# Final Project: Jewel Matching

## Due Date

Thursday, December 13th, at Noon.

You may watch the video for this project on the course website.

## Objectives

Develop a complete application using XAML to allow a user to play the Jewel Matching game.

## Instructions

In this assignment, you will be building the game Jewel Matching. To determine your grade, please complete this project in the following phases. Each Jewel in this game is called a "Jelly".

### Phase 0. Run the project.

Just run the project. Look at all of the existing classes. If there was a private data member that I used in creating this project, I left it in. Make sure that you review all of the files before we begin.

PLEASE, PLEASE, PLEASE: Rename the project folder with your name.

Here are the files that I used to complete this project:

* GameBoard.cs: Represents the board itself and how it behaves, as well as the score.
* GameModel.cs: It has a representation of the board and will store information about mouse clicks. This file is not included.
* Tile.cs: Represents each graphic that should appear in the game.
* MainWindow.xaml.cs: Represents our game's controller. If it connects the player to the game, it should go here. I call this the "Controller class".
* GameView.cs: Represents how the game should be represented.
* ImageSelector.cs: This is our tool for pulling images from the resources file. This is unchanged from our video game that we wrote in class except to change the image sizes from 80 pixels to 50 pixels.
* IViewable.cs: Our interface for anything which should be viewable. It's the same as in our last game.

### Phase 1. Create a GameModel. 10 points.

The GameModel should maintain aspects of the game that are hard to put into other features of the game. In this case, that will be things like mouse clicks. I'm not giving you a GameModel class. I want you to make one.

1. Create a class called GameModel.
2. The public constructor should take an int width and int height parameters and create a new GameBoard object, passing the width and height to that class' constructor. Make this GameBoard object a property with a private setter. Give the constructor some documentation.
3. Have your class implement IViewable and have your GetTiles method return the GameBoard object's GetTiles content. This method only needs one line of code. Give the method some documentation.

### Phase 2. View the board. 40 points.

In this phase, you will get the board to be displayed. In the Controller class, build an object for the following:

* A GameModel with a width of 12 and a height of 10.
* A GameView which connects the model to the canvas named world.
* Call your GameView object's Update method to view the content.

You should see rows of yellow pieces on the screen. I have that there to make sure that you see something. Go into the constructor of the GameBoard object and change the YELLOW to EMPTY. Rerun your program to see nothing appear (because all of the tiles are empty). You can experiment with other available colors, but I want these by default to be empty.

### Phase 3. Randomize those tiles. 10 points.

Figure out a way to randomize the tiles. I recommend going into the GameBoard class and adding a public method called Randomize which will roll a number from 0 to 5 and select a Jelly based on that number for each EMPTY square on the board. That last bit is important. I only want Randomize to overwrite squares on the board which are EMPTY. If a cell on the board already has a color, skip over it and go to the next cell. The Board property in this file is a 2D array of JellyCode enums.

Make sure that your Randomize method has some documentation.

There are methods already in GameBoard in order to help you with this. Where would be the best place to call your new Randomize method when you are done with it? Place the call in the appropriate spot.

You should see a rainbow of Jelly tiles on the screen.

### Phase 4a. Create a method in the Controller class to fetch mouse clicks. 10 points.

For this, you'll need to modify the XML in the MainWindow.xaml file (not the MainWindow.xaml.cs file yet). Add a new event handler to the Canvas line called MouseDown and create a method to go with it. The default name that is suggested is world\_MouseDown. I think that's an okay name. It should create a method in your MainWindow.xaml.cs file that looks like this:

private void world\_MouseDown(object sender, MouseButtonEventArgs e)  
{  
  
}

The MouseButtonEventArgs object will store information about your mouse clicks.

* e.GetPosition(this).X will give you the X coordinate of your mouse click.
* e.GetPosition(this).Y will give you the Y coordinate of your mouse click.

Pass this click information to the GameModel object in a method called RegisterClick (which you haven't written yet).

### Phase 4b. Create the RegisterClick method. 10 points.

Create a method called RegisterClick which will take an (X,Y) coordinate pair and do the following:

* If this is a "first click" (the player is clicking on the first Jelly to move), then the method remembers this information and does nothing. There has been 1 click.
* If this is a "second click" (the player is clicking on the second Jelly to move), then the method should figure out which two Jelly objects the user was trying to click on using the first and second click information. Then the method should call the Swap method in the GameBoard class with the coordinates of these two Jellies. The method should then forget that there were any clicks. At the end, there has been 0 clicks.

You'll notice that when you click around in your game, nothing happens. You need to update the GameView object. Where would be the best place to call Update on your GameView object? After you properly update the view, you should see Jelly objects swap places if they are side-by-side.

### Phase 5. Looking for runs. 10 points.

This will be challenging. You need to search the board for "runs". A "run" is a sequence of Jelly codes on the board that are 3, 4, or 5 elements in length that appear either vertical or horizontal on the board. Honestly, I wrote about 100 lines of code to look for all of these runs. If you want a real challenge, look for runs of 3, 4, or 5. If you are pinched for time, look for runs of 3 (this will hurt your score for later points).

Create a method called Update in the GameBoard class that will look for runs. The Update method should identify a run and then replace all elements in that run with a code of EMPTY.

While you are updating the game, compute a score for each run found.

* Runs of 3 are worth 50 points.
* Runs of 4 are worth 100 points.
* Runs of 5 are worth 200 points.

You'll need to create a int property in the GameBoard called Score. Initially, set it to 0. Each time a run is discovered, update the Score with the appropriate score increase.

You will need to call this Update method after each mouse click, but before the GameView Update method is called. When you play the game, the runs should disappear and nothing will reappear in their place.

## Phase 6. Show the score. 5 points.

This will require updating the GameView class with our score. Add a TextBlock directly on the canvas named world within the GameView Update method or add a Label interface element to the MainWindow.cs file. You'll need to modify the GameView class to update the your label with the game score. This may require that you add an additional parameter to the GameView constructor of the Label interface element.

Each time you find a run in the game, the game should update and the score should increase.

## Phase 7. Implement Gravity.

Traditionally these games work by new items falling from the top of the screen. If there is a cell containing a jewel "above" (visually speaking) an empty spot, the jewel "falls" to the empty spot. If there are multiple empty spots, the jewel falls until there is a jewel beneath it or the jewel hits the bottom of the game board.

There are two ways in which you can handle this issue.

1. Implement Gravity. 5 points.
2. Just call "Randomize" to fill in the gaps. You've already done this. 0 points.

Here's how I implemented gravity.

1. I scanned the board for each column and for each row (except for the last row).
   * IF a cell contained a Jelly and the cell beneath it on the same column was empty.
     + I swapped the cell with the Jelly with the empty cell.
     + I also set a boolean flag to true indicating that I made at least one swap.
2. If I made at least one swap, I reset the boolean flag to false and went back to the first step.
3. Eventually, all of the empty cells will make their way to the top of the map. Then I called Randomize to fill in all of those empty cells.

That should be 100 points of your grade. If you wrote a new public method, whether I mentioned to add documentation or not, please document the new method. Upload your project as a zip file to D2L.

Have fun with your game. It's been a wonderful time teaching all of you this semester.