Project 2

Given on Tuesday, October 14, 2014
Phase 1 demo by Friday, October 17, 2014 at 5:00 PM
Phase 2 demo by Friday, October 24, 2014 at 5:00 PM
Final version due via upload program on Monday, Nov. 3, at 11:00 PM

The Problem

For this project, you will implement the game of Master Mind as described in `http://en.wikipedia.org/wiki/Mastermind_%28board_game%29`, using the graphics package. Your implementation is concerned with playing one game only.

The Initial Set Up

First, we will have to create a few graphics objects that enable us to play the game (see the figure for phase 1 and the video on the course's website). For example, we will have to draw the palette (six distinct colors) and the place holders (4 circles with white background) for the code-breaker to use to build a guess pattern. In addition, we need a circle that can be used to exit the game regardless of the state of the game. Finally, during the development of the game, it is helpful to see the secret code (4 circles in randomly selected colors) so that we can verify the feedback algorithm that needs to be developed. In the completed game, the feedback circles (key pegs in the description of the game) will be printed in white and red after a code-breaker has entered a correct guess. The last item is a *state* circle — a circle that gets displayed on the top right of the game board. Initially the background of this circle is white. We will discuss this circle later in this write up.

The following code-segment includes the definition of a number of variables that you should use to create the above geometric objects.

```
g_window_width = 360
g_window_height = 700

leftmost_x = 50
palette_y = g_window_height - 50

circle_radius = 20
circle_diameter = 2 * circle_radius
h_circle_spacing = 5
v_circle_spacing = 10

exit_x = 10 + circle_radius
exit_y = 30

state_x = g_window_width - circle_radius
state_y = exit_y
circle_width = 1

secret_code_x = exit_x
```

```
secret_code_y = exit_y

guess_y = palette_y - 100
guess_x = leftmost_x + circle_radius + h_circle_spacing

feedback_circle_radius = 5
feedback_x = leftmost_x + 5 * circle_radius * 2 + 4 * h_circle_spacing + feedback_circle_radius
feedback_y = guess_y + circle_radius - feedback_circle_radius
feedback_h_spacing = 20
feedback_v_spacing = 20

secret_code_x = guess_x
secret_code_y = 150

palette_colors = ['green',  'orange', 'darkblue', 'yellow', 'darkred', 'lightblue']
num_palette_colors = len(palette_colors)
```

Details

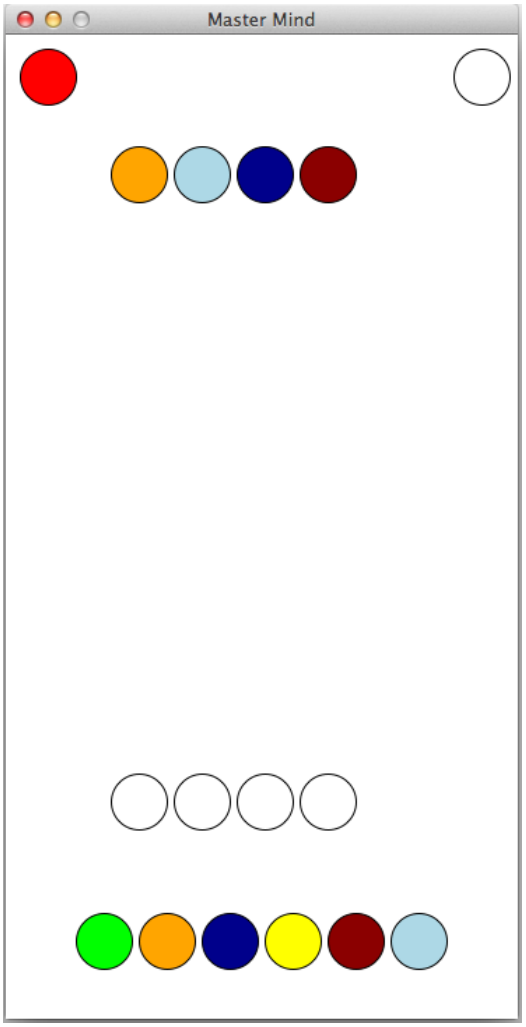
The code breaker assembles a guess by clicking on a palette circle as though choosing it and then clicking on one of the *guess* circles. The guess circle so chosen will be colored with the selected palette color. Once all four guess circles have been assigned colors through this process, the program compares the guess colors with that of the secret code and provides feedback with zero to four *key pegs* — small circles printed on the right side of the guess that was just completed. Zero pegs indicates that none of the colors that the code-breaker guessed matched any of the secret code colors. A red-peg indicates that one of the guess colors matched one of the secret code pegs in color and position. A white-peg indicates the existence of a correct color match placed in the wrong position. The game is won when the code-breaker earns four red feedback pegs. It is lost when the code-breaker makes eight (8) wrong guesses.

