Specification for Assignment 6 of 8

Your submission ***must include a completed copy of the assignment header*** *that* ***must include your name*** *(as it appears on cuLearn),* ***student number****, and* ***section****.*

Your submission ***must be a source file named 'comp1405_f17_#########_a6.py'*** *(with the number signs replaced by your nine digit student number),* ***must be written*** *using the* ***Python 3*** *language, and* ***must run on the official virtual machine****.*

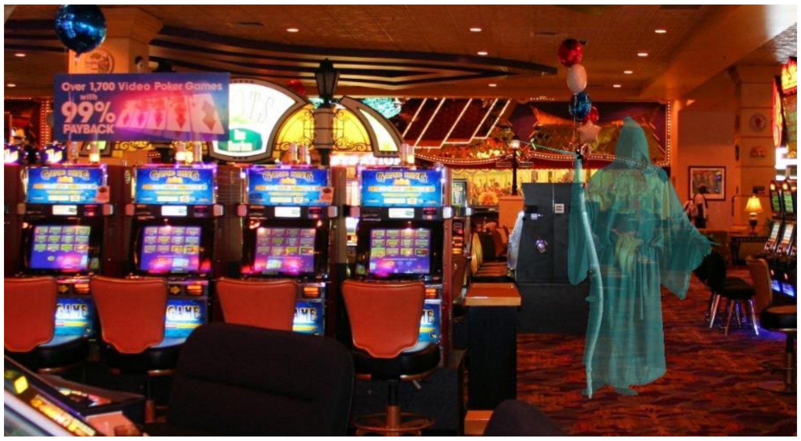
Do not compress your submission *into a "zip" file.*

Late assignments will not be accepted *and will* ***receive a mark of 0****.*

Submissions that crash *(i.e., terminate with an error) will* ***receive a mark of 0****.*

The due date for this assignment is Thursday, November 2, 2017, by 11:30pm.

For this assignment, you will create a program that uses only pygame to perform a simple chroma-keying (also known as green screening) operation, to merge two images. The first image will be a background image and the second image will be the image of a ghost against a green background (i.e., pixels where the red and blue values are 0 and the green value is 255). Your task will be to overlay a semi-transparent image of the ghost on top of the background, centered at a point defined by the user. The result will be an image where the monster appears as a ghost, as depicted in the example below.



You must create a flowchart before you start writing your code for this assignment. You will not be submitting the flowchart on cuLearn, but if you visit the instructor or a teaching assistant **you will be asked to show your flowchart before you will receive any assistance**.

Specification for Assignment 6 of 8

This is actually a very simple problem, but only if you take the time to **apply a divide-and-conquer methodology**. It can be easily completed with about 30 lines of code, but you will need to combine many of the different programming skills you have learned.

You must **use meaningful variable names** and you must **comment your code**.

Your program must first **ask the user if they require instructions**, and if they say yes you must print some instructions to the terminal. This requires a branching control structure.

You will then need to **get the filenames for the ghost image and the background image**. If you wish, you may use command-line arguments to do this. If you do decide to use command-line arguments then you would use the number of command-line arguments to determine if the user needs instructions - if the user does not provide the two required arguments then you would provide instructions and terminate.

Once you have the filenames, **display the background image**.

After you have displayed the background image, **ask the user for the x and y co-ordinates at which you must center the ghost.** You can ask for these values through the terminal.

The x and y co-ordinates you receive from the user must not be outside the background image (i.e., the x value, for instance, cannot be less than 0 or greater than the width of the background image). **If the user gives you invalid x or y co-ordinates, you must loop back** and ask the user to enter the co-ordinate(s) again.

You must **use nested loops to check each of the pixels** from the ghost image and see if that pixel is green or not. **If the pixel is a green pixel, then you will not copy that pixel** from the ghost image onto the background.

If your nested loop does encounter a pixel and it isn't green, then **you must average the red, green, and blue values of the non-green pixel from the ghost with the corresponding pixel from the background.** This will achieve the semi-transparency effect.

Please note that **the co-ordinates of a pixel from the ghost image will not be the same as the co-ordinates of the corresponding pixel in the background**. You will need the co-ordinates provided by the user to center the image and achieve the semi-transparency effect.

