

Name: \_\_\_\_\_ Andrew Id: \_\_\_\_\_

### 15-112 Fall 2024 Quiz 7

Up to ~~25 minutes~~ 30 minutes. No calculators, no notes, no books, no computers. Show your work!

Do not use dictionaries, sets, try/except, or recursion on this quiz.

1. (6 points) **Code Tracing:** Indicate what the following program prints. Place your answer (and nothing else) in the box below the code.

```
def ct(b, a):
    a = b
    b = copy.copy(a)
    c = copy.deepcopy(a)
    a[0] += [6]
    c[1] = b[1]
    a[0] = a[0] + ["Nimr"]
    b[1][2] = "Bye"
    a.append(112)
    print(f"a: {a}")
    print(f"b: {b}")
    print(f"c: {c}")

a = [[1, False, 3], [4, None, "Hi"]]
b = [[10, 20], ["Alpha", "Beta"]]
print(ct(a, b))
print(f"a: {a}")
print(f"b: {b}")
```

## 2. Free Response: Twenty-One

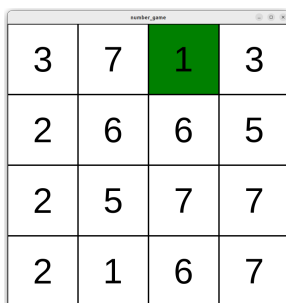
This is a multi-part question wherein you will build a simple, grid-based puzzle game called Twenty-One.

In the game of Twenty-One, the user is presented with a 2D board of randomly chosen numbers in the range 1 - 9. Here is an example:

|   |   |   |   |
|---|---|---|---|
| 3 | 7 | 1 | 3 |
| 2 | 6 | 6 | 5 |
| 2 | 5 | 7 | 7 |
| 2 | 1 | 6 | 7 |

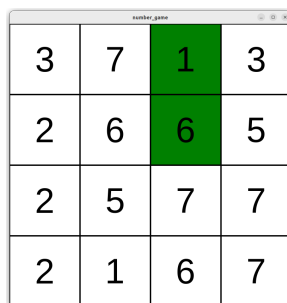
The user's goal is to find a path of numbers which sum to 21. A path is a series of adjacent numbers. Adjacent means that the squares are directly touching, either horizontally, vertically, or diagonally.

In the provided example, there is a valid 21-path starting at the cell with (row, col) coordinates of (0, 2): 1, 6, 7, 7. The user selects the first cell of the path by clicking on it, then clicks on the next cell, etc. Here is the sequence of events:



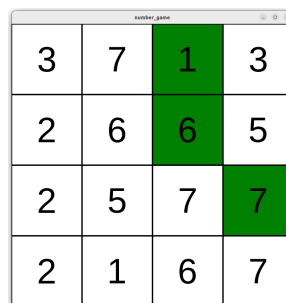
|   |   |   |   |
|---|---|---|---|
| 3 | 7 | 1 | 3 |
| 2 | 6 | 6 | 5 |
| 2 | 5 | 7 | 7 |
| 2 | 1 | 6 | 7 |

(a) After clicking the first cell in the path.



|   |   |   |   |
|---|---|---|---|
| 3 | 7 | 1 | 3 |
| 2 | 6 | 6 | 5 |
| 2 | 5 | 7 | 7 |
| 2 | 1 | 6 | 7 |

(b) After clicking the second cell in the path.



|   |   |   |   |
|---|---|---|---|
| 3 | 7 | 1 | 3 |
| 2 | 6 | 6 | 5 |
| 2 | 5 | 7 | 7 |
| 2 | 1 | 6 | 7 |

(c) After clicking the third cell in the path.



(d) After clicking the fourth cell in the path.

Note that not all valid paths are of length 4. They could have fewer, or more, cells. The important thing is that the path sums to 21.

You will start by writing some helper functions, and then you will write the game.

The questions begin on the next page.

There is more space for the helper functions than you actually require, so don't get worried if you have short answers for (a)-(c).

- (a) (2 points) Write the function `genBoard(numRows, numCols)` which, given a number of rows and number of columns, generates a 2D list with `numRows` rows and `numCols` columns containing random numbers from 1 to 9. For example, calling `genBoard(2,3)` could produce a board like the following:

```
[[3, 2, 8],  
 [8, 1, 2]]
```

- (b) (2 points) Write the function `pathSum(path, board)` which, given a 2D list of integers `board` and a list of tuples of `(row, col)` coordinates `path` on that board, returns the sum of all of the integers at those coordinates. (You may assume that all of the coordinates in `path` are valid for `board`.)

Consider the following testcases:

```
board = [[5, 1, 5, 8],
         [7, 9, 3, 3],
         [8, 6, 4, 9],
         [9, 8, 2, 1]]
path = [(0,0), (1,0), (2,1), (3,1)] # This path goes 5, 7, 6, 8
assert pathSum(path, board) == 26
path = [(1,2), (1,3), (2,2), (3,1)] # This path goes 3, 3, 4, 8
assert pathSum(path, board) == 18
```

(c) (2 points) Write the function `isContigPath(path)` which, given a path, returns `True` if the path is contiguous through the board, and `False` otherwise.

By 'contiguous' we mean that each coordinate in the path is directly adjacent (next to, above/below, or diagonal to) the coordinate before it.

Consider the following testcases:

```
path = [(0, 0), (1, 0), (2, 1), (3, 1)] # This path is contiguous
assert isContigPath(path) == True
path = [(0, 0), (1, 0), (2, 1), (0, 1)] # The 4th coordinate is not adjacent to the 3rd.
                                         # The 4th coordinate is adjacent to the 1st,
                                         # but that doesn't matter

assert isContigPath(path) == False
path = [(0, 3), (3, 3), (3, 0), (0, 0)] # None of these are adjacent to each other
assert isContigPath(path) == False
```

- (d) (8 points) Using the helper functions you wrote in the previous parts of this problem, implement Twenty-One using `cmu_graphics`. Here are some additional criteria:
1. Your board should be 16 cells in a 4x4 grid. When the game starts, the value of each cell should be randomly generated.
  2. When a user clicks a cell, that cell gets added to the current path. If a user tries to add a new cell that isn't adjacent to the most recently added cell, then don't add it to the path.
  3. All cells on the currently chosen path should be green, the other cells should be white.
  4. The game is over when the sum of all cells on the path is 21. (See the screenshots at the beginning of this problem for an example of the game over screen.)
  5. If the user makes a mistake, they can press 'u' to undo the most recent addition to the path. If they keep pressing 'u', eventually there will be no cells in the path.
  6. At any time, the user can press 'r' to reset the game (with a new board) and play again.

Notes/Hints:

- Don't forget to **use your helpers functions**. You may assume that they work properly, even if yours do not.
- You may assume that `runApp()` is already called appropriately, but your game should look good at a variety of widths and heights.
- You will need to write at least `onAppStart`, `redrawAll`, `onMousePress`, and `onKeyPress`.

There is additional answer space on the next page.

