CPSC 231 ASSIGNMENT #5

Taffy Tangle

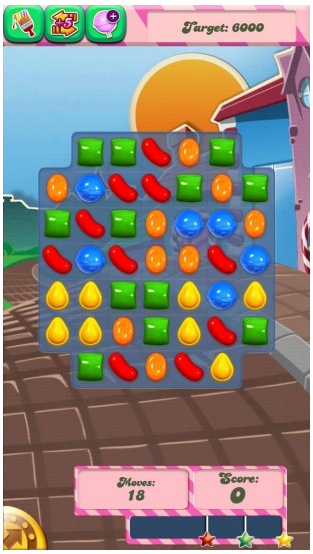

Figure 1: Candy Crush Saga is an incredibly popular match-three game that has provided billions of hours of entertainment to the world.

If you've ever played casual games on your mobile device, or even on the internet through your browser, chances are that you've spent some time with a "match three" game. Of the recent varieties, *Candy Crush Saga*, published by King in 2012 and shown in Figure 1, has been one of the most wildly successful titles for mobile platforms.

The goal of this assignment is to create our own simple variant of a match-three game, which we'll call "Taffy Tangle." (Feel free to call yours whatever you'd like. We just made this one up to create a consistent alliteration through the problems of this assignment.)

The base mechanics of a match-three game are quite simple. A (usually) rectangular board is initially filled with a grid of randomly-selected objects from a prescribed set. There are usually five or six varieties of these objects (be they candies, jewels, or other things) which are visually differentiated by their colour and shape.

After the initial board set-up, the player makes a move by swapping two adjacent objects so that a line (horizontal or vertical) of at least three objects of the same kind is made. Then all lines of at least three adjacent objects of the same kind are eliminated and a score proportional to the number of objects eliminated is added to the player's total. Remaining objects "fall" from top to bottom to fill the gaps created by the eliminated objects, and random new objects are created at the top to fill up the board again. Any new lines of three or more matches are again eliminated, and the falling and random creation process is repeated until there are no more matches. The player then continues to make moves until a certain objective (*e.g.* score) is met, or no valid moves remain.

It's hard to make sense of such a verbal description, so if you haven't played Candy Crush or Bejeweled before, we encourage you to try them out. They are free to play, though be forewarned that they can be quite addictive. Don't forget about your assignment!

Figure 2: Bejeweled is another popular variant of the match-three game whose first version was released a full decade before Candy Crush Saga.

Due Dates

Individual component	Friday, November 9, 11:59 PM
Paired component	Friday, November 23, 11:59 PM

Individual Component

We're now down to a single problem for each component of the assignment! By this point in the course, we trust that you're a well-trained computer scientist[1] who doesn't need our help anymore to apply the principles of top-down design and stepwise refinement to solving computing problems. It is entirely up to you now to exercise your skills as a computer scientist and devise algorithms to solve the problems, then plan out the structure and layout of your program, before you implement your solutions as Python programs.

We believe these initial steps are best done with a pencil and a pad of paper, and we strongly recommend that you spend at least half of your time for each problem on this step. Although you're not required to hand this in, don't say we didn't warn you if you decide to skip this step!

You can use the exact same programming environment you set up for previous assignments to complete this assignment. Create as many modules or classes as you wish to help you organize your code, but be sure to submit all your `.py` files and indicate clearly which script (program file) your TA should run for each problem. Note also that the bonus problem for this assignment is attached to the individual component rather than the paired component.

[1] Or perhaps an expert computer scientist in training?

Problem 1: Tic-Tac-Toe

Tic-Tac-Toe is another classic two-player game, played on pencil and paper, that most of us surely learned before elementary school. It is played on a three-by-three grid of squares as shown in Figure 3, and the object of the game is to claim a row of three squares on the board, either horizontally, vertically, or diagonally.

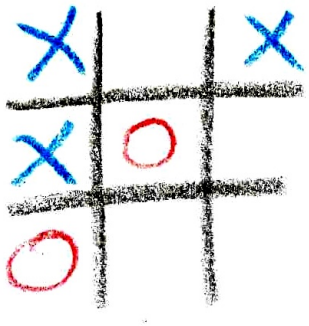

Figure 3: A Tic-Tac-Toe game in progress.

