Arcade project

How to implement a new graphic library:

- The first step is that your class must implement the IGraphic interface.
- The IGraphic interface is contained in the graphic namespace so you can call it from outside with graphic::IGraphic.
- The interface contains 8 functions that you must implement in order to get your graphic library working with this project.
- You can find the interface file in ./lib/include/IGraphic.hpp when you are at the root of the project directory.

Here are the 8 functions and their usage:

- isThereEvent() which is a constant member function returning a boolean that tells the Core class if there is any new user event to manage, this function will be called at each game loop iteration

- getKey() which is also a constant member function returning an int which is the ID of the key that was pressed, there are already 31 keys with a define that can be pressed in the Arcade, feel free to add more if you need it.

- setRes() is a void member function that takes 3 string vectors as parameter, the first one contains paths to textures if your library needs it, the second contains ascii characters if you are using a library like NCurses and the last vector contains RGB colors definitions as strings like “255/255/255”, you will need a call to std::getline() to parse it. This function sets the resources that your library will need to draw the game, for example the Nibbler game contains 5 different types of elements in its map so this function will be called with the first and last vector being empties and the second one containing the ascii characters that the Core expects you to draw.

- An init() void member function that will be called each time a new game is loaded, this function is supposed to do the necessary so we can call the display() member function after this one.

- A display() void member function that takes an int as argument, this int corresponds to the element of the game you needs to draw, it is the same ID as the one you receive in setRes() at the beginning. If the argument is –1, you are expected to go on a new line. This function will be called for each element in the game, so if the game's map is 20x20, this function will be called 400 times plus the newline calls!

- A clearScreen() void member function that will be called once in each game loop, after a call to this function the screen is expected to be cleared so we can display a new map. Mind that the display function doesn't get any position in its parameters to draw, so it's up to you to manage the position of each element you are drawing, if you have to manage the position of the elements you need to draw with x and y coordinates, it would be judiscious to reset o draw with x and y coordinates, it would be judicious to reset x and y to 0 when this function is called, it means that you will have to draw again from the beginning.

- A readKey() void member function that is expected to read user events, this is the function that is expected to set a boolean to true so the isThereEvent() function will let the Core class know that there a new event to analyze.

- Finally, a stop() void member function that is expected to end the display of your library properly, if your library opens a window, this window is expected to be closed in this function.

To finalize your implementation you must write an extern "C" function to avoid g++ mangling so dlsym() will find the symbol of your function.

This function must be named createIGraphic and must return a naked pointer on graphic::IGraphic containing an instance of your concrete class.

How to implement a new game library:

- The first step is that your class must implement the IGame interface.
- The IGame interface is contained in the game namespace so you can call it from outside with game::IGame.
- The interface contains 12 member functions that you must implement in order to get your library working with this project.
- You can find the interface file in ./games/include/IGame.hpp when you are at the root of the project directory.

Here are the 12 functions and their usage:

- getScore() is a const member function returning the current score of the game as an int, it is up to you to manage the score.

- getMap() is a const member function returning the current map of the game as a vector containing vectors of int. For example, if your game's map is 20x20 then the vector wil contains 20 vectors that contains 20 values each. You are free to choose the size of the map but currently they are around 20x20 for each game.

- loadRes() is a void member function that loads all types of resources needed by the graphic library to draw your game, the types of these resources are explained in the next point.

- There are 3 const member functions, getTextrRes(), getAsciiRes() and getColorRes() that returns vectors of strings. The first one is returning a vector of the paths to the textures that the graphic library will need to draw your game. The second one returns the characters that will be used to draw your game's elements by libraries like NCurses. The last one returns RGB codes for the graphic libraries that wants to draw colored squares, these RGB codes are contained in strings like "255/255/255", you will need a call to std::getline() to parse it. Finally, the goal of these functions is to give everything needed to draw your game to the Core class. Only one of these three functions will be called by each graphic library, but you need to implement them all if you want your game to work with any graphic library.

- There are 4 member functions that lets the player manipulate the character of your game, so it allows him to move through your game’s map. By now you should have guessed that these functions are moveUp(), moveRight(), moveDown() and moveLeft(). After a call to one of these functions, the character is expected to be moved across the map so the next call to getMap() will have took the move into account.

- automaticMove() is a void member function that forces the character to move forward even if the user didn’t pressed any key. For example in a game like Nibbler the snake is expected to move automatically even when the player does nothing, but the player can’t go left if the snake is already oriented left nor right, so this function “forces” the movement if necessary thanks to an _automatic boolean variable contained in the AGame abstract class that you should use to implement the IGame interface. This _automatic variable is always set to false except when the automaticMove() function is called, so the move*() functions contains conditions like “if the snake isn’t already on the right or the left, let him go to this direction, if he is already on the right or left and the _automatic variable is set to true then you can force this movement”. This function is implemented in the AGame abstract class using the _automatic variable, so you should just implement the move*() functions taking this boolean into account.

- IsGameOver() is a const member function returning a boolean, this function is supposed to return true when the game is over, so the game loop contained in the Core class can have its condition like “while(!game->isGameOver())” and automatically ends the game when it's over.

To finalize your implementation you must write an extern “C” function to avoid g++ mangling so dlsym() will find the symbol of your function.

This function must be named createIGame and must return a naked pointer on game::IGame containing an instance of your concrete class.