Homework 7 – Level Editor

**Due Monday, December 9th by 11:59PM**

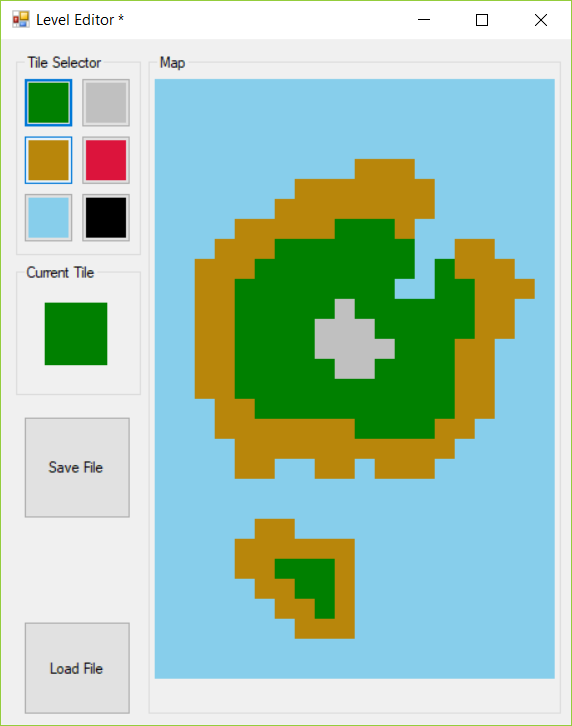
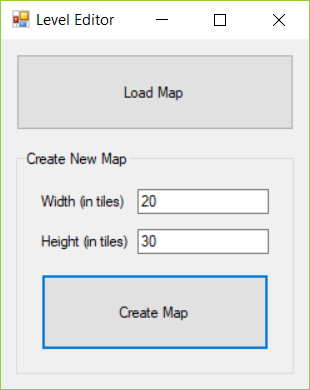
This program will be a level editor. This editor could be used for a 2D side-scrolling platformer or a top-down shoot-‘em-up. You will use a Windows Form Application to create this visual level editor.

For this assignment, you may use either text files or binary files when saving/loading levels.

# Task Overview

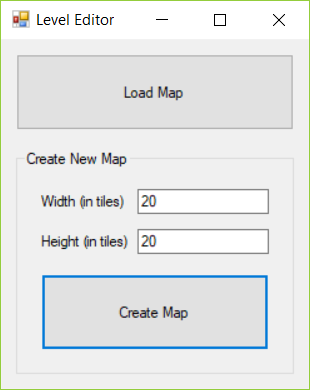
This is a brief overview of the tasks you must complete for this assignment. Specifics are given in the corresponding sections later in this document. ***Read the entire document before starting.***

* Create 2 forms – one for the main page, and another for the level edits
* Allow the user to select from several tile colors
* Display an array of colored map tiles that the user can interact with
* Use the built-in OpenFileDialog and SaveFileDialog windows
* Read data from an external file
* Write data to an external file
* Check for edits before closing

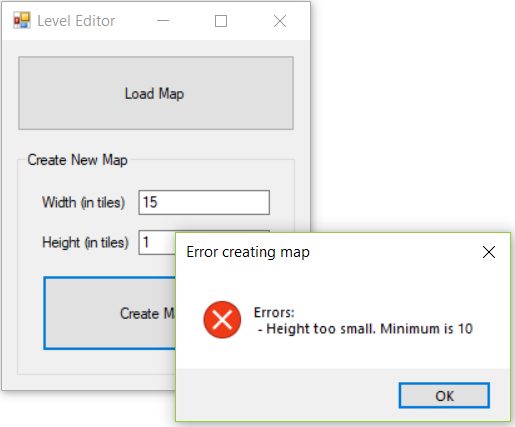


# Main Form

Your main form will be the form that first appears when a user runs your program. There is an option to load a previously created map, and another option to create a brand new map.