The player who goes first is known as $\times$ and the second player as $\bigcirc$. The first player starts by claiming one of the nine squares by marking it with an $\times$. Then the second player takes an open square by marking it with an $\bigcirc$. The players continue to alternate turns this way until one player, the winner, makes a line of three squares in a row with his or her symbol, or until all nine squares are filled. If the grid is filled but neither player has a line of three, the game is a draw (also known as a "cat's game").

Create a visual and interactive Python program that will facilitate a game of Tic-Tac-Toe between two human players. Draw the board on the screen and wait for the first player to click in one fo the nine squares, then draw an $\times$ in that square. Next, allow the second player to do the same with an $\bigcirc$. Write a message in your game's window to indicate whose turn it is. Continue until one of the players wins, or the game ends in a draw. Indicate the the final result of the game in the graphics window, then wait until the players close the window (or ask if the players would like to play another game).

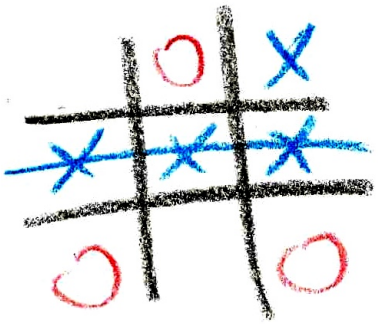

Figure 4: Player $\times$ has won the game by forming a line of three.

Inputs: A series of mouse clicks from two human players taking turns to place their ×'s and ○'s.

Outputs: A Tic-Tac-Toe board (which looks like a #) drawn in a window on your screen with ×'s and ○'s drawn as they are placed.

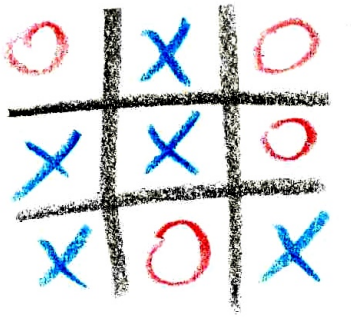

Figure 5: The game is drawn when all squares are filled and neither player has a line of three.

Bonus Problem: Computer Tic-Tac-Toe Player

Program a computer artificial intelligence capable of playing Tic-Tac-Toe so that you can create a single-player version of your game from Problem 1. Your AI must have at least two distinct difficulty levels:

Hard should be able to play a perfect, or near-perfect, game of Tic-Tac-Toe. It should be able to force a draw against the human player every time, and take a win if the player makes a mistake.

Moderate should make a mistake or sub-optimal move once in a while to give the human player a chance of winning, but still play a decent game. This difficulty ought to be great for small children.

You may either use a command line argument to select the AI difficulty level or have the player click on a "button" to choose the difficulty at the beginning of the game. Simulate a coin toss to decide whether the human player will be × or ○. Then play the game out interactively just as you did for Problem 1, indicating the end result in your game's window as appropriate.

Inputs: A series of mouse clicks from the human player to place his or her ×'s or ○'s.

Outputs: A Tic-Tac-Toe board drawn in a window on your screen with ×'s and ○'s drawn as they are placed by the human and computer players.

Paired Component

We *strongly* encourage you to work with a partner to complete the remainder of this assignment. It will likely be extremely helpful to have someone with whom to talk through the strategies for solving this problem, and its sub-problems, as you do the top-down design and program decomposition for this component.

Once again, your partner may be any other student in the class, but remember that you must work with a *different partner* for each assignment! If you are solving this problem as a pair, one submission is sufficient for both students. If you've already exhausted all your friends in this class through the first four assignments, perhaps you can ask your TA to help you with match-making. Maybe even try hunting for your best movie match from Assignment #3.

Problem 2: Taffy Tangle

Your goal for this problem is to a visual and interactive Python program that allows a human player to play our variant of the match-three game, Taffy Tangle (or whatever you've christened your game). Our basic variant of the game will be played on a board measuring 9 rows by 7 columns with six different kinds of "taffies", as shown in Figure 6. The player makes moves by swapping adjacent taffies so that a line (horizontal or vertical) of at least three objects of the same kind is made. The game ends when no more valid moves remain.

