

## Rodbourn's Blog

<< Undo Within Selected | Home | Generating a Symbolic Vector in Matlab >>

### Creating a Bejeweled Blitz Bot in C#

One of the more addictive games out there is Bejeweled Blitz on Facebook. It's fun to play, but as a programmer one thing nags at you as you play. The thought 'I could program something that would do much better.' I program mostly scientific codes and business applications. A bot application is quite different, and hence a great project. I'll cover the process of making a bot here; however, I will not give a direct download. If you want to run the bot you will have to compile your own application. Cheating is not a goal here.

First, credit where it is due, [Mike Vallotton posted on the same topic](#). Between this and his article assembling a bot should be straight forward. My additions are in locating the game window, using state to identify game pieces, and using a damper on game moves. He reports a score of 212K; I have reached 1.06M without much attention to the pattern recognition.

The approach is to create a bot that interfaces with the game in the same way a user would, through a mouse. Doing so involves three major steps, reading the game state from the screen, solving for a move, and finally making the move. Reading the game state from screenshots may be the trickiest of the three.

### Locating the game window

Locating where the game window is on the screen was one of the more interesting problems to solve. We could use some libraries to get the window location of a browser which contains the Bejeweled game, but we would not know where within the window the game resides. Further, when needing to know where the game is to the pixel, the game location on the page is far too volatile with changing ads, content, etc.

To find the window I use a screenshot of the whole desktop and then find the game within this image. Doing a per-pixel match is absurdly expensive, so I use an idea I borrowed from [multigrid solvers](#), yet simpler. To find the window I first capture a screenshot and crop it down to the game I want to find, and save it to disk; in this case the title screen of the game. Then I take a screenshot of the desktop captured during the run and attempt to find the title screen within.

To find the title screen I first convert both images to grayscale by simply taking the red channel. With grayscale images we then have a two dimensional array to work with, speeding things up a bit. To speed things up further, both images are scaled down to 0.10 their original size (this is the idea of successive scales is borrowed from the multigrid solver). Then the title screen is subtracted from the array at various locations. After the subtraction the two dimensional array is summed to get a total sum value. The location where the sum has the least value is taken to be the location where the window is located.

For the first levels of locating we do not need to be exact; we are just looking for regions where the title screen is so that the next level will have a reduced area to search. For the first search I stepped across the screenshot by 2 pixels (20 pixels on the original image). This then locates the game title screen to +/- (2\*10) pixels on the screen. Great, now we can resize to 0.25 and search a 10 by 10 region. This is repeated at 0.50 before searching the final full scale screenshot. I prefer to work in Matlab for such operations. This does require compiling the Matlab code and referencing it from the .NET application.

The C# code to get a screenshot and call the compiled Matlab function:

```
private void FindGameScreen()
{

    Rectangle r = new Rectangle(0, 0,
        Screen.PrimaryScreen.Bounds.Width,
        Screen.PrimaryScreen.Bounds.Height);

    Bitmap bmp = new Bitmap(r.Width, r.Height);
```

```

using (Graphics g = Graphics.FromImage bmp))
{
    g.CopyFromScreen(
        new Point(0, 0),
        new Point(0, 0),
        r.Size);
}

int[,] bmpmat = new int[r.Height, r.Width];
for (int i = 0; i < r.Width; i++)
    for (int j = 0; j < r.Height; j++)
    {
        Color pixelColor = bmp.GetPixel(i, j);
        bmpmat[j, i] = pixelColor.R;
    }

var bmpinp = (MWNumericArray)bmpmat;
var fileinp = (MWCharArray)@"gametitle.bmp";
MWArray test = (MWNumericArray)findWindow.findwindow(bmpinp, fileinp);

double[,] data = (double[,])test.ToArray();
double x = data[0, 0];
double y = data[0, 1];
gamex = Convert.ToInt32(x);
gamey = Convert.ToInt32(y);
}

```

The Matlab code which performs the described method:

```

%
% Uses rescaling to find successive approximate locations of the window,
% finally finding the location to the pixel.
function [data]= findwindow(fullscreen, game_filename)
gametitle = imread(game_filename);
fullscreen = uint8(fullscreen);
gametitle = gametitle(:,:,1);
fullscreen2 = imresize(fullscreen, .1);
gametitle2 = imresize(gametitle, .1);
[x,y] = findwindow2(fullscreen2,gametitle2,1,1,0,0,2);
x = x*10;
y = y*10;
fullscreen2 = imresize(fullscreen, .25);