#### Width and Height Ranges

This form has textboxes for the width and height of the map. When creating a new map, the minimum tile count is 10, and the maximum tile count is 30. Use a MessageBox to remind the user of those limits if either value is out of range.

#### 

#### Creating a Map

Creating a new map will bring the user to the second form in this assignment: the editor form. Pass any necessary arguments (the file to load or map sizes) to the editor form when you instantiate it.

#### Loading a Map

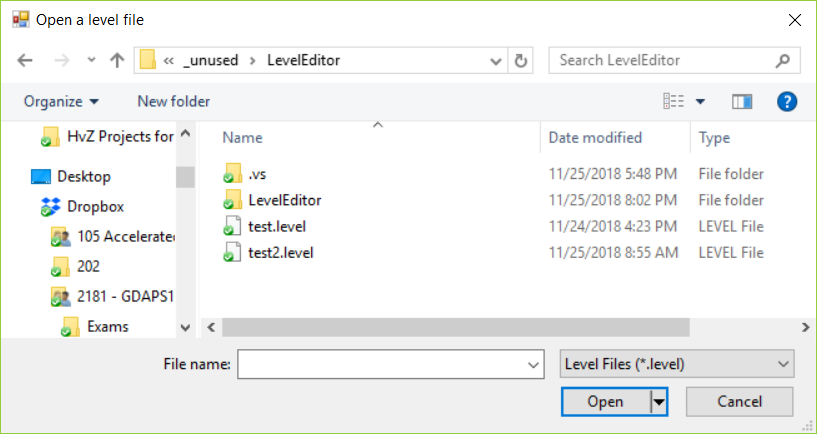
An OpenFileDialog object can allow a user to choose a file and path easily. Instantiate a new OpenFileDialog when the user wants to choose a file to open. Before showing it to the user, you can change several of its properties to customize it.

This file dialog window doesn’t actually open a file for you; it simply allows the user to choose a file and path, and allows you (the programmer) to get that information easily. The title bar of the file dialog window should say “Open a level file.” A filter can be set to only open particular types of files; use the Filter property with a value of “Level Files|\*.level”. This limits the types of files that can be opened to any file ending in the .level extension. The chosen filename can be retrieved with the FileName property.

Like any other form we’ve used, the file dialog window must be opened with ShowDialog(), which returns a DialogResult value. After the OpenFileDialog object is opened with ShowDialog(), pressing any of the buttons (X, Open, or Cancel) will return various DialogResult values. If a user chooses the Open button, it will return DialogResult.OK.

If the user chooses to load an existing file, open the editor form and then load that file in the editor form. File reading or writing should NOT be done in this main form.

If they hit cancel, do not load any files. If this is the case, the user will still be on the main form.