Your program must be able to perform the following functions to satisfy all requirements of this problem:

1. Generate an initial board (9 rows by 7 columns) filled with random selections of the 6 different kinds of taffies, and draw the board to the screen.
   - Each kind of taffy must be visually distinct from the others in both colour and shape. (Yours do not need to look like the ones shown as long as they meet this requirement.)
   - The initial board should not contain any lines (horizontal or vertical) of three or more taffies of the same kind in a row.
2. Allow the player to make moves of swapping two adjacent taffies on the board by first clicking on one taffy, then clicking on an adjacent one, while the game is not won or lost.
   - After the first click, indicate which taffy is "selected" in some visual way (*e.g.* as shown in Figure 6).
   - Cancel the move if the player then clicks a second taffy that is not adjacent to the first, selected one.
   - Forbid or undo a move that would not result in creating a new match of three or more taffies.
3. After a valid move from the player, repeatedly eliminate taffies that are matched by following the sequence of steps below, until no more matches are found. Update the visual board and pause briefly after each step so that the player can see what happened.
   - Eliminate all taffies on the board that are part of a horizontal or vertical line of three or more taffies of the same kind (Figure 8).
   - Allow taffies in each column to "fall" so that it contains a continuous stack of taffies from the bottom, and no gaps remain in the middle (Figure 9).
   - Refill each column with new, random taffies at the top until all columns again contain 9 taffies (Figure 10).
4. Keep track of the player's score and display it visually within the game window. In our basic variant, the score will simply be the

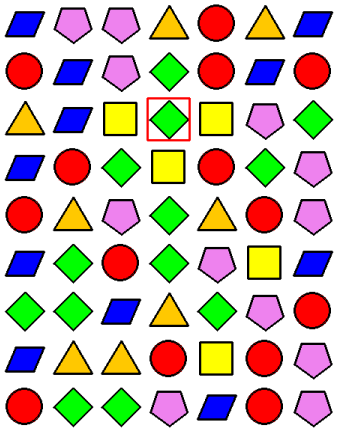

Figure 6: The initial board of taffies in our game, with one taffy selected (inside the red box), ready to be swapped.

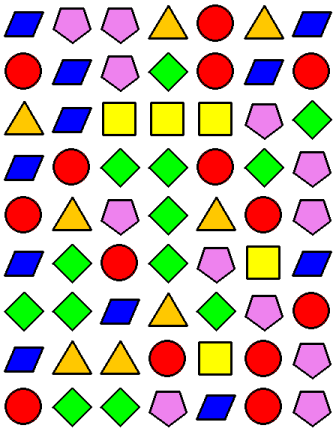

Figure 7: The green taffy has been swapped with the yellow one below, creating two lines of three matches.

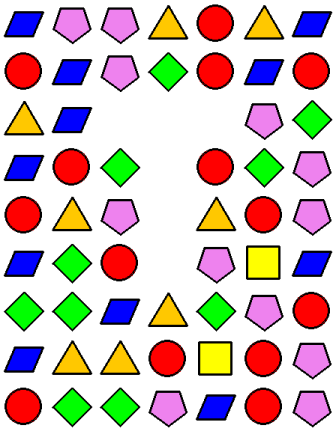

Figure 8: The two lines of matching taffies have been eliminated, leaving a gap in the middle.

number of taffies the player has eliminated. You may use a more sophisticated scoring mechanic if you like.

5. Detect when no more valid moves remain, then display a "game over" message with the player's final score in your game window. If you don't like losing all the time, you may optionally set a win condition or goal (*e.g.* reaching a certain score, perhaps within a given time limit) and end the game with a victory message once the condition is met.

It is entirely up to you to decide how you want to organize your program into functions, modules, and classes. While it is not strictly required that you use any or all of these constructs, we are pretty certain that you will want to use them if you plan to complete this problem while staying sane. Don't hesitate to ask the instructors if you need advice for any part of the design process. Good luck!

Inputs: A series of mouse clicks from the player, desperately trying to swap taffies on the visual game board.

Outputs: A colourful visual display of taffies being swapped, disappearing, dropping, and reappearing. Don't forget to pause briefly (*e.g.* for half a second) after each step!

