EECS 211 Lab 6

Bejeweled

Winter 2019

Today we will be looking at yet another a C++ program using the GE211 game engine in a reasonably advanced example game. You may have played this game in various forms such as in Candy Crush or other tile swapping games, but the basic concept is to destroy sets of like tiles by swapping two tiles to create a set. In this version of the game a set will be considered 3 or more tiles of the same type. You can only make swaps that will result in the destruction of a set.

This game uses the model–view–controller pattern not-yet-described in class, which allows defining the look, user interaction, and "business logic" of an interactive program as separate components. Provided are the `Model` class which defines the internal game state, the `View` class for rendering game to the screen, and the `Controller` class for reacting to user input and tying it all together. In addition, because the tiles in this game are rather complex themselves, we have tiles.h which sets up `Board_Position` (where something is on a board, basic operators for that position, and functions for finding what is around it), the `Tile_Data` (uses board positions), defines a `Tile_Handler` for processing special types of tiles, the `Tile` struct (a combination of tile data and handlers with the function for swapping with another tile), and finally a `Tile_Handler_Reference` that points to the specific handler to be used in a given tile. All of this allows for the creation of more tile handlers which can each have special destructive powers. You can see that we have provided the normal handler (just delete the set of tiles we created) and a horizontal lazer (deletes all tiles in the row in addition to the set we created by swapping) inside the tile_handlers.cpp.

Getting the starter code

For this lab, the starter code is provided as a ZIP file here: http://users.eecs.northwestern.edu/~jesse/course/eecs211/lab/eecs211-lab06.zip. Extract the archive file into a directory in the location of your choosing. Once you have your new directory containing the starter files, you can open it in CLion.

Be careful, as CLion will only work correctly if you open the *main project directory* with the CMakeLists.txt in it. If you open any other directory, CLion may create a CMakeLists.txt for you, but it won't work properly.

General idea

The version of Bejeweled that you have been given is not following all of the rules we discussed earlier but is otherwise a fully functioning

version of Bejeweled. This game loads a board (defaults to 10 by 8) with several (defaults to 6) groups - which will behave as tile colors for grouping same colors - and as many types as you have tile handlers (starts at 2). From here, the controller decides when to update a frame and in each update `model_.run_step()` is called which is the brains behind finding what needs to change, detecting the set of tiles to be destroyed (including any caused by destroying a special type), and removes them as needed. Looking through this function and the other functions it uses in the `Model` class should help you understand how the game is utilizing tiles. The `Controller` also utilizes some of the `Model` functions in `Controller::on_mouse_up` which (when the view isn't going through animations) on first click of a valid tile will select that tile and on second click of another tile will attempt to swap them. Upon swapping and creating a set to be destroyed, that set of tiles will be removed and the tiles above them will be shifted down, and new random tiles will also be shifted down to fill the gaps created at the top. All of these changes are animated by the `View` class which will make the program unresponsive to input while it displays the changes slow enough for you to actually see what happens.

More Valid Swaps

We mentioned above that this version of Bejeweled does not follow all of the rules we described at the beginning. Specifically, this version of Bejeweled allows for any two tiles that are next to each other to be swapped instead of limiting the swaps to only those that will create valid sets for destruction.

In controller.cpp the function discussed above - `Controller::on_mouse_up` - uses `model_.is_valid(bp)` to check that the selection is inside the board. However, we want something more complicated than just checking that the selection is on the board, we also want to make sure that the selection has a valid swap. Your job is to modify the code in the `Model` class to only tell the `Controller` a selection is valid if it will also be valid for a swap. To accomplish this you will have tom

1. Modify `Model::is_valid` to only return true when the position is on the board and if the there is a possible valid swap. Remember, there is a function `Model::is_valid_swap` that may be useful. As well, keep in mind that there are four possible ways that a tile could swap and the one you select only needs to be able to swap with one of them (although more won't hurt).
2. Modify `Model::is_valid_swap` to only return true when both tiles are on the board, next to each other, and will result in a set of tiles

to be destroyed. Consult the aforementioned `model_.run_step()` to see how valid groups for destruction are created - even bigger hint, look at `Model::get_group_` to see how sets of a group of tiles are found.

Cooler Tile Handlers

Currently, the special tile handlers are relatively boring. Your job is to add a few more handlers to make the game more interesting. To add a tile handler you will have to:

1. Create a new class in tile_handlers.h following the style of Normal and Horizontal Lazer with your new name instead.
2. Define the `process_removal` function for your new class (follow the style of Normal and Horizontal Lazer here as well but make the inside of the function only choose the tiles to delete that you want) inside tile_handlers.cpp.

Go ahead and add vertical lazer, destroy all tiles in this group, and destroy all tiles on the diagonals (X) handlers. You will also need to modify game.cpp to know about these new types:

1. Update `types_count` to the correct number of types you've added.
2. Inside the `main()` function the tile handlers are instantiated and used in the call to `Controller(...)` by adding a `Tile_Handler_Reference` to the set of handlers the game knows about. Instantiate your own handlers and add the references to the game.