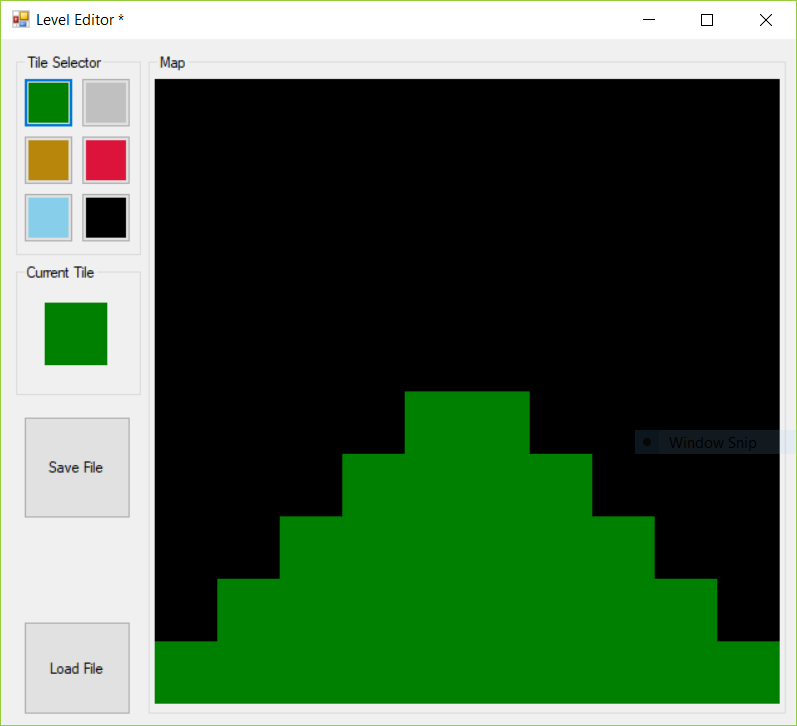
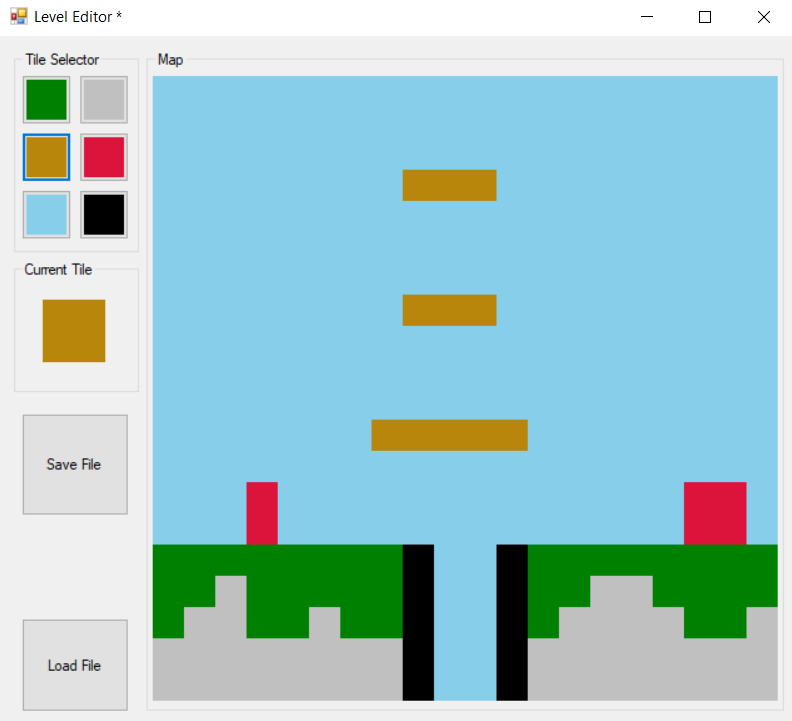
# Editor Form

This second form allows a user to change the color of tiles in their map. The exact tile colors the user may choose from are chosen by you and should correspond to the type of tile/platform in the game. For instance, gray could be rock, green could be grass, brown could be sand or wood, blue would be water or sky, etc. A preview of the currently chosen color appears below the color choices. There are buttons to save a file or load a pre-generated file.

Each of the 6 color options are Buttons, while the “grid” for the level is a 2D array made of PictureBoxes. (Buttons could be used for the actual tiles, but buttons have a border which would get in the way the smaller the tiles get. Please use PictureBoxes.)  
  
The level editor should stay roughly the same *height* regardless of the number of tiles. Its width, however, should change depending upon the overall dimensions of the map. Each tile should always be a perfect square (no rectangular tiles), but the exact size of each tile is based on the total number indicated from the main form.

20 x 20 tiles

10 x 10 tiles



#### Selecting Tiles

After a user chooses one of the 6 pre-determined color buttons on the upper left, they can then click on a tile (an individual PictureBox) in the map and that tile will change color to the selected color. Choosing a different color and clicking on a tile will change its color again.

Being able to click and drag across multiple picture boxes to “paint” them is not a requirement of this assignment. See the last page of this document for information about implementing such a feature if you so desire.

