

# CS 132, Winter 2022

## Programming Project #2: Tiles (20 points)

Due Thursday, January 20, 2022, 11:59 PM

thanks to Mike Clancy and Marty Stepp for parts of this project

This project will give you insights into how operating systems manage multiple programs' windows. Its implementation focuses on using vectors, graphics and basic object-oriented programming. Turn in files named `Tile.cpp`, `Tile.h`, `TileManager.cpp` and `TileManager.h`. You may also submit `lib132.cpp` and `lib132.h` if you use them.

You will need the `tileTester.zip` and `tiles.zip` starter code from the Project section of the course web site. You will need to unzip these starter project before opening them. Only turn in the files listed above. Do not alter the other files in the project.

You should not modify the provided files. The code you submit must work properly with their unmodified versions.

### Program Description:

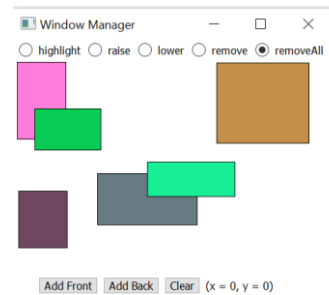
In this assignment you will write the logic for a graphical program that allows the user to click on rectangular tiles. You will not write any code to create a window, user input controls or deal with mouse clicks as this code is provided. The only graphical code you will write is code to draw rectangles.

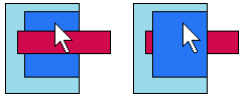
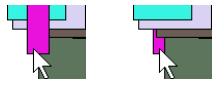


The main client program you should run is the provided `TileMain`. When it runs, it will create a graphical window on the screen; this is an object of type `GWindow`. The panel displays a list of tiles. (Initially the panel shows 20 tiles, but you can add more by clicking the "Add top" or "Add bottom" button.)

Each tile is represented as an object of another class you will write: `Tile`. Each tile's position, size, and color are randomly generated by `TileMain`. You will write the `Tile` class to store this information and perform a couple useful operations. You will write the `TileManager` to store and maintain a list of pointers to `Tiles`.

The order of `Tile` pointers stored in the `TileManager`'s list determines tiles' drawing order. For example, consider the tall pink tile that overlaps the wide green tile in the upper left corner of the screenshot at right. The two tiles occupy some of the same (x, y) pixels in the window but the tall pink one was drawn first because it occurred before the wide green one in the vector. When the wide green tile was drawn later, it covered part of the tall pink tile. The vector's ordering is called the 3-dimensional ordering or *z-ordering*.

The graphical user interface ("GUI") displays the tiles and allows the user to manipulate them. Depending on the radio button selected, different actions occur:



- If the user clicks the mouse on a tile while the **highlight** radio button is selected, that tile is surrounded by a yellow border. This border should disappear the next time the user performs an action other than highlight or adding a tile.
- If the user clicks the mouse on a tile while the **raise** radio button is selected, that tile is moved to the very *top* of the z-ordering (the end of the tile list). 
- If the user clicks the mouse on a tile while the **lower** radio button is selected, that tile is moved to the very *bottom* of the z-ordering (the start of the tile list). 
- If the user clicks the mouse on a tile while the **remove** radio button is selected, that tile is removed from the tile list and disappears from the screen. 
- If the user clicks the mouse on a tile while the **removeAll** radio button is selected, *all tiles that occupy that pixel* are removed from the tile list and disappear from the screen. 
- If the user clicks the **Add bottom** button, a new randomly positioned and colored tile is created and added to the screen at the bottom of the z-ordering.
- If the user clicks the **Add top** button, a new randomly positioned and colored tile is created and added to the screen at the top of the z-ordering.

If the user clicks a pixel that is occupied by more than one tile, the top-most of these tiles is used. (Except if the user selected **removeAll**, in which case it deletes all tiles touching that pixel, not just the top one.)

Note that **your code does not need to directly detect mouse clicks or button presses**. Code in `TileMain.cpp` detects any user input. When user input is detected, member functions that you define in `TileManager.cpp` are called, as described on the next page.

## Implementation Details:

You will be implementing two classes in this project.

### Tile.h / Tile.cpp

Your `Tile` class should store information about a **single** tile. It shouldn't store information about any other tiles and may **not** store an array or `vector`. Don't worry if your code is short and doesn't appear to do much – that is expected here.