Input

Input is provided strictly through the mouse clicks. The color of the *state circle*, the circle on the top-right of the board, is either white or it is one of the palette colors. When the color of the state circle is white, it means that the code-breaker has not selected a color from the palette yet. A palette color is selected when the code-breaker clicks the mouse in one of the palette circles. As soon as that happens, the background color of the state circle changes to the color of the palette circle that received the mouse click. At this point, if the code breaker clicks the mouse in one of the guess circles whose background is white, the background color of the guess circles changes to that of the state circle's color and the state circle changes to white. On the other hand, if while the state circle has a palette color the code-break clicks anywhere on the board outside of the guess circles that are white, the state circle loses its selection and its background changes to white. This is equivalent to having undone a color selection.

Phase 1 — 20 points

For phase 1, your program should display the configuration board depicted below.



Use of functions is critical to the successful implementation of this project. Since this phase builds the ground work for the future phases, you will have learn and put into practice certain design principles. Let's elaborate on some of the concepts and design decisions.

1. Each of the secret-code, guess-circles, and color palette-circles is a list of circles. Therefore, you should write a function to build and *return* each of these lists.

2. You should have a function that, given a list of circles, draws the circles in the window. Therefore, for each of these lists, you will have a function that creates and returns it, but doesn't necessarily draws it. Of course, you only draw the guess-circles as you develop your code.

3. After each of the *exit* and *status* circles is created and drawn, it needs to be made available to the rest of the program for future reference. Specifically, every time that you accept a mouse-click, you will have to decide whether the mouse-click was in the exit circle or not so that you can exit the program if it was. Likewise, when a color is selected from the

color palette or when a guess circle is assigned a color, the color of the status circle needs to be changed. Therefore, you will have to create these structures in the main function in the form of assignment statements. For example:

```
def main():
    # some code here

    color_palette = create_color_palette(left_circle_center_x, left_circle_center_y)
    draw_circles(color_palette)
    exit_circle = create_exit_circle(center_x, center_y)
    exit_circle.draw(window)

    # perhpas more code here...

    mouse_click = window.getMouse()
    if is_click_in_circle(mouse_click, exit_circle):
        window.close()
        return         # return from "main" will terminate the program.
    elif # the rest of the condition here.

    # the rest of the code here...
```

With that introduction, let's look at the functions that you need to develop for this phase of the project.

1. A function called `create_circle()`

```
def create_circle(center_x, center_y, radius)
# Using the (center_x, center_y) and radius, creates and returns a circle.
# Return value: The circle that this function creates.
```

2. A function called `create_exit_circle()`

```
def create_exit_circle(center_x, center_y, radius)
# Using the (center_x, center_y) and radius, creates a circle,
# set its fill-color to red, and returns it.
```

3. A function called `create_state_circle`

```
def create_state_circle(center_x, center_y, radius)
# Using the (center_x, center_y) and radius, creates a circle, uses
# white to fill it, and returns it.
```

4. A function called `create_color_palette`

```
def create_color_palette(left_circle_center_x, left_circle_center_y, radius, palette_colors)
# "palette_colors" is a list of color-names. This function creates six
# circles (call it "palette_circles") using the colors in
```

```
# "palette_colors" and returns it. The color of the circle at
# "palette_circles[0]" will be "palette_colors[0]" and its center will
# be at (left_circle_center_x, left_circle_center_y).  The center of
# the second circle will be at
# (left_circle_center_x + circle_diameter + h_circle_spacing, left_circle_center_y)
# and its color would be "palette_colors[1]". Note that
# h_circle_spacing is a global variable.
```

5. A function called `create_skel_circles`

```
def create_skel_circle(left_circle_center_x, left_circle_center_y, radius)
# This function creates a list of four circles in white color and returns it.
# The center of the first circle on this list is at (left_circle_center_x, left_circle_center_
```