#### Saving File

The user may choose to save the file at any time by clicking the “Save File” button. When this occurs, you should create a new SaveFileDialog object to allow the user to choose a proper save location. Similarly to the OpenFileDialog objects, the SaveFileDialog has a properties for Filter, Title and FileName. The filter should be the same as for opening a file, and the title should say “Save a level file.” Pressing the “Save” button returns DialogResult.OK. Pressing any other button (X or Cancel) should not result in action.

Once you have the user’s selected file name and path, write the information about each PictureBox to a file. When saving color data to a file, you may find the .ToArgb() method helpful. When called on a control’s BackColor property, it returns a 32-bit integer of that control’s background color. Conversely, a background color can be set with the Color.FromArgb() method, where a 32-bit integer can be passed in.

#### Loading File

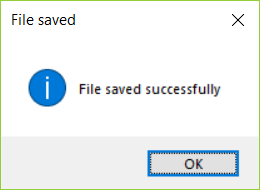
The user may choose to load a file anytime with the “Load File” button. When pressed, use an OpenFileDialog object to gather the filename from the user, then read that file.

When loading data, the sizes of the PictureBox objects may need to change depending on the file. We recommend that the first 2 pieces of information in your file are the width and height of the level. Storing this information will assist you in calculating the right size for each tile.

Use data from the file to populate the current colors of the PictureBoxes in the 2D array.

#### Successful Saving & Loading

Upon a successful save or load, a MessageBox should appear, letting the user know the file was saved or loaded successfully. The title bar of the editor form should change to show the current saved or loaded file.



Level Editor title bar, displaying currently loaded file name

#### Unsaved Changes

Anytime there are unsaved changes, the title bar of the editor form should show an asterisk next to the title “Level Editor.” This occurs regardless whether a file was loaded and edits made afterward, or whether a user started with a new map and made changes.

When changes are saved, the asterisk will not be shown.



Title Bar - File "test.level" loaded, and unsaved changes detected

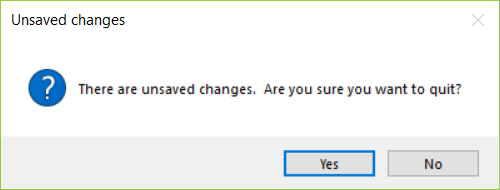
Title Bar - No file loaded, no unsaved changes

Title Bar - No file loaded, unsaved changes detected

An easy way to track unsaved changes is with a simple Boolean field. When the program starts up, there are no unsaved changes. Any time the user clicks any PictureBox tile, there has been a change. After saving, there are no pending changes.

#### Handling Unsaved Changes

Every form has a FormClosing event. Should a user click the “X” button in the top-right corner of the editor form, check for any new changes to the level. If a user has made changes since they saved last, or if they haven’t saved at all, prompt them with a MessageBox.



If the user chooses no, do not allow the page to close. This can be done with the FormClosingEventArgs e parameter. By setting e.Cancel to true, it will cancel the current action (attempting to close the window) and will keep the editor form open. If they choose yes, the closing action will continue as usual with nothing to do on your part.

#### Not Required: Clicking & Dragging Across PictureBoxes

Clicking and dragging across multiple PictureBoxes is a more complex feature. The first issue is that upon any control (PictureBox, Button, etc.) receiving a MouseDown event, that control “captures” the mouse cursor. This means only THAT control receives further mouse-related events until the mouse button is released.

The first step in getting click/drag to work is to un-capture the mouse. To do this, you’ll need to set the PictureBox’s Capture property to false inside its MouseDown event. This will allow other controls to receive MouseEnter events while the mouse button is held down.

Secondly, you only want to change a PictureBox’s color if it receives the MouseDown event itself, or if the left button is pressed when it receives a MouseEnter event. To facilitate this, you can check the form’s MouseButtons property, which is an enum value representing which button is pressed.