decisions.	Poorly organized code (indented/ readable Verilog) AND comments were disorganized or provided little insight into the code and implementation decisions.	Completely unorganized code and lacked helpful comments.
Report	1		Well-written answers that demonstrate strong thought processes and reasoning, includes graph	Well-written answers that demonstrate acceptable thought processes and reasoning, includes graph	Major errors in cases or end condition or no graph	Missing cases or end condition or no graph	Missing completely
Late / Deductions			-20 %(1 Day Late)	-50% (2 Days Late)	-5 (Submission Instructions)		
TOTAL							