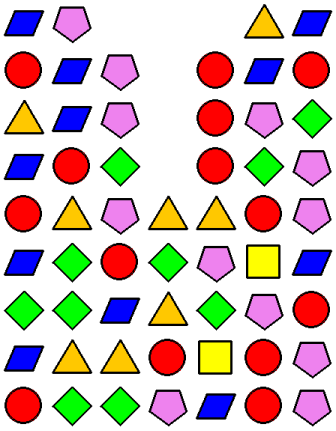

Figure 9: The taffies above the created gap have fallen down to fill the empty space so that every column is a continuous stack of taffies from the bottom of the board.

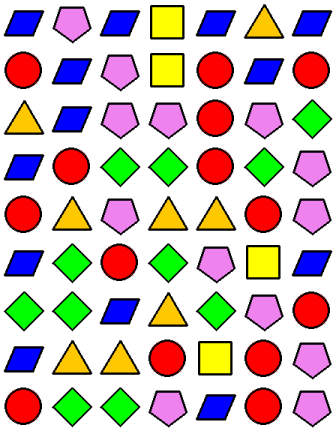

Figure 10: Random new taffies have been created to fill every column to the top again. Note that the three red taffies should now be eliminated and the falling and refilling process repeated until no more matches remain, before the player makes his or her next move.

Creative Bonus

This assignment is just full of ways in which you can creatively enhance your "Taffy Tangle" game. There are so many open-ended possibilities that we've deliberately omitted a bonus problem from the paired component so that you may spend your time on this bonus. That is, if you miraculously have any time left!

Naturally, the more enhancements you add, the better your game becomes to show off, and the more fun it is to play. Rather than try to list specific possibilities for enhancements here, we'll simply suggest that you play some of the other variants of match-three games for inspiration.

As with the previous assignment, your TA will have liberty to award this bonus to the game that he or she deems the most creative submission. This could be judged in terms of aesthetics, design, quality of experience, entertainment value, or any combination of the above. Do whatever you'd like to improve the gameplay experience, though if you deviate much from the "standard" Taffy Tangle of Problem 2, please submit a separate program for this bonus.

Java Bonus

This bonus is always here, but you might have noticed that it's getting harder and harder to obtain. Is it still worth it? As with the

previous assignments, we will award you another bonus if you submit a correct Java program for Problem 2, in addition to your Python program.[2] This goes without saying, but your Java program must satisfy all the base requirements of the problem as well.

[2] Note that you should consider your Python program to be your de facto solution for this assignment. You will not get credit for this component unless your Python solution is correct.

You may set up your Java program however you'd like, as long as it is convenient enough for your TA to run them on the CPSC lab machines. There exists a version of our course text that uses the Java programming language, titled *Computer Science: An Interdisciplinary Approach*.[3] It has corresponding support libraries that can be found at `https://introcs.cs.princeton.edu/java/home/`. If you're not sure where to start with Java, you may find it most convenient to follow the instructions from that booksite. You may also use functionality from their Java Booksite Library if that helps you to create Java-language equivalents of your Python programs.

[3] Robert Sedgewick and Kevin Wayne. *Computer Science: An Interdisciplinary Approach*. Addison-Wesley, 2016

Submission

When you've completed the individual component, submit your Python program (`.py` file or files) for Problem 1. After completing the paired component, submit all the files needed to run your program for Problem 2. Please indicate clearly, in the file names or otherwise, which Python file to run for each problem. If you completed the Java Bonus, submit the source of your all your Java classes (`.java` files) needed to run your problem.

Use the University of Calgary Desire2Learn system[4] to submit your assignment work online. Log in using your UofC eID and password, then find our course, CPSC 231 L01 and L02, in the list. Then navigate to Assessments → Dropbox Folders, and find the folder for Assignment #5 here. Upload your program for Problem 1 (and the bonus problem if you completed it) in the individual folder, and your other program source files in the paired folder (if your partner hasn't already done so). One submission will suffice for each pair of students.

[4] `http://d2l.ucalgary.ca`

If you are using one or more of your grace "late days" for this assignment, indicate how many you used in the note accompanying your submission. Remember that late days will be counted against *both* partners in the paired component. If you completed any of the bonus problems, please indicate that here as well.

Credits

Although we technically came up with the same idea independently, Ben Stephenson has also used a variant of the match-three game, called "codeCrusher", for an assignment in the CPSC 217 course here at the University of Calgary. As one might naturally expect, Taffy Tangle is a little more intense than codeCrusher.