CS 105 (C++)

Assignment 8: Sprites

I. Overview

In this assignment, I will provide functions for rendering four different video game sprites, and you will develop polymorphic classes (one for each sprite) in such a way that all four of them can be drawn by making the same call through pointers having a single base class.

You will need to download the files `drawing.h` (here) and `drawing.cpp` (here). You will gain access to the necessary drawing functions by including `"drawing.h"` in the `main.cpp` file that you create. The following functions are declared in `drawing.h`:

```
void beginDrawing();
void drawShadow(int row, int col);
void drawSpeedy(int row, int col);
void drawBashful(int row, int col);
void drawPokey(int row, int col);
void drawErrorMessage(int row, int col);
void endDrawing();
```

Here's what you need to implement:
(Note: All of your code should exist entirely within the file `main.cpp`.)

- A base class called `Sprite` with the following members:

    - Private integer data members to store the sprite's row and column.
    - The public member function `setPosition`, which takes integers for row and column (in that order) and stores them in the private data members described above, returning nothing.
    - Public member functions `getRow` and `getCol`, which take no arguments and return the integer values of the corresponding private data members.
    - The public member function `draw`, which takes no arguments and returns no arguments:
        - When called on a pointer to the base class, this function should resolve polymorphically to a call to the corresponding function in a dynamically allocated instance of a derived class.
        - This function is not intended to be called on an actual object of the base class, and if it is, the function `drawErrorMessage` should be called (with the values of the row and column data members passed to it).
- Four classes derived from the base class, each one named for one of the original Pac-Man ghosts (with the capitalization shown here): `Shadow`, `Speedy`, `Bashful`, and `Pokey`. For each of the derived classes:
    - Define the `draw` function so that it will be properly invoked by polymorphic calls to base-class pointers to derived objects.
    - In each Derived class the `draw` function's only job is to call the corresponding draw function from `drawing.h` (passing it the base class object's row and column values). For example, `Shadow` calls `drawShadow`, `Speedy` calls `drawSpeedy`, etc.
- A `main` function to do the following:
    - Declare an array of 4 pointers to the base `Sprite` class. (Note: The array itself does not need to be dynamically allocated.)
    - Call `beginDrawing` to prepare the window for drawing.
    - Dynamically allocate one instance of each of the derived classes, assigning one of the array's pointers to point to each.
    - Set the positions of the derived class objects to the following rows and columns (respectively):
        - `Shadow`: 5, 10.
        - `Speedy`: 5, 44.
        - `Bashful`: 22, 44.
        - `Pokey`: 22, 10.
    - Call the polymorphic `draw` function through each of the base class pointers in the array.
    - Deallocate any memory that was dynamically allocated earlier. (For any call to `new` above, there should be a corresponding call to `delete`. As always, I recommend drawing all dynamic memory operations to ensure correctness.)
    - Call `endDrawing` to: wait for a key to be pressed then clean up the drawing window.

If this is done correctly, the first four sprites from the image above will be rendered (although probably with slightly different colors) in a 2-by-2 arrangement. Successful polymorphism will be indicated by the rendering of four different sprites, despite the fact that the same function call is made to all four base-class pointers.

To compile, include all necessary `.cpp` files along with the ncurses library flag as follows:

```
g++ main.cpp drawing.cpp -lncurses -o a8
```

II. Grading

The following is a list of specific assignment requirements, along with the grade value for each (out of a total of 10 points for the assignment).

- Minimum Requirements
  - The correct result is obtained (through the proper use of polymorphism) as described above.
  - All of your work must be submitted in a file called `main.cpp`.
  - This file (when combined with the given `drawing.h` and `drawing.cpp`) must compile on a department UNIX machine with the following command:
    ```
    g++ main.cpp drawing.cpp -lncurses -o a8
    ```
  - Before evaluation, your code must be submitted via `turnin`, using the following command on a department UNIX machine:
    ```
    turnin --submit dlessin a8 main.cpp
    ```
- Graded Elements
  - No class members beyond those specified in Section I can be included.
  - Proper use of derivation.
  - Proper use of the keyword `virtual` to implement polymorphism.
  - Use functions declared in `drawing.h` exactly as described above.
- Provisional Dealbreakers
  - Due to the importance of proper handling of dynamic memory, you will lose **50%** of your programming assignment grade if a valgrind evaluation of your work shows any memory leaks or errors *other than memory which is "still reachable"*. (Due to this assignment's implementation, there will be some memory listed as "still reachable" in valgrind's leak summary output. This is completely acceptable, and will not affect your grade.)
  - **However**, in recognition of the difficulty of this task, you may resubmit your work at any time before the end of the course for regrading on this portion of your score. (And of course, I'll be happy to help you find and fix any problems.)
  - Valgrind evaluation is performed as follows:
    - Prepare your executable for valgrind by compiling with debugging information on and optimization off. Use the following command for this (Note that "`O0`" is a capital letter "o" followed by the number zero.):
      ```
      g++ -g -O0 main.cpp drawing.cpp -lncurses -o a8
      ```
    - Run valgrind by prepending the following to your normal command (including all normal arguments):
      ```
      valgrind --leak-check=yes
      ```
