MiniProj 1: Dog Survives Evil Forest

(Classes, Objects, Matrices, and random numbers)

80 pts, Due Fri, Feb 24 at midnight

Objectives:

This lab is to shore up your understanding and comfort with classes, objects, matrices, and random numbers

Video:

This miniproject is closely tied to week 2 videos and ppts on my web site: https://www.eecis.udel.edu/~yarringt/CISC220 (yeah, I'm just gonna keep plugging it until everyone gets used to the idea that that is where I place my course content).

Partners:

This project may be done with a partner or you may choose to work alone. If you choose to work with a partner, make sure you both turn in the project, and make sure you include both names on the project. Equally, note your partner's name in canvas. Please be aware that if your partner flakes on you, you are still responsible for completing the mini project and turning it in on time.

Timeline:

Expect to put in approximately 6-15 hours outside of class and lab time on this project, depending in part on how comfortable you are with classes and objects.

Purpose of the Project:

The purpose of this project is to establish a firm foundation in creating and using basic classes and objects. For many of our future data structures we will be using classes and objects. Random number generation is thrown in as well (because you can never go wrong with a bit of randomness).

Thus I am requiring the 2 classes: Dog and Board. For this project I am giving you a good deal of structure, and a plan for developing this project. I expect you to adhere to the structure and the plan. You are allowed to add methods (helper methods, perhaps) and even modify the method parameters.

This lab is also very matrix-intensive. There are many many reasons to use matrices – aside from technically being a data structure in its own right, we use it in the implementation of more complex data structures as well.

Dog Survival Game: Description of Game

For this lab you will be using classes and objects to write a program that gets a puppy dog (Fluffy) safely through an evil forest.

There's a video here.

Your goal is to get Fluffy the dog from the random start position on the left side of the board to the random end position on the right side of the board. You must help fluffy navigate around (or through) walls, deal with potentially hidden traps, and gather food to have enough strength to make it to the other side. Every time fluffy moves up, down, left, or right, they lose 2 strength points. In addition, if fluffy lands on a trap, they will lose a random amount of strength. If, however, they land on food, fluffy will gain a random amount of strength. Your goal is to make sure fluffy has enough strength to navigate the board to get to the end on the right side.

FLuffy may break through walls if they have enough strength.

If, however, fluffy runs out of strength points before making it to the other end of the evil forest, then fluffy dies a sad and horrible death.

Your job:

For this project, there are 2 classes:

- the **Dog class** and
- the **Board class**.

I am giving you:

- The Board.hpp header file for the Board class
- Some Board methods (for the Board definition file)
- The mainGame.cpp main file:

You must write:

- The rest of the Board methods (as described below)
- The Dog.hpp (header) and the Dog.cpp (definition) files for the Dog class (as described below.

Getting Started:

Start by creating a new project in eclipse (make sure you set the compiler to be mingw on a pc, macosx on the mac).

1. File->new->new c++ Project then c++ Manage Build, and then give the project a name and MAKE SURE YOU SELECT MinGW or MacOSX for the compiler.

Make sure you have downloaded and unzipped Lab2's files from my web site.

2. Take the .hpp and the .cpp files and place them in the folder you just created, above.
3. Refresh eclipse by selecting the project you created above in eclipse (click on it) and then right or control click, and select refresh.

Note: if you still can't see the files, you may need to create each .hpp and .cpp file in eclipse, then copy and paste the code from the downloaded file to the appropriate file you just created. To do this:

4. right(control) click on the project you just created (above), and select file->new->header file to make each new .hpp header file, (for Dog.hpp and Board.hpp
5. and right(control) click on the project, then select file->new->source file to make the associated .cpp file. (for Dog.cpp, Board.cpp, and mainGame.cpp)

In C++ we separate the class definitions from the class declarations. The class declaration goes into its own header file (.hpp)

The header file should have the fields (aka properties) associated with the class, and the method declarations. It is the header file that is included in other files

For instance,for the board class, the files should be Board.hpp and Board.cpp, respectively. The Board.hpp will have the method declarations and fields associated with the board class, whereas the Board.cpp will have the method definitions associated with the board class.

6. ***Note that we have included the .hpp in the .cpp, (and not vice versa!! -Including both in both will cause compiler issues)***

The associated .cpp file contains the class' definitions for all the methods. This includes the method constructors (possibly overloaded).

Now that you have everything set up, here are the instructions for the coding part:

/***/

Board Class Methods (38 pts):

I'm giving you the header file for the board class, and some of the class definitions as well. I am giving you the constructors for this class, as well as the InitAll method and the playGame method. Your job is to write the following methods in the Board.cpp (the definition) file:

- The printBoard (8 pts) method, (as defined, below)
- The boardConfig (8 pts) method (as defined, below)
- The addFood (5 pts) method (as defined, below)
- The addTraps (5 pts) method (as defined, below)
- The moveDog (12 pts) method (as defined, below)

I STRONGLY SUGGEST you get Board.hpp and Board.cpp opened in your project, and then follow the step-by-step instructions in the initAll method in Board.cpp. I am taking you through how to approach writing and testing code in a class here. Please note that as the semester progresses, I will be slowly giving less and less information on how to tackle projects, so pay attention to how it is being done here.