6. A function called `create_secret_code_colors`

```
def create_secret_code_colors(palette_colors)
# The function uses "palette_colors" to randomly select four
# colors to be used as the secret code. The colors are stored in a list
# which the function returns.
```

7. A function called `create_secret_code_circles`

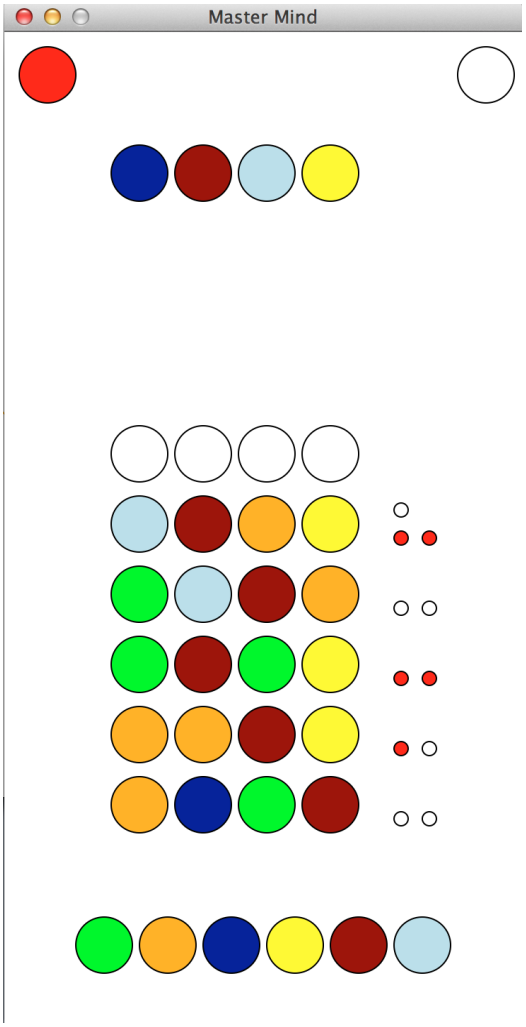
```
def create_secret_code_circles(left_circle_center_x, left_circle_center_y, radius, secret_code
# The function creates a list of four circles using the colors in
# secret_code_colors. The color of the first circle is "secret_code_color[0]"
# and its center is at (left_circle_center_x, left_circle_center_y).
```

8. A function called `main`

```
def main():
# Call the functions defined above to create the configuration shown in the first figure.
```

Phase 2 — 50 points

For this phase of the project, your solution should allow the player to compose as many as eight guesses. In addition, if the player were to click on the *exit circle* — the circle at the top-right of play area — at any time, the program should exit. The *state circle* should behave as described in the *Input* section of this write up. Please refer to the project page for this course to see a demonstration of how your solution is expected to work.



Here is a list of new functions that you need to implement for this phase.

1. A function called `create_and_display_final_text`. You will use this function to create a text-box that reads "You Win!" when the game is over and the player wins, or for this phase, to display "Done!". Notice that you will display "Done!" after the player has completed eight (8) guesses.

```
def create_and_display_final_text(final_text_x, final_text_y, text_to_use):
    """
    Creates a Text-box, sets its text to ‘‘text_to_use’’, its anchor-point
    at (final_text_x, final_text_y), and displays it in the window.

    text_to_use: the text for the Text-box to be created
    win: the graphics window into which the text should be displayed
    return value: None
    """
```

2. A function called `find_clicked_circle`. You will use this function when the player clicks in one of the palette circles or in one of the guess circles. If one of the circles was clicked, the function returns its index not the circle.

```
def find_clicked_circle(click_point, list_of_circles):
    """
    Returns the index of the circle in "list_of_circles" that includes
    point, "click_point". If no circle in "list_of_circles" include "click_point",
    the function returns None.


    click_point: an instace of "Point"
    list_of_circles: a list of instances of "Circle"
    return value: the index of circle in "list_of_circles" that
    includes "click_point" or None.
    """
```

3. A function called `is_click_in_circle`. This function can be called by `find_clicked_circle` or directly, for example, when you want to find out if the player has clicked the red circle that stops the program.

```
def is_click_in_circle(click_point, circle):
    """
    Returns True if point "click_point" is in circle "circle". Otherwise,
    it returns False.

    click_point: An instance of "Point"
    circle: An instance of "Circle"
    return value: returns True if "click_point" is anywhere in "circle".
                  Otherwise, it returns False.
    """
```