```

```
gametitle2 = imresize(gametitle,.25);

x = x*.25-6*3;
y = y*.25-6*3;
xn = x+2*6*3;
yn = y+2*6*3;
if (x<1)
x=1;
end
if (y<1)
y=1;
end

[x,y] = findwindow2(fullscreen2,gametitle2,x,y,xn,yn,3);

x = x*4;
y = y*4;
if (x<1)
x=1;
end
if (y<1)
y=1;
end

%
fullscreen2 = imresize(fullscreen,.5);
gametitle2 = imresize(gametitle,.5);
%
x = x*.5-3*3
y = y*.5-3*3
xn = x+2*3*3;
yn = y+2*3*3;
%
[x,y] = findwindow2(fullscreen2,gametitle2,x,y,xn,yn,3);
```

```
x = x*2;
y=y*2;
if (x<1)
x=1;
end
if (y<1)
y=1;
end
%
x = x-2*3
y = y-2*3
xn = x+2*2*3;
yn = y+2*2*3;
%
[x,y] = findwindow2(fullscreen,gametitle,x,y,xn,yn,1);

data = [x,y]
end

function [x,y] = findwindow2(fullscreen,gametitle,x0,y0,xn,yn,stepsize)

xmin=0;
ymin=0;
testmin = inf;

[gheight, gwidth] = size(gametitle);
[fheight, fwidth] = size(fullscreen)

if (xn == 0)
xn=fwidth-gwidth-stepsize
end
if (yn == 0)
```

```
yn = fheight-gheight-stepsize
end

for(x=x0:stepsize:xn)
for(y=y0:stepsize:yn)
test = error_fun([x,y]);
if (test < testmin)
xmin = x;
ymin = y;
testmin = test;
end
end
end

x=xmin;
y=ymin;

function f = error_fun(x)
f= test_sub(x(1),x(2),fullscreen,gametitle);
end
end

function fit = test_sub(x, y, full, game)
x = round(x);
y = round(y);
[height, width] = size(game);
sub = full(y:y+height-1,x:x+width-1);
sub = sub-game;
fit = sum(sum(sub,1));
end
```

## Recognizing pieces

To get the game state from the screenshots I chose to avoid OCR methods and just go with a statistic approach. The idea was that the histogram for a piece will be unique from the other colors. For example the mean red value for red would be different than that of the other pieces. To get the reference statistics from the game I used AForge.NET. The Framework comes with a nice application to work with images library to reference from the bot.

The first step is to collect statistics for all the game pieces. To do this, get a screenshot of all of them, then crop down to just the game pieces. Each piece has some shape surrounded by the background. To improve the statistics I chose to grab a 20 by 20 pixel box centered on the piece location. That is to crop down the 40 by 40 location to just the inner 20 by 20 and what remains is the core of a game piece.

The statistics I collected using AForge .NET.

Piece	Red Mean	Green Mean	Blue Mean
white	218.79	218.79	218.79
purple	176.7325	38.1	177.2425
blue	16.975	109.5675	204.7625
green	43.06	214.2775	75.365
yellow	227.01	197.165	28
orange	238.49	143.535	55.7425
red	242.855	31.28	62.07

Great, we now have readily available statistics to match a game piece too. Further, the statistics are calculated by the same library so matching should be good.

Matching a piece to the statistics is straightforward using a Root Sum Square error approach. For a statistic at a location we calculate the error to each piece and take the one with the smallest error. The code will look something like:

```
double bestScore = 255;
double curScore = 0;
foreach (KeyValuePair<Color, List<double>> item in statReference)
{
    curScore = Math.Pow(item.Value[0] / 255 - stats.Red.Mean / 255, 2)
        + Math.Pow(item.Value[1] / 255 - stats.Green.Mean / 255, 2)
        + Math.Pow(item.Value[2] / 255 - stats.Blue.Mean / 255, 2);
    if (curScore < bestScore)
    {
        PieceColor = item.Key;
        if (item.Key == Color.YellowGreen)
            PieceColor = Color.Yellow;
        if (item.Key == Color.MediumPurple)
            PieceColor = Color.Purple;
        bestScore = curScore;
    }
}
```

In this implementation a score, or error, is calculated by differencing the mean color values from an initial value and then performing the RSS on that value. The score with the smallest error indicates the piece which has the closest match. With pieces 20 by 20 and a cheap RSS being used, this method parses the game state very fast and accurately. Note that it is important to crop down to the 'core' of the game piece as the background board tends to smooth out the statistics, especially between yellow and orange.