`Once you are to the point in the Board class where you need to tackle the Dog class, you will need to do the following:`

Dog Class (33 pts) :

For the dog class, you should create both the header file and the definition file (Dog.hpp and Dog.cpp, respectively).

(8 pts for creating a proper Dog header file)

The dog class consists of the following fields:

- string name; // for the dog's name
- int strength; //for the dog's current strength
- int x; // the x coordinate of where the dog is currently on the board
- int y; // the y coordinate of where the dog is currently on the board

And the following methods (all are defined in more detail below this list):

- (4 pts) Dog(string n); //constructor
- (3 pts) Dog(); //constructor
- (6 pts) bool changeStrength(int amt); //changes dog's strength field
- (3 pts) void die(); //die message when strength at or below 0
- (3 pts) void printDog(); // prints out info about dog
- (3 pts) void won(); //won message for when dog is at end of evil forest
- (3 pts) void reset(); //resets dog for restarting a new game

NOTE: The dog header file (created by right-clicking on the project, then selecting file->new->header file) should contain the following line right below the `class Dog {` line:

```
friend class Board;//by making the Board a friend of this class, the
                   //Dog class is allowing the board class and objects
                   //to access all of the dog class' private fields
```

Here are more in-depth explanations of the constructors and the methods for the dog class:

- Dog(string n); (

```
/* constructor, sets the name to be whatever name gets passed in, the
 * strength to be 50 (I just picked 50 - you want another number, go for it!) and I made
 * the original coordinates be 0 and 0 because the compiler yells at me a tiny little yell
 * if I don't initialize all my fields in the constructor
 */
```

- Dog();

```
/* constructor, I made the default be fluffy, but you can make it anything you like!!
 * I set the strength to be 50 (again, I just picked 50), and I made the original
 * coordinates be 0 and 0 because...
 */
```

- bool changeStrength(int amt);

```
/*changeStrength modifies the strength property by whatever amt is (basically adding here)
 * It then calls printDog method to print out current information about the dog.
 * It also checks to make sure that the dog object's strength is >0.
 * If the dog object's strength is 0 or less, it calls the die method.
 * It returns true if the strength is still >0 and false otherwise.
 */
```

- void die();

```
/* This method just prints out a sad pathetic death of a dog message, letting everyone
 * know of the demise of the dog object.  My message has ACK!!! and judgmental statements
 * about how the user is a cruel, heartless person for letting a puppy dog die, but
 * yours can be whatever you like.
 */
```

- void printDog();

```
/* this method just prints out the basic information about the dog - its
 *name and its current strength.
 */
```

- void won();

```
/* again, just a message about the user's incredible prowess in successfully navigating the
 * sweet little puppy dog through the evil forest to the other side.  You can make it whatever
 * you like.
*/
```

- void reset();

```
/* this method just resets the strength (for me it was 50) and the x and y to 0 and 0.
 *
 */
```

```
/*****************************************************************************/
```

(10 pts for getting everything to run smoothly!!!)

If you get all this written, tested, and working, you're done!!! Congratulations!!! Savor this coding moment of victory.

Note: If a pop-up box arises that says "Errors exist, do you wish to continue?" your code DOES NOT COMPILE. Don't click continue. Fix your code. You will not receive credit for code that does not compile.

TO TURN IN:

You should turn in:

A **zip file** (named AA_BC_Lab1.zip where AA is your initials and BC is your partner's initials. If you do not have a partner, skip the second set of initials) of **code that compiles**, with the following files:

- Board.cpp
- Board.hpp
- Dog.cpp
- Dog.hpp
- mainGame.cpp
- Screenshot of your code running*

NOTE that if you worked with a partner,

- **BOTH** your names must be at the top comment section of each file
- While you may both turn in the same zip file (so AA_BC_Lab1.zip – it doesn't matter whether your initials are AA and your partner's initials are BC, or vice versa), you **BOTH SHOULD TURN IT IN!**
  - Why? This helps us if there are ever any issues with things being turned in (e.g., your partner was supposed to turn it in but their computer crashed at 11:59pm, you forgot because there was free pizza in the hallway, you turned in an old version that didn't work, etc.). This way we have back-up.

**(Why a screenshot of your running code? Because if there's an issue getting your code to run on our computer(s), a screenshot gives us enough info to know whether to meet with you to see it running on your computer. I do not want screenshots of every single part of the code running properly – just one or two shots of the running code that indicates that the code did actually compile and work).*

EXTRA CREDIT OPPORTUNITY (up to 20 pts total):

A) C++ is a middle level language (as opposed to Java, which is a high-level language). In a low level langage, there's a 1-1 correspondence between commands and what the processor executes. In a high-level language, the focus is on higher level functioning and usually no ability to manipulate memory and processor executions. C++ bridges the gap between those two levels. It is often used for systems programming (for instance, your operating system) because of this.
I told you all of that because I'm explaining why this lab has a pretty klugey interface. Yeah, not pretty. I get that. Wanna make it pretty? Up to 15 points extra credit to make a prettier interface. (You'll have to look up and import libraries, most likely).

B) Along those lines, this is a pretty basic game. What did you want for a one-week lab? But I can think of many ways to make it way cooler. Up to 15 points extra credit for coming up with some cool improvements on the game

(Note that, while each of these opportunities is worth up to 15 points, even if you do both in an astounding, breathtaking way, you can only get 20 pts max extra credit on this lab).