CSc 110, Autumn 2017
Programming Assignment #9: Tiles (20 points)
Due Tuesday, November 7, 2017, 7:00 PM

thanks to Mike Clancy of UC Berkeley and Marty Stepp of Stanford

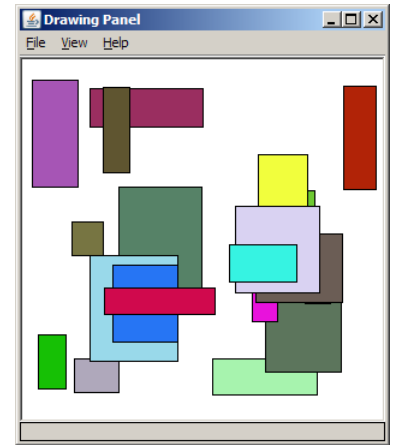## Part 1:

This project will give you insights into how operating systems manage multiple programs' windows. Its implementation focuses on using lists. Turn in a file named `tile_manager.py`. You will need `tiles.py` and `DrawingPanel.py` from the Homework section of the course web site; place them in the same folder as your program.
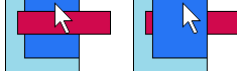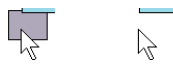
Program Description:

In this assignment you will write the logic for a graphical program that allows the user to click on rectangular tiles. To test your code, run `tiles.py`. It imports the functions you will write in `tile_manager.py`.

You should store all of the information needed to draw each tile in a **tuple** and store one tuple for each tile in a **list**. The order tiles are stored in the list should determine their drawing order. For example, consider the tall tile that overlaps the wide tile in the upper left corner of the screenshot at right. The two tiles occupy some of the same (x, y) pixels in the window but the wide one was drawn first because it occurred before the tall one in the list. When the tall tile was drawn later, it covered part of the wide tile. The list's ordering is called the 3-dimensional ordering or *z-ordering*.

The provided code reacts to mouse and key presses. You must write the functions listed on page 2. Your code will be called by the provided code to create and display the tiles and to change what is displayed when a key or mouse button is pressed. Depending on the kind of input, different actions occur:

- If the user clicks the **left mouse button** while the mouse cursor points at a tile, that tile is moved to the very *top* of the z-ordering (the end of the tile list).

- If the user clicks the **right mouse button** while the mouse cursor points at a tile, that tile is moved to the very *bottom* of the z-ordering (the start of the tile list).

- If the user clicks the **left mouse button** and holds down the **Shift key** while the mouse cursor is pointing at a tile, that tile is removed from the tile list and disappears from the screen.

- If the user clicks the **right mouse button** and holds the **Shift key**, *all tiles that occupy that pixel* are removed from the tile list and disappear from the screen.

- If the user types the **N key** on the keyboard, a new randomly positioned tile is created and added to the screen.

- If the user types the **S key** on the keyboard, the tiles' order and location are randomly rearranged (*shuffled*).

If you use a Mac with a 1-button mouse, you can simulate a right-click with a Ctrl-click (or a two-finger tap on the touch pad on a Mac laptop).

If there is no tile where the user clicks, nothing happens. If the user clicks a pixel that is occupied by more than one tile, the top-most of these tiles is used. (Except if the user did a Shift-right-click, in which case it deletes all tiles touching that pixel, not just the top one.)

Note that **the code to detect mouse clicks and key presses is provided for you**.

Implementation Details:

Your `tile_manager` program should **store a list of tuples called `tiles` and a `DrawingPanel` called `p` as global variables** (this means variables declared at the top of your program outside any function). The various functions listed below will cause changes to the contents of that list and `DrawingPanel`.

The following sections describe in detail functions you must implement in your `tile_manager` file. After any click or press, re-draw all of the tiles in your list.

```
def add(event)
```
Called when the user presses n. Adds a tile to the top of the list.

This tile should have a random size between 25 and 60 pixels wide and tall and be located at a random location but fully visible. This means the random position should be such that the rectangle's top-left x/y position is non-negative and also such that every pixel of the tile is within the width and height provided of the `DrawingPanel`. For example, if the width of the `DrawingPanel` is 300 and the height is 200, a tile of size 20x20 must be placed at a random position such that its top-left x/y position is between (0, 0) and (280, 180).

It should be a random color. You can generate a random color by creating a tuple of three random numbers between 0 and 255. This tuple can be passed directly to `fill_rect` where you would normally pass a color name.

```
def add_all()
```
Called when the program starts. Adds 50 tiles (as described in `add`) to the list of tiles.

```
def draw_all()
```
Called when the program starts. Draws all of the tiles in the list on the `DrawingPanel`. Each tile should have a solid background color and a black outline around it. Tiles should be drawn from the bottom (start) to the top (end) of your list.

```
def raises(event)
```
Called when the user left-clicks. The event parameter contains the coordinates where the user clicked. If these coordinates touch any tiles, you should move the topmost of these tiles to the very top (end) of the list.

```
def lower(event)
```
Called when the user right-clicks. The event parameter contains the coordinates where the user clicked. If these coordinates touch any tiles, you should move the topmost of these tiles to the very bottom (beginning) of the list.

```
def delete(event)
```
Called when the user Shift-left-clicks. The event parameter contains the coordinates where the user clicked. If these coordinates touch any tiles, you should delete the topmost of these tiles from the list.

```
def delete_all(event)
```
Called when the user Shift-right-clicks. The event parameter contains the coordinates where the user clicked. If these coordinates touch any tiles, you should delete *all* such tiles from the list.

```
def shuffle(event)
```
Called when the user types s. This function should move every tile on the screen to a new random x/y pixel position. These positions should follow the same rules as described in `add`.