Finding the statistics for the pieces can be done like so, where `game` is the current screenshot cropped game and `gGamePieces` is an array of initialized Graphics objects.

```

for (int i = 0; i < 8; i++)
  for (int j = 0; j < 8; j++)
  {
    gGamePieces[i, j].DrawImage(
      game,
      new Rectangle(
        new Point(0, 0),
        new Size(20, 20)),
      new Rectangle(
        new Point(gameStartX + (i * 40) + 10,
          gameStartY + (j * 40) + 10),
        new Size(20, 20)),
      GraphicsUnit.Pixel);
    ImageStatistics stats = new ImageStatistics
      (bmpGamePieces[i, j]);

    ImageStatisticsYCbCr stats2 = new ImageStatisticsYCbCr(
      bmpGamePieces[i, j]);

    pGame[i, j].Parse(stats, stats2);
  }

```

I chose to store the piece reference statistics in a dictionary and initialize as follows:

```

private void BuildReference()
{
  //RedMeanB GreenMeanB BlueMeanB
  statReference = new Dictionary<Color, List<double>>();
  statReference.Add(Color.White,
    new List<double> { 218.79, 218.79, 218.79 });
  statReference.Add(Color.Purple,
    new List<double> { 176.7325, 38.1, 177.2425 });
  statReference.Add(Color.Blue,
    new List<double> { 16.975, 109.5675, 204.7625 });
  statReference.Add(Color.Green,
    new List<double> { 43.06, 214.2775, 75.365 });
  statReference.Add(Color.Yellow,
    new List<double> { 227.01, 197.165, 28 });
  statReference.Add(Color.Orange,
    new List<double> { 238.49, 143.535, 55.7425 });
  statReference.Add(Color.Red,
    new List<double> { 242.855, 31.28, 62.07 });
}

```

We now have a way to locate the game window and identify the game pieces on the board. We have a great game state, great.

## Determining a move

For this I did not do anything fancy. With our known game state I simply search the board for any location where swapping two pieces resulted in a three piece chain. I didn't bother to add any checks for large combos as I did not see this as accomplishing anything other than improving a score.

There was an additional component to determining the moves. After doing a simple implementation I noticed an interesting dynamic feedback on the game. The bot would move fast enough to try and create combos where they did not exist. For example, it would swap a piece and while it was moving notice that it also moved another piece a combo would be created while the pieces moved. The game does not allow you to move two pieces to create a combo, so this just resulted in a bunch of spinning pieces that oscillated back and forth. I chose to damp the board to prevent this.

To do this, I used an eight by eight double array to keep track of a delay. Any time a move was performed I added 550 to that location and 275 to locations around it. The values are delay values in milliseconds. In this way the bot will not move pieces away from an action in that area. Every tick of the engine would subtract the elapsed time from the entire array. Game moves are only made to locations where the delay value is zero (negative values are kept to zero). This allows us to simply call all possible moves without destroying previous moves. Note that for each tick I send moves for every possible match on the board. I don't perform a move and exit the loop; I send them all without a Sleep. The game does remarkably well at handling this.

Running the bot is similar to a game loop. I just created a timer which ticked at 50 milliseconds and stopped after 63 seconds (typically three seconds from the game pausing for specials).

## Making the move

Making the move involves an Interop call to user32. Here I refer you to Mike Vallotton's post. One 'go here, if you are running on x64 you need to change the offsets in the class or nothing will happen. Als suggest putting in a Thread.Sleep while you test; the last thing you want is your mouse moving out o control with a bug, clicking around your Facebook page at 20 Hz.

```
[StructLayout(LayoutKind.Explicit)]
private struct INPUT
{
    [FieldOffset(0)]
    public int type;
    [FieldOffset(8)]
    public MOUSEINPUT mi;
    [FieldOffset(8)]
    public KEYBDINPUT ki;
    [FieldOffset(8)]
    public HARDWAREINPUT hi;
}
```

Of course you will need to figure out the pixel offsets for the game board within the game and how to calculate piece locations as well. Something like the following where the `gameStart` values are found by an image editor (the game seems to change every once in a while as they update) and the `game` locati come from the title screen locating.

```
var x1 = i1 * 40 + gameStartX + gameX + 20;
var y1 = j1 * 40 + gameStartY + gameY + 20;
var x2 = i2 * 40 + gameStartX + gameX + 20;
var y2 = j2 * 40 + gameStartY + gameY + 20;
```

```
SendInputClass.Click(x1, y1);
```

```
SendInputClass.Click(x2, y2);
```

Good luck, and have fun.

---

## Related Links

[Interpolation/Curve Fitting in C++ with the Fenics Project in 1D, 2D and 3D \(11/23/2011\)](#)  
[XML Serialization and Tiered Architecture \(5/10/2008\)](#)  
[Setting Up a Blog or Personal Website on a Home Computer with Vista \(7/16/2008\)](#)  
[Seam Carving for Resizing using SEAMonster \(6/5/2008\)](#)  
[Allowing for Dynamic Embedded View Substitution with MVC \(1/2/2009\)](#)

Print | posted on Sunday, September 05, 2010 3:43 PM |

## Feedback



### # re: Creating a Bejeweled Blitz Bot in C#

Awesome!! :) Thanks

9/12/2010 12:29 AM | Brett



### # re: Creating a Bejeweled Blitz Bot in C#

Nice Job. By the way, how do you determine when the game ends. Does your program waits "hotkey" or did you use a more "intelligent" approach?

