Candy Crush/Bejeweled bot

1. What is your program? Give a description of what your program is going to do.

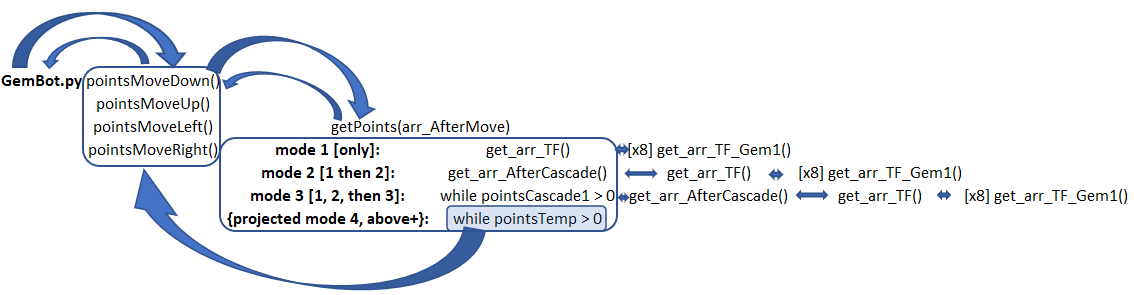
A bot that plays a free online version (www.DigBejeweled.com) of the popular puzzle game by arranging 3 or more gems in a row and/or column on an 8 by 8 board.

2. How is it going to work? Describe how you expect the program to perform its operations.

(a) The bot will import the following modules:

from pynput.mouse import Button, Controller # pip install pynput https://pypi.org/project/pynput/ [Win + Mac]  
from pynput import keyboard  
import time # built-in  
from PIL import ImageGrab # pip install Pillow https://pypi.org/project/Pillow/ [Win + Mac + Linux]  
from python\_imagesearch.imagesearch import imagesearch # pip install imagesearch https://pypi.org/project/imagesearch/ https://brokencode.io/how-to-easily-image-search-with-python/ [Win + Mac + Linux]  
import numpy # [comes with imagesearch] pip install numpy https://pypi.org/project/numpy/  
import time # built in  
import sys # built in  
import argparse # built in  
import ArrOps # custom

The bot can be launched from a terminal/CMD with a -m/-mode [1,2,3] argument to change how it calculates points: mode [1] considers only the points of the initial move [2] adds points from one cascade [3] also adds points from any remaining cascades. I have yet to try out the Max Tree RA, or the human like search for matching gem pairs in order to optimize the code. The difference in how fast the bot makes moves is noticeable between the three currently implemented modes.



The bot mostly works autonomously (without user input). The user can change modes and stop the bot. In order to prevent user created obstructions, pressing any key will stop the bot. The bot keeps track of levels and saves the current level to a text file in its directory for retrieval upon resumption after a pause. The user could attempt to change the contents of the file, but several checks are performed to ensure uninterrupted operation: error handling and a safety fail reset (starting the game over from level 1 will reset the count to 1). The current version of the bot has code commented out to stymie operation at a level predefined level in order to keep up with the timer for a longer period and attempt for a high score. I’ve commented the code out because additional testing is needed. Additionally, unfortunately, score submission seems to be disabled, so I’m unable to get on the scoreboard. But the bot could be repurposed and used on similar games.

Speaking of repurposing, I tested the bot on different browsers and resolutions and had to redo the way it recognizes the different colors of gems from exact RGBs to an RGB range. This should hopefully work on your system/improve compatibility.

The bot is broken down into a main file and a support/array operations module [GemBot.py and ArrOps.py]. There are also a few .PNG images required for it to work. More on that on the next page.

Additionally, there are docstrings attached to functions and modules, and some hopefully helpful comments throughout the code.

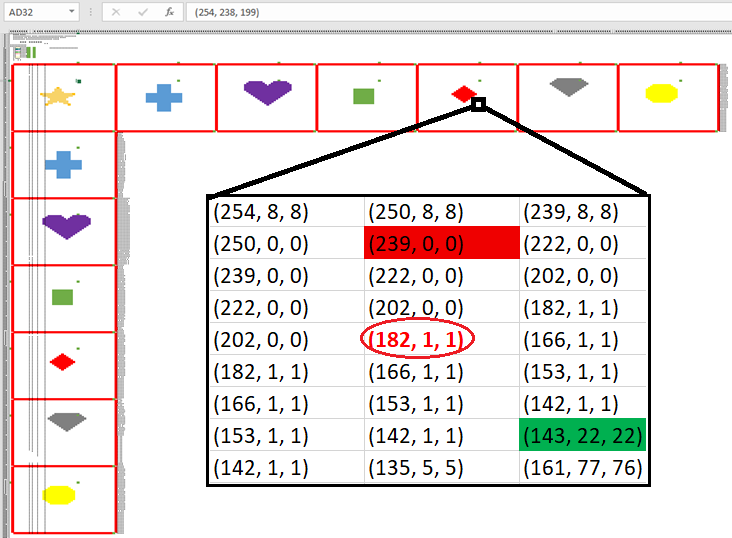
Once the game is started and the timer bar is running, the user can press the cog wheel/options/pause menu button below the score to pause the game and thus get additional time to start the bot. Once launched, the bot will recognize the pause menu and close it before playing. In order to do that, the bot will attempt to find the board using an image called ‘anchor.png’



