**Test Cases**

The best way to test a model is to know if the parameters that it contains are useful for the program.

**Settings Model**

* In this case, ID and Name are useful. We can use this to create a list of combo box values to populate a combo box or multiple.
* One test case would be:
* Create individual values for a combo box using a SettingsModel.
* Ex:

List<SettingsModel> comboBoxValues = new List<SettingsModel>();

comboBoxValues.Add(new SettingsModel() { ID = 1, Name = "3" });

* This is a good test case because using this list we can determine which values we want inside our Combo box and in which order by using the ID.
* Lastly, one may add multiple Combo Box values using the SettingsModel list to create a Combo box, which would be the final step to testing.

**Tile Model**

* This model is useful for the Color Buster game because it contains all the properties which a tile should.
* The best way to test this model would be to create a tile and add it into the view.
* Example:

TileModel tile1 = new TileModel()

{

Name = "tile1",

xLocation = 0,

yLocation = 0,

imageIndex = 0,

width = 0,

height = 0,

control = new PictureBox()

};

* Using the **Name**, we can identify which tile was clicked.
* A test case would be clicking a tile and displaying its name through a message box.
* Using the **Location** of the tile we can map it into the board and locate its adjacent tiles.
* A test case would be adding a new tile right next to the original tile.
* Using the **Image Index** one can know which is the current image (color) that a tile has.
* A test case would be to change the color after a click event by changing the image index of the tile.
* The **Width** and the **Height** can be used to resize the tiles based on the number of rows and columns in a container of tiles (board).
* A test case would be changing the size of the tiles to fit the board.
* The **Control** is used for mapping the tile into the view. This is because this game was built using windows forms which has many components. In this case, a tile is a picture box.
* A test case would be to create a tile and add it to the view: board.Add(tile1);

**Board Model**

* The board model contains static values that can be both accessed and modified through getters and setters. These properties inside a Board Model are crucial to the game because they help keep progress of the game.
* There are many ways to test a Board Model because it contains many properties, therefore, there need to be more focused test cases.
* The **number of Rows and Columns** are dynamic variables with an initial value of 6 which determine the number of tiles in a board.
* Test case 1: would be to debug the game and through a breakpoint see if the starting values are indeed 6.
* Test case 2: would be to change the values through the combo box created by the SettingsModel.
* The set integer used to set the **Number of Tiles** that need to be matched in a game to gain points can be grouped together with the **List of Tiles** that have matching colors and are adjacent. This is because the numbers can be compared to check if the game settings (rules) are being met.
* Test case 1: Set the number of required tiles using the SettingsModel and check if the value was assigned.
* Test case 2: Debug the program, put a break point and check if the initial value is indeed 3.
* Test case 3: Fill the list of matched tiles with Tile Models and use the property of lists “count” to compare that number to set amount in the settings.
* The two lists of images that contain the **Colors of Tiles** and then the other which contains the images of the **Popped Tiles** can be grouped together because they are related.
* Test case 1: The simplest and most essential test case would be to display the images in the view.
* Test case 2: Adding each image to a list of images so that the index of each list can be assigned to a Tile Model.
* Test case 3: Setting a time out for changing the image from a colored tile to a popped tile, which will test the intended functionality.
* The **Tile list** contains all the available tiles in the board to keep track of their location and to be able to compare them to see if they match or not.
* Test case 1: Add multiple tile models to the list and compare their locations on the board to see that all the coordinates are properly set.
* The **Match Tiles List** is important because the match tiles list contains the tiles that match to a clicked tile so that they can be popped and replaced by new tiles. This list gets reset every time a tile is clicked.
* Test case 1: Click a tile and make sure that the list contains only tiles that have the same image index, which means that they contain the same color. If they do not, then the test failed.
* The is **Move Available Tile List** contains all the matched tiles but starting from the first tile every time a tile is clicked until the list contains several matched tiles that matches the required number in the setting. If the LINQ query returns NULL, no move is available.
* Test case 1: Make sure that the recursive method starts at the first tile every time.
* Test case 2: Make sure that the recursive method keeps going until a tile is found.
* Test case 3: Make sure that when no tile is found, null is returned.