The following sections describe in detail each member function you must implement in your `Tile` class. You may assume all parameter values are valid.

```
Tile(int x, int y, int width, int height, string color)
```

This constructor is called every time a new tile object is created. Make sure to store any information you will need later on to be able to write the other functions.

```
int getX()
```

Returns the x coordinate of the upper left corner of the tile

```
int getY()
```

Returns the y coordinate of the upper left corner of the tile

```
int getWidth()
```

Returns the width of the tile

```
int getHeight()
```

Returns the height of the tile

```
string getColor()
```

Returns the color of the tile

```
bool contains(int x, int y)
```

Returns `true` if the tile contains the x, y location and `false` otherwise.

You can figure this out by comparing the x/y passed in to the x/y area covered by the tile. For example, if a tile has a top-left corner of (x=20, y=10), a width of 50, and a height of 15, it touches all of the pixels from (20, 10) through (69, 24) inclusive. Such a tile contains the point (32, 17) because 32 is between 20 and 69 and 17 is between 10 and 24.

```
void draw(GWindow& window)
```

Draws the tile on the passed in window at its x, y position, at its width and height and in its color

You can change the color of all future drawing on the window by calling the `GWindow` member function `setFillColor(color)` and passing it a string representing the color. Draw a rectangle on a window by calling the `GWindow` member function `fillRect(x, y, width, height)`.

For example, if you wanted to draw a red rectangle at an x of 10, y of 30 that was 100 wide and 150 tall on a `GWindow` named `window` you could write:

```
window.setFillColor("red");  
window.fillRect(10, 30, 100, 150);
```

```
void print()
```

Prints out a text representation of the tile in the following format:

```
x = xValue, y = yValue, width = widthValue, height = heightValue, color = colorValue
```

For example, a red rectangle at an x of 10, y of 30 that was 100 wide and 150 tall would print as:

```
x = 10, y = 30, width = 100, height = 150, color = red
```

## TileManager.h / TileManager.cpp

Your `TileManager` class should store a list of `Tiles` as a **member variable of type `vector<Tile>`**. The various member functions listed below will cause changes to the contents of that list.

The following sections describe in detail each member functions you must implement in your `TileManager` class. For any member function that accepts parameters, you may assume that the values passed in are valid.

### `TileManager()`

This constructor is called every time a new tile manager object is created. Initially your manager is not storing any tiles.

### `void addBottom(Tile& rect)`

Called when the program starts and when the "Add bottom" button is pressed, adds the given tile to your tile manager's list of tiles so that it appears on the bottom when drawn.

### `void addTop(Tile& rect)`

Called when the "Add top" button is pressed, adds the given tile pointer to your tile manager's list of tiles so that it appears on the top when drawn..

### `void drawAll(GWindow& window)`

This member function should cause all of the tiles in the tile manager to draw themselves on the passed in window. Make sure to take advantage of other functions you may already have written to do this drawing. Draw the tiles from bottom (start) to top (end) of your manager's list.

### `void clear()`

This member function should remove all tiles in the tile manager.

The next four member functions are called by the graphical user interface ("GUI") in response to mouse clicks, passing you the x/y coordinates where the user clicked. If the coordinates passed do not touch any tiles, no action or error should occur. After any click (except when highlight is selected), the GUI will clear the screen for you and call `drawAll` to re-draw all of the tiles in your list.

### `void highlight(int x, int y, GWindow& window)`

Called when the user clicks and the highlight radio button is selected. `TileMain` passes the x/y coordinates the user clicked on and the window. If these coordinates touch any tiles, you should draw width 10 yellow border around the topmost of these tiles.

Set the border color to yellow with the `GWindow` member function `setColor(color)`. Change the line thickness to 10 with the `GWindow` member function `setLineWidth(width)`. Draw the border with the `GWindow` member function `drawRect(x, y, width, height)`.

For example, if you wanted to draw an 8 wide blue border around a rectangle at an x of 10, y of 30 that was 100 wide and 150 tall on a `GWindow*` named `window` you could write:

```
Window.setColor("blue");
Window.setLineWidth(8);
Window.drawRect(6, 26, 108, 158);
```

Notice that the coordinates are not quite the same as the rectangle the border is surrounding. They are  $\frac{1}{2}$  the border thickness higher and to the left and the border thickness more wide and tall. If you draw the border at exactly the same place the rectangle will overlap with part of it.