4. A function called `move_circles_up`.

   From phase 1, you probably already have a list of *guess* circles for the player to compose a guess. The player chooses a palette color first and then clicks on one of these *guess* circles. You can use `find_clicked_circle` to get the index of that circle like this.

```
idx =  find_clicked_circle(click_point, guess_circles)
if idx is not None:
    clicked_circle = guess_circles[idx]
    new_circle = clicked_circle.clone()   # make a copy of it.
    new_circle.setFill( the paletter color that the player selected )
    new_circle.draw()
    ...
```

   So, by cloning the circle that was clicked and drawing it, you will cover over the guess circle that was clicked; the guess circle is still where it was. Following this method, after the player colors all four guess circles, you already have four circles in the window with

the four guess circles hidden under them. Now, you use the following function to move the guess circles up so that the player can compose another guess. That is, you draw the guess circles once and from then on, when a guess is complete, you just move the guess circles up for the next round.

```
def move_circles_up( circles, dy ):
    """
    It moves each object in "circles" up by "dy" pixels.

    circles: a list of circles that are already drawn in the graphics window.
    dy: a positive integer. Each circle in "circles" will be moved UP by
        this number of pixels.
    """
```

Complete Project — 90 points

For this phase of the project, you are to implement a working version of the game. If the user guesses the secret-code in eight (8) guesses, (s)he wins, otherwise, the game is lost.

Here is a list of functions that would be helpful for this phase.

1. A function called `guess_is_right` to determine whether the four colors that the code-breaker has selected (the guess) and the secret code match. If this function returns true, then the code-breaker has succeeded guessing the secret-code and thus has won the game.

```
def guess_is_right(secret_code_colors, guess_colors):
    """
    Returns True if the corresponding elements of secret_code_colors
    and guess_colors match.

    secret_code_colors:  a list of colors
    guess_colors: a list of colors where
                  len(secret_code_colors) == len(guess_colors) is true
    return value: True if the corresponding element of the two
                  lists are equal. Otherwise, the function returns False
    """
```

2. A function called `create_feedback_skel` to create four smaller circle to be used as feedback circle after the code-breaker has entered a guess.

```
def create_feedback_skel():
    """
    Creates four smaller circles such that when drawn, they form
    two rows each of which includes 2 circles.

    return value: a list of four circles
    """
```

3. The implementation for following function has been posted on the *projects* tab of the web-pages for this class.

```
def create_feedback_circles(secret_code_colors, guess_code_colors, feedback_skel_circles):
    """
    Compares secret_code_colors and guess_code_colors and based
    on the outcome, it creates zero to four circles.

    secret_code_colors: a list of 4 colors that represent the secret code
    guess_code_colors: a list of 4 colors that represent the code-breaker's guess
    feedback_skel_circles:
         a list of 4 circles to be used as a model to create feedback
         circles. For example, if the guess colors were to give rise to two
         feedback circles, this function clones the first two circles in
         feedback_skel_circles, "color" them appropriately, and returns
         them. The original feedback_skel_circles will not be modified in any
         way by this function.  return value: a list of as few as 0 and as
         many as 4 circles.
    """
```

How to Submit the Final Version

You should submit the final version of this project via the upload program posted on the projects page of the web-site for this course. Before submitting it, be sure that you have the right version and that your name is recorded at the top of the file that you are about to submit.

Grading

The most important part of your grade is the correctness of your final program. Your program will be tested numerous times, using different inputs, to be sure that it meets the specifications. A correct program will receive 85% of the grade for this project. The remaining 15% is for programming style and design. This includes good design, use of spaces between operators and operands, comments, and the use of descriptive variable-names.

Collaboration policy

Programming projects must be your own work, and academic misconduct is taken very seriously. You may discuss ideas and approaches with other students and the course staff, but you should work out all details and write up all solutions on your own. **The following actions will be penalized as academic dishonesty:**

- Copying part or all of another student's assignment.
- Copying old or published solutions.
- Looking at another student's code or discussing it in great detail. You will be penalized if your program matches another student's program too closely.
- Showing your code or describing your code in great detail to anyone other than the course staff or tutor.