The anchor image serves two purposes: 1) see if the game is started and 2) locate the top left gem using the x, y coordinates of the anchor plus a pre-measured offset of pixels:

If anchor.png is not found the bot will print a message to the immediate window and stop. The search for anchor.png is done once. If anchor.png is found, the bot also checks on the lvl1.png and if that fails, try to grab the level count from the local levelcount.txt text file (if the Options menu was open to begin with). The bot grabs multiple pixels from each gem on the board in order to identify them using ranges of the pixels’ RGB values. I’ve measured the distance between gems (geometric squares) to be 45 pixels apart (d = 45 in the code). Also grabbed are pixels for recognizing a game over/results screen and level up screens (for counting levels).

I initially pulled out all the pixels’ RGB values of gems into Excel using my helper bot (also written in Python) to find a pixel in the same position relative to the individual gem’s square that’s unique to each of the 7 gem types. Since then I’ve added 7 more pixels per gem with one of each color among a pool of distinctly recognizable set (for example, red’s location has a pool of RGB’s with a high contrasting red value (182) as compared to green and blue values (a value of 1 for both). Hope this helps make gem recognition stable, but unfortunately, I can’t say it will work on your system without testing it. Regardless, I will share a video of how the bot works and how the algorithm can be tested by calling functions and passing prefilled Numpy arrays to the bot; that’s of course no fun, so I’m hoping for the best.



I’ve tested the pixel locations and cross referenced the logic with values from all 7 locations of all the gems so hopefully there is enough space for some variation.



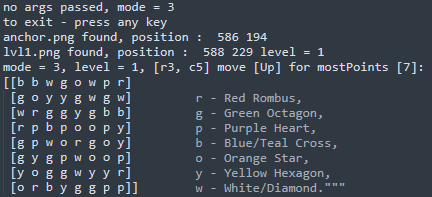
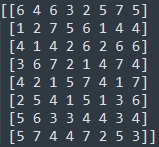
I’ve tested the logic in Firefox and Chrome, and hope it works on your system, but more tests are probably warranted because of the snag I hit with varying RGB values between browsers. Resolution didn’t seem to make a difference, at least not under visible conditions. It helps to have the window of the browser with the game maximized. Regardless, more testing could be performed.

The values for each gem will be loaded into an array and one by one checked for a possible match if the gem is moved: to the left, right, up, or down. As matches are found, the points (# of gems in a match) and the coordinates (from which row column and to which row column should a gem be moved to achieve the match) will be saved in variables. If a better match is found, these variables will be updated with better values. Once the end of the board is reached, these variables will hold the coordinates for the match with the most points. The mouse will be moved and clicked using the coordinates (because the gems are a set 45 pixels apart from each other, it’s easy to convert the row/column location in the array into x,y coordinates on the screen since we know the x,y of the top left gem which we got through anchor.png’s x,y coordinates).

(b) The user might try to break the bot by intentionally obscuring part of the board or use the mouse in an attempt to see what would happen. The program will hope the user is its ally. If 'anchor.png' is found the bot will attempt to locate the gems as part of its mode of game play. If not all gems are recognized the bot will continue trying to locate them (perhaps the user will move the rest of the board into view), additionally I might implement a partial board recognition so the initial click on the board brings the rest of the game window into view. If the mouse is moved around, the bot may miss a move but will fight the user for mouse control (pressing any key on the keyboard should stop the bot and pause the game). Missed moves do happen sometimes, perhaps because a matching RGB value is grabbed while gems are in transit/being removed. The bot could get confused if the user paints the entire screen (or even just a part that covers the gems) as one of the gems RGB values during a pause; to prevent confusion the bot clicks into the gameboard if it can recognize it and once it’s playing any keystroke stops the bot should an endless loop develop.

3. How will the data be represented? You do not have to know how to handle the data yet but do think about what kind of data you expect to be working with.

Data will be represented as an 8 by 8 NumPy array of integers from 0 to 7 (zeros are unrecognized gems, either because something went wrong or because the board is repopulating during an ‘out of moves’ event). The user could watch the printed-out messages on the state of operations and is presented with a more readable array view (instead of ints – letters corresponding to gem colors). The letters chosen are matched to the initial letter of each color – this legend won’t be displayed and is for reference here only. For readability, row and column values start from 1



4. Describe any problems you may experience and how you plan on handling it. It is good to think about what might happen, so you are prepared to handle it. [these are from the midterm; I’ve mentioned solutions above]

1. The RGB values could be slightly off for some gems resulting in an unrecognized gem value in the array.

* I will test on a different system and with different resolutions to get ahead of the issue. If there are only one or two unrecognized gems, the bot would still work but I would prefer if it worked at 100% so there isn’t a situation where the bot thinks it’s out of moves when there are still moves available.
  1. Additionally, as the gems disappear and are repopulated, there might be a small chance of them being misrecognized at just the right time as they fall or transition.
* I will need to check, somehow, if this is a significant issue. One way could be to grab screenshots of the board during transition periods and try to find my predefined RGB values in the usual x,y locations.