10/15/2010 6:18 PM | Royce



### # re: Creating a Bejeweled Blitz Bot in C#

Fortunately they game tends to take a certain amount of time on average, about +6 seconds just had a timer that stops the bot at a certain point. A more correct way would be to read the progress bar at the bottom of the screen (should be simple to pull from the screen scrape).

10/17/2010 8:40 PM | Charles



### # re: Creating a Bejeweled Blitz Bot in C#

I noticed in your code to find the window you have two findwindows. Did you split your matlab code between findWindow and findwindow and reference them separately?

```
MWArray test = (MWNumericArray)findWindow.findwindow(bmpinp, fileinp);
```

11/15/2010 6:48 AM | Charles



### # re: Creating a Bejeweled Blitz Bot in C#

There are two functions within the matlab script, but the second function is called from the n script by findwindow. findwindow2 is called a few times by findwindow for successive resolut the image. findwindow is called just once from c#, by the line you show.

11/15/2010 7:43 PM | Charles



### # re: Creating a Bejeweled Blitz Bot in C#

I wrote a similar bot in c#. I like your approach for finding the board. I just have a UI that allow you position a grid over the board to identify it's location. The rest of the approach was very similar to yours. I did match on all 3, 4, 5 gem permutations and rank order the best moves. first look for hyper gems and match the most colors on the board to take them out. Hyper gems are very hard to identify because they spin and constantly change color. Also, when you get "Blazing Speed" the colors of the gems change as there are flashes and statistics change for identifying yellow and orange. Thanks for posting your article. It's nice to see that other people thinking the same things.

12/4/2010 1:24 PM | Mark Lindell



### # re: Creating a Bejeweled Blitz Bot in C#



The changing game states and holiday theming definitely makes keeping up to date with the tricky. At one point they even changed the sizes. Nice work on ranking the best moves first!

12/8/2010 8:56 PM | Charles



**# re: Creating a Bejeweled Blitz Bot in C#**

is there a way to dowload this?

1/1/2011 5:58 AM | AWESOMER THAN YOU



**# re: Creating a Bejeweled Blitz Bot in C#**

@ Awesomer Than You: "I will not give a direct download"

1/8/2011 10:36 AM | Kenny



**# re: Creating a Bejeweled Blitz Bot in C#**

how would I save the file?

2/7/2011 5:26 PM | Dylan



**# re: Creating a Bejeweled Blitz Bot in C#**

like what is the extension called?

I copied all the code into notepad, so how do I save it?

And how do I stop the bot?

2/7/2011 5:29 PM | Dylan



**# re: Creating a Bejeweled Blitz Bot in C#**

Some of the code is matlab, and some is in c#. For matlab the extension is '.m' and for c# it is '.cs'. You will need matlab and visual studio to run the code, however. I have also omitted some 'trivial' parts of the code, so simply copy pasting into a file will not yield a bot.

2/7/2011 5:40 PM | Charles



**# Canada Goose Parka**

Canada Goose Parka in superior material are selling in our online store. Canada Goose Jacket made of 100% wool, good flexibility, lint-free and never shrink, fashionable and elegant design you feel interested in it, do not hesitate to place an order on our site. If you want to know more information about Cheap Canada Goose Expedition Parka, you can contact us at any time. In the high quality and beautiful design Canada Goose Parkas Jackets will be your favourite. Pay attention to our site please, you will find more surprise here every day. Canada Goose Chilliwack Jacket Sale on our website is waiting for your coming!

12/14/2011 7:37 PM | Canada Goose Parka



**# re: Creating a Bejeweled Blitz Bot in C#**

Some specialists tell that home loans help a lot of people to live the way they want, because they are able to feel free to buy needed stuff. Moreover, various banks give consolidation loan for young and old people.

12/25/2011 4:33 PM | MARIEAllen30



**# re: Creating a Bejeweled Blitz Bot in C#**

At present, the [ringtones](#) and [nextel ringtones](#) are very usable, but, there are people, which continue to order the great thought like your.

12/31/2011 6:37 PM | Talley33JUNE

**# re: Creating a Bejeweled Blitz Bot in C#**

If you are willing to improve your knowledge close to this good post, look for [thesis writing](#) s or [dissertation writing](#) and order sureme masters thesis over there.

1/4/2012 2:29 PM | Wyatt25Cassandra

**# re: Creating a Bejeweled Blitz Bot in C#**

I have ne'er used such services earlier, hence I was a bit in question if I could really get solic essay that was not plagiarized. My teacher said that she liked new conceptions and well-four intentions I presented custom essay. She is very finicky and seldom says somethinglike that my paper. I am very gladsome that I have launch this [writing service](#). The quality was truly beneficial and I am gladsome with the available prices.

1/4/2012 3:53 PM | Parker22JEWELL

Copyright © Rodbourn