### `void raise(int x, int y)`

Called when the user clicks and the raise radio button is selected. `TileMain` passes in the x/y coordinates the user clicked on. If these coordinates touch any tiles, you should move the topmost of these tiles to the very top (end) of the list.

### `void lower(int x, int y)`

Called when the user clicks and the lower radio button is selected. If these coordinates touch any tiles, you should move the topmost of these tiles to the very bottom (beginning) of the list.

### `void remove(int x, int y)`

Called when the user clicks and the delete radio button is selected. If these coordinates touch any tiles, you should delete the topmost of these tiles from the list.

```
void removeAll(int x, int y)
```

Called when the user clicks and the deleteAll option is selected. If these coordinates touch any tiles, you should delete *all* such tiles from the list.

## Development Strategy and Hints:

Developing your code in stages and knowing how to test your solutions will be critical to your success on 132 assignments. One of the most important techniques for professional developers is to write code in stages ("**iterative enhancement**" or "**stepwise refinement**") rather than trying to do it all at once. This includes testing for correctness at each stage before moving to the next one. The next few paragraphs contain a detailed development plan. Study it carefully and think about why we suggest the plan we do.

Start by writing the `Tile` class. Write the constructor first, then the getter member functions. Then write `contains` and `draw`. Use the provided `tileTester` project to make sure your `Tile` works correctly before continuing.

Next, start writing the `TileManager` class by writing empty "**stub**" versions of all the required member functions so the code will compile.

We suggest that you write your constructor, `addBottom` and `addTop` first, then `drawAll`. You can then run the program to make sure that you can see the tiles appear on the screen. Add click-related member functions one at a time and test each one individually to be sure it works before moving on to the next.

If you have bugs or segfaults in your code, there are several things you can try. You can print out the state of your list and of each `Tile` object with temporary **cout statements**. (Though this is a graphical program, `cout` output does still appear on the console.) You should remove any such `cout` statements before you turn in the assignment. You can also use a **debugger** as found in VSCode or QtCreator to pause the code at a breakpoint in the middle of an operation. Stepping through line-by-line can help you to see what has gone wrong.

A **video** of the program running is available on the course web site. You can use it to verify your program's behavior.

## Style Guidelines and Grading:

A major focus of our style grading is **redundancy**. As much as possible, avoid redundancy and repeated logic in your code. Make sure that you call `Tile`'s functions wherever they are useful in `TileManager`. Don't repeat the same logic.

Another powerful way to avoid redundancy is to create "helper" member function(s) to capture repeated code. It is legal to have additional member functions in your `TileManager` class beyond those specified here but they must be `private`. For example, you may find that multiple member functions in your class do similar things. If so, you should create helper member function(s) to capture the common code.

Your tile manager should maintain its list of tiles internally in a member variable of type `vector` that stores `Tiles` as stated previously. You should not use any other data structures. Your code should use type parameters appropriately when creating any collection objects. You must declare such collections with a suitable element type within `<>` brackets. You should use the vector and its member functions appropriately and take advantage of its ability to "shift" elements as they are added or removed. Your vector should not store any invalid elements as a result of any mouse click activity.

Place all function headers and class declarations in your header files, place function implementations in your `cpp` files. Only include header files in other files, do not include `cpp` files.

Properly **encapsulate** your objects by making any data members in your class `private`. Avoid unnecessary member variables; use member variables to store important data of your objects but not to store temporary values only used in one place. Member variables should always be initialized inside a constructor or member function, never at declaration.

You should follow good general C++ style guidelines such as: appropriately using control structures like loops and `if/else` statements; avoiding redundancy using techniques such as member functions, loops, and factoring common code out of `if/else` statements; properly using indentation, good variable names, and types; and not having any lines of code longer than 100 characters in length. (If you have any such lines, split them into two or more lines using a line break.)

You should **comment** your code with a heading at the top of your class with your name, section, and a description of the overall program. Also place a comment heading on top of each member function, and a comment on any complex sections of your code. Comment headings should use descriptive complete sentences and should be written *in your own words*, explaining each member function's behavior, parameters, return values, and assumptions made by your code, as appropriate.

Unless otherwise specified, your solution should use only material taught in class and in the book chapters covered so far.