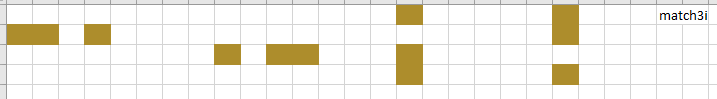
2. The user might intentionally obscure a part of the board or use the mouse in an attempt to see what would happen.

* I will leave this up to the user to have fun with. The program will hope the user is on its side.

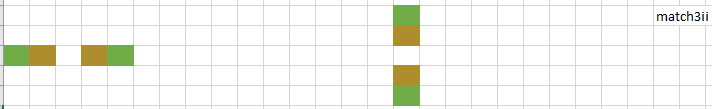
3. The timer might run out ending the game on later levels with fewer and fewer seconds on the timer with each new level.

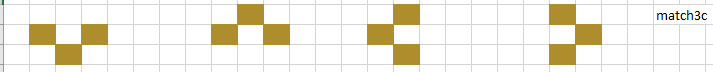
* I will attempt to make the bot smarter. By implementing functions and a class for easier readability and writability, I will try to have the program create arrays of the board as if after each move is made to see what matches will happen unintentionally as a direct result of each possible initial move (cascades). Additionally, I could further test for more moves even thereafter, even though the addition of new gems could likely create better overall moves than initially planned with the limited board.
* For now, I have a bot that’s all in one class, it only looks at the initial points of each possible match, and its code could use a breaking down into functions for easier management.

Oh, and just for fun, here are the possible matches I was able to work out in Excel:

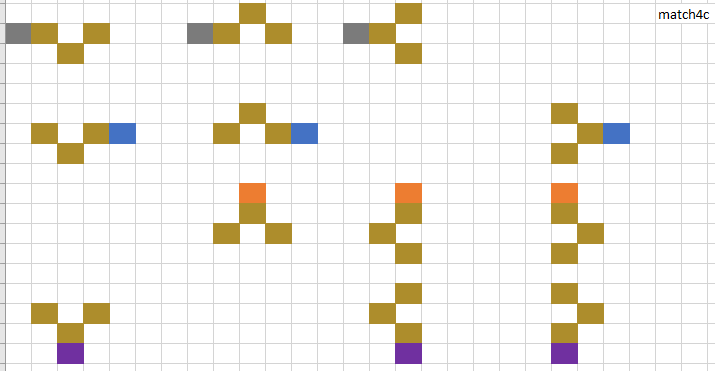


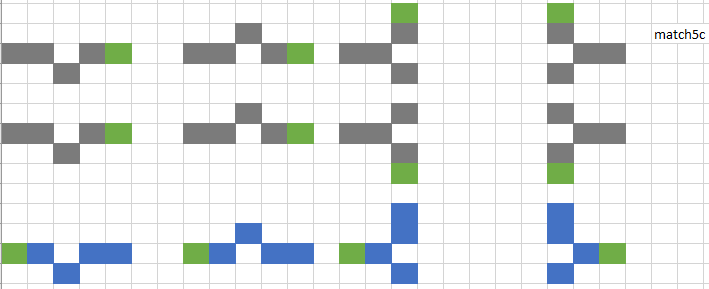
3i is part of the name because there are 3 gems in this match and the initial configuration looks like a lower case ‘i’ or an exclamation point



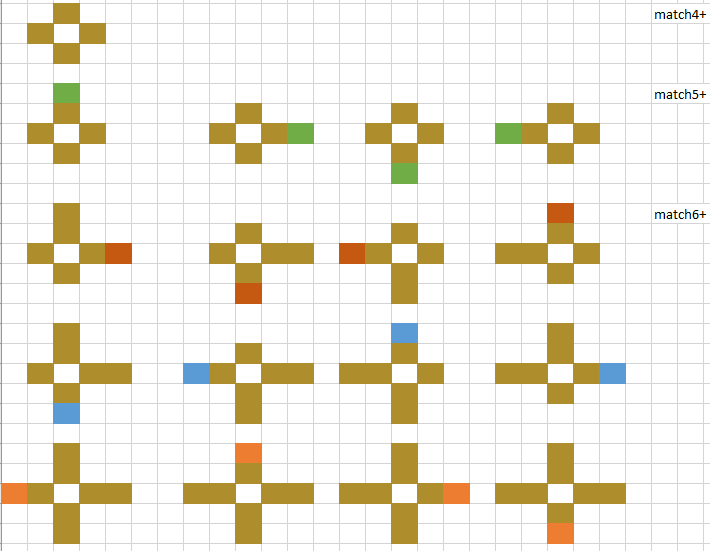


This configuration looks like the letter ‘c’. The different colored block identifies a newly added block to the previous/base configuration/pattern.





The base/initial version of these patterns looks like a plus sign.



And here’s a close up of the RGB pull of one of the seven gems from the previous screenshot. The RGB values were transferred automatically, using Python. 