fullName:_____ andrewID:_____ section:___

15-112 S25
Quiz8 version B

Read these instructions carefully before starting:

1. Quiz versions are color-coded. You must have a different version (color) of this quiz than the students sitting to your left and right.
2. Stop writing and submit the entire quiz when instructed by the proctor.
    - Do not unstaple any pages.
    - You must submit the entire quiz with all pages intact.
3. Do not discuss the quiz with anyone else until after 5pm.
    - This applies to everyone, including students in either lecture.
4. Do not use your own scrap paper.
    - You should not need scrap paper, there is plenty of room for you on the quiz.
    - However, if you absolutely must use scrap paper, raise your hand and we will provide some. Then, you must write your andrew id clearly on the scrap paper, and hand in the scrap paper with your paper quiz. We will not grade anything on your scrap paper.
5. You may not ask questions during the quiz.
    - The one exception is for English-language clarifications.
    - If you are unsure how to interpret a problem, just take your best guess.
6. Do not use any concepts (including built-in functions) not covered in the notes through week 8 or beyond unit 5.
    - Do not use dictionaries, sets, or recursion.
7. Do not hardcode your solutions.
    - We may test your code using additional test cases.
    - Hardcoding will receive zero points.
8. Assume almostEqual(x, y) and rounded(n) are both supplied for you.
    - You must write all other helper functions you wish to use, unless we specify otherwise.
9. Good luck!

Fill In the Blank (FitB) [32 pts, 8 pts each]
Fill in the blanks so that the code below will behave as specified. You may not modify any code that is already written, nor may you add code before or after the blank.

If you must replace your answer, please erase your first answer and write a new one, or cross out your answer completely and draw an arrow that *very clearly* indicates your new answer that fills in the blank. Regardless, your answer may not span multiple lines.

FitB #1:

```
# This should draw a polygon using the points in a 1D list L
# of unknown length. For example, but not necessarily,
# L might be [100, 100, 50, 200, 300, 300, 250, 50]:


drawPolygon(_____________________________________________,
                                fill='cyan', border='black')
```

FitB #2:

```
# This should increase app.counter by 1 whenever the 'x' key
# is held down:

def onKeyHold(app, keys):


    if _____________________________________________:
        app.counter += 1
```

FitB #3:

```
# This should set app.color to 'blue' whenever any key is
# released:


def ____________________________________________________:
    app.color = 'blue'
```

FitB #4:

```
# Fill in the blank from the snake example:

def takeStep(app):
    # Find the new position of the snake's head:
    headRow, headCol = app.snake[0]
    drow, dcol = app.direction
    newHeadRow = headRow + drow
    newHeadCol = headCol + dcol
    # And move the snake there if we we can:
    if ((newHeadRow < 0) or (newHeadRow >= app.rows) or
        (newHeadCol < 0) or (newHeadCol >= app.cols)):
        # ran off the board
        app.gameOver = True


    elif ____________________________________________________:
        # ran into itself
        app.gameOver = True
    else:
        ... # Assume the code works correctly beyond here
```

Multiple Choice (MC) [15 pts, 5 pts each]
Select the *best* answer by completely filling in the the correct bubble below. Select only one. If you believe multiple correct answers are technically correct, choose the most generally correct one.

MC #1:
In the note on drawing a 2d board, why did we need the function `drawBoardBorder(app)`?

◯ A) Without this function, the border around the board was half as thick as the interior borders.

◯ B) Without this function, the border around the board was not drawn at all.

◯ C) Without this function, there were no borders around any cells in the board.

◯ D) We did not need or use this function.

◯ E) None of the above.

MC #2:
In the note on cell selection, how and why did we need the function `getCell(app, x, y)`?

◯ A) This function is called by `onKeyPress` to determine which cell to select.

◯ B) This function is called by `onStep` to determine which cell to select.

◯ C) This function is called by `onMousePress` to determine which cell to select.

◯ D) This function is called by `redrawAll` to determine which cell to draw as selected.

◯ E) None of the above.

MC #3:
In `getCell(app, x, y)`, why did we use `math.floor(dy / cellHeight)` instead of `(dy // cellHeight)`?

◯ A) We need the floor, but (dy // cellHeight) rounds.

◯ B) If dy is a non-number, math.floor(dy / cellHeight) will not crash but (dy // cellHeight) will crash.

◯ C) If dy is a float then (dy // cellHeight) will be a float.

◯ D) We did not need or use math.floor here.

◯ E) None of the above.

Code Tracing (CT) [20 pts, 10 pts each]
For each CT, indicate what the code prints.
Place your answer (and nothing else) in the box below the code.