Your solution **must use nested loops to process each pixel** in the ghost image to achieve the semi-transparency effect and do the copying onto the background. **You will be severely penalized if you use any of the pygame colorkey or alpha functions**.

You must **update the display once you are finished** processing all the pixels (using a call to the `pygame.display.update` function) and you must **leave the window open** for the user to review the result. The code for leaving the window open is provided on the next page.

The **functions you require** for completing this assignment are listed and described on the following pages.

The `get_rect` function gets the dimensions of an image. It can be called on an image and will return an ordered pair that contains the width and height of that image. If a is the name of a variable holding an image surface, then the following statement will get the dimensions of that image and store them in the variables b and c (for width and height, respectively).

```
(b, c) = a.get_rect().size
```

The `pygame.display.set_mode` function creates a window. It takes an ordered pair of integers as an argument and returns a window surface with those dimensions. If a and b are integers for the desired width and height, the following statement will create the window and return it for storage in variable c.

```
c = pygame.display.set_mode( (a, b) )
```

The `get_at` function gets the colour of a pixel. Called on an image, it takes an ordered pair of integers as an argument and returns an ordered quadruple (i.e., four values) that contains the colour of the pixel in the image at those co-ordinates. If a is an image and b and c are the co-ordinates of a pixel in that image, then d, e, and f will be assigned the red, green, and blue components of the colour of that pixel. Note that you may not use the fourth return value (i.e., the underscore below) to attempt to receive an alpha value for transparency.

```
(d, e, f, _) = a.get_at( (b, c) )
```

The `set_at` function sets the colour of a pixel. Called on an image (or a surface like one would get as the return value from a `set_mode` function call), it takes two arguments and has no return value. The arguments are an ordered pair for the co-ordinates of the pixel to change and an ordered triple (i.e., three values) for the red, green, and blue components of the desired pixel colour. If a is an image, b and c are the co-ordinates of a pixel in that image, and d, e, and f are the red, green, and blue components of the desired colour, then the following statement will change that pixel to the desired colour.

```
a.set_at( (b, c), (d, e, f) )
```

The `blit` function copys one image onto another. Called on an image (or a surface like one would get as the return value from a `set_mode` function call), it takes two arguments and has no return value. The arguments are the image to be copied and an ordered pair for the co-ordinates for where to start copying the image (i.e., the top-left corner). If a and b are the background and foreground images (respectively) and c and d are the co-ordinates of a pixel, image b will be copied into image a such that b's top-left corner will be at (c, d)

```
a.blit( b, (c, d) )
```

The `pygame.image.load` function loads an image. It takes a string argument and returns the loaded image as a surface. If `a` is the name of an image file then the following statement will load that image and store the surface in the variable `b`:

```
b = pygame.image.load(a)
```

Please also note that adding the following block of code at the end of your program will **force the pygame window to stay open until the user tries to close the window** by clicking on button in the top-right corner.

```
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
    pygame.display.update()
```

Your program is expected to work with any pair of bitmaps (i.e., ".bmp" images) provided by the user. That said, you are not limited to the background images provided on cuLearn. If you are particularly proud of an image you have created from with your program and you wish to share it with the class, please email it to the instructor. If the instructor receives enough images that can be shared with the class, then they will be added to a short slideshow for the beginning of the subsequent lecture.

Finally, there is an additional feature you can consider adding **if you wish to receive bonus marks** on your assignment. Since this is a bonus feature, you will not receive any assistance from the instructor or the teaching assistant for accomplishing this - you are expected to complete this task individually if you wish to receive the bonus.

The bonus feature would require that you allow the user to use the mouse (instead of the terminal) to select the center point at which the ghost should be copied. You can read about the functions contained in the pygame module for interacting with the mouse at https://www.pygame.org/docs/ref/mouse.html. Once you have displayed the background image, you must repeatedly get the co-ordinates of the pixel where the mouse is currently located. You must then render a pygame font object containing those co-ordinates and "blit" it at the corresponding location on the background image. You can read about pygame font objects at https://www.pygame.org/docs/ref/font.html. To do this will require a looping control structure and you will be expected to redraw the image whenever the mouse is moved (such that only one set of co-ordinates ever appears on top of the background). Once the user clicks the mouse, you must redraw the background and proceed with the rest of your program.