You will notice that several of the functions above take an `event` parameter. This parameter represents the mouse click that happened that prompted your function to be called. You can access the y location where the user clicked by accessing `event.y` and the x location by accessing `event.x`.

Development Strategy and Hints:

One of the most important techniques for professional developers is to write code in stages ("**iterative enhancement**" or "**stepwise refinement**") rather than trying to do it all at once. This includes testing for correctness at each stage before moving to the next one. The next few paragraphs contain a detailed development plan. Study it carefully and think about why we suggest the plan we do.

Start by writing empty "**stub**" versions of all required functions so that `tiles` will run without errors. Stub functions mean writing the function header with the Python keyword `pass` as the only contents of the body of the function. This will allow you to create the function without putting anything in it. Remove `pass` when you are ready to write the actual function body. For example:

```
def add_tiles:
    pass
```

We suggest that you write your `add` first, then `draw_all`. You can then run the program to make sure that you can see the tiles appear on the screen. Next write `add_all`. Then, add click-related functions one at a time and test each one individually to be sure it works before moving on to the next.

One part of this program involves figuring out which tile(s), if any, touch a given x/y pixel. You can figure this out by comparing the x/y position of the click to the x/y area covered by the tile. For example, if a tile has a top-left corner of (x=20, y=10), a width of 50, and a height of 15, it touches all of the pixels from (20, 10) through (69, 24) inclusive. Such a tile contains the point (32, 17) because 32 is between 20 and 69 and 17 is between 10 and 24.

If you have bugs or errors in your code, there are several things you can try. We recommend you print out the state of your list with temporary **`print` statements**. You should remove any such `print` statements before you turn in the assignment.

Your `DrawingPanel` size and background color are up to you.

Style Guidelines and Grading:

As always, a major focus of our style grading is **redundancy**. As much as possible, avoid redundancy and repeated logic in your code. One powerful way to avoid redundancy is to create "helper" function(s) to capture repeated code. It is just fine to have additional functions in your `tile_manager` beyond those specified here. For example, you may find that multiple functions in your program do similar things. If so, you should create helper function(s) to capture the common code.

You should not use any other data structures besides the list of tiles. You should use the list and its functions appropriately, and take advantage of its ability to "shift" elements as they are added or removed. Your list should not store any invalid or `None` elements as a result of any mouse click activity.

You should introduce **constants** for any hardcoded values that appear in the code.

You should follow good general Python style guidelines such as: appropriately using control structures like loops and `if`/`else` statements; avoiding redundancy using techniques such as functions, loops, and factoring common code out of `if`/`else` statements; good variable and function names; and not having any lines of code longer than 80 characters in length.

You should **comment** your code with a heading at the top of your file with your name, section, and a description of the overall program. Also place a comment heading on top of each function, and a comment on any complex sections of your code. Comment headings should use descriptive complete sentences and should be written *in your own words*, explaining each function's behavior, parameters, return values, and assumptions made by your code, as appropriate.

Your solution should use only material taught in class.

Part 2:

This part of the assignment will give you an opportunity to demonstrate and improve your testing and debugging skills. Turn in two files `debug_test_process.txt` and `not_buggy.py`. To complete this part you will need `buggy.py` and `voting.txt`, provided on the course web site.

The file `buggy.py` contains some very buggy code. It is your job to debug, test and fix this code's style. We will grade this process by looking at the corrected version of the code which you will submit in `not_buggy.py` and reading a description of your process which you will submit in `debug_test_process.txt`.

To document your process write a list of steps you took to debug and results you got from these steps. Here are some examples of valid steps:

- run the program with added prints to show you the value of a particular variable or variables
- add print statements in areas of code you are not sure if your run is entering (a print in an `if` statement, for example)
- alter inputs to the code and comparing the resulting output with the expected output
- run the code and interpret the error message it outputs

It is not enough to just spot the bug by looking at the code. You must get the code to display the bug in a run, document what you did to find it and then what you did to fix it. You can see an example of what we are looking for posted on the lecture calendar for 11/1.

The provided buggy code reads from a file in the following format:

```
Aberdeenshire
yyynnnnynynyannnynynanynaanyna
Midlothian
nnnnynyynyanyaanynyanynnnanyna
Berwickshire
nnnnnnnnnnnnnnnnnnnnynnnnnynnnnny
```

It contains pairs of lines, the first containing a shire name and the second containing a series of `a`, `y` and `n` characters. The `y` characters represent a yes vote, the `n` a no vote and the `a` an abstain vote in the 2014 referendum for Scottish independence.

The program should read through this file and compute the percentage of yes votes for each shire and the percentage of yes votes overall.

Below is what the program should output when run on example input file `votes.txt`:

```
Aberdeenshire: 33.3
Midlothian: 30.0
Berwickshire: 9.1
Dunbartonshire: 63.4
Fife: 35.9
Lanarkshire: 75.0
East Lothian: 37.5
Ayrshire: 81.2

Overall there were 43.2 yes votes
```

Note that the program should be able to run correctly on a different input file so long as that file is in the same general format as described in this document.