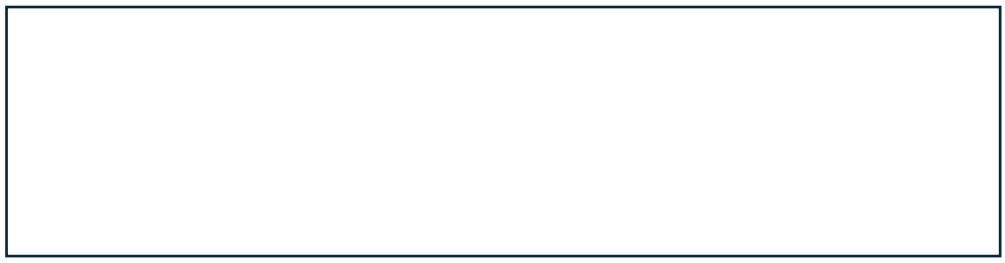
Note: If floats occur in these CTs, they will have no more than one digit after the decimal point.

CT1:

```
# getRadiusEndpoint is taken verbatim from the notes:
# (verbatim means we have not made any changes to the code)

import math
def getRadiusEndpoint(cx, cy, r, theta):
    return (cx + r*math.cos(math.radians(theta)),
            cy - r*math.sin(math.radians(theta)))

def ct1():
    L = getRadiusEndpoint(60, 70, 80, 90)
    return [round(v) for v in L]

print(ct1())
```

CT2:

```
# distance is taken verbatim from the notes:

import math
def distance(x0, y0, x1, y1):
    return ((x1 - x0)**2 + (y1 - y0)**2)**0.5

# getRadiusAndAngleToEndpoint is verbatim from the notes:

def getRadiusAndAngleToEndpoint(cx, cy, targetX, targetY):
    radius = distance(cx, cy, targetX, targetY)
    angle = math.degrees(math.atan2(
                         cy-targetY,targetX-cx)) % 360
    return (radius, angle)

def ct2():
    L = getRadiusAndAngleToEndpoint(60, 70, 40, 70)
    return [round(v) for v in L]

print(ct2())
```



Free Response (FR1): Growing Rects Animation [33 pts]

Note: For full credit, **you must represent each rectangle as an instance of SimpleNamespace.**

With that, write an animation where:

- At first, the canvas is empty.
- The canvas will eventually contain rectangles where:
  - Rectangles have no fill.
  - Rectangles have a border that is black, except that one rectangle may be selected, and if so, that rectangle has a border that is red.
  - Each rectangle contains a counter that is drawn centered in the rectangle.
- Each time the user presses the mouse:
  - If the mouse press is not inside a rectangle, then a new 20x20 rectangle is drawn centered on the mouse press. The new rectangle's counter starts at 1, and the new rectangle is selected, so its border is red.
  - If the mouse press is inside a rectangle, then that rectangle becomes selected and its counter increases by 1 (and any rectangle that was selected is de-selected). The other counters remain unchanged.
  - If the user presses the mouse over overlapping rectangles, then the most-recently-created rectangle is the one that is selected, and only its counter increases.
- In onKeyHold, if there is a selected rectangle, then:
  - When the user holds down the 'w' key, the selected rectangle's width increases by 2 while the rectangle's center is unchanged.
  - When the user holds down the 'h' key, the selected rectangle's height increases by 2 while the rectangle's center is unchanged.
  - However, when the user holds down BOTH the 'w' and the 'h' key at the same time, then nothing happens (the width and height remain unchanged).

Notes:

- **You must follow MVC rules.** MVC violations carry large deductions.
- You do not need to set any unspecified details (such as font size of the counter or the border width)

```
#Begin your answer to FR1 here:

from cmu_graphics import *
from types import SimpleNamespace

def onAppStart(app):




def onMousePress(app, mouseX, mouseY):
```

```
#Continue your answer to FR1 here:

def onKeyHold(app, keys):
```

```
#Continue your answer to FR1 here:

def redrawAll(app):









runApp()
```

Bonus Code Tracing (BonusCT) [Optional, 2 pts]
Bonus problems are not required.
This is an animation CT. Draw a picture of what this app will draw when it runs. Draw your answer (and nothing else) in the box below the code.

BonusCT1:

```
#Assume the canvas is 400x400

import math
def redrawAll(app):
    for i in range(2):
        d = 100*i
        e = -60*i
        f(150 + d, 100, 40, 25)
        f(200, 200 + e, 80 + d, 10 + d)

def f(a, b, c, d):
    for t in range(360):
        drawCircle(a + c * math.cos(math.radians(t)),
                   b + d * math.sin(math.radians(t)),
                   3)

runApp()
```