Published

Week 13: Multidimensional Lists, and Recursion | 6 of 7

13 Week 13 Assignment

Computer SCIENCE

You will complete two tasks. Task 1 is much more involved and complex than Task 2.

Objectives

1. Multidimensional Lists
2. Recursion

Task 1 - Memory

Starter Code: **memory_starter.py** (https://usu.instructure.com/courses/681553/files/82901880/download?download_frd=1) **memory_test.py** (https://usu.instructure.com/courses/681553/files/82901879/download?download_frd=1)

You will be creating the memory game. You probably played this as a small child. There are a number of cards face-down, and each player flips two over on each turn. If there is a match, they get a point and go again. If they do not get a match, the cards are flipped back over and the next player goes.

Starter code has been provided. The test.py code will enable you to build and test your GameBoard and MemoryCard classes without having to worry about the actual game play. Once those are working, you can move to the memory_starter.py file. Rename this file task1.py. This file provides a close implementation of the game play, but has bugs in it. You will need to identify and fix the bugs. You will also document what the bugs were in a file called memory_bugs.txt. You will only create Software Designs for the GameBoard and MemoryCard classes. Include UML and designs for each public method, including __init__(). Save these in a file called plan1.txt.

Requirements

- Create a class called MemoryCard in a file called memorycard.py. Place the file in a folder called modules.

- The constructor will have a parameter that is an id number. Each card will have a unique id number
- Define the following public methods:
  - getValue() - returns a value associated with a card. Note, this is not the same as the id as matching cards will have the same value
  - toggleFlipped() - change the card from a face-up to a face-down, and vice versa, state
  - isFlipped() - returns true if the card is currently face-up
  - displayCard() - returns a string. If the card is face-up it returns a symbol associated with the value. If it is face-down, it returns a space. If you don't return a space then your game board printout will be kinda messed up.
    - Symbols are ascii characters starting with "!". Check out **https://www.asciitable.com/ (https://www.asciitable.com/)** for a refresher. Don't forget functions we've learned to encode/decode ascii values.
- You may include any private data and methods that you see fit.
- Hint: I would build this first, then test it with the code in test.py.

- Create a class called GameBoard in a file called gameboard.py. Place the file in a folder called modules.
  - The constructor will have two parameters indicating the number of columns and number of rows in the game board.
  - Define the following public methods:
    - getBoard() - returns a single string representation of the game board. There is an example below, and it can be seen in the demo video. Note that the GameBoard class does not call print(), it just returns a string that can be printed.
    - flipCard(xPos, yPos) - takes an x, y coordinate from the game board and flips the card at that location.
    - isCardFlipped(xPos, yPos) - returns true if the card at that position is face-up, otherwise returns false
    - isMatch(pos1, pos2) - pos1 and pos2 are the x, y locations of two different cards given as a list. For example [5, 3] could be the actual value associated with pos1.
  - The game board should place the cards in a random order. We have done this multiple times in class.
  - You may include any private data and methods that you see fit.
  - Hint: Make sure your class works for different combinations of column and row counts (the board doesn't have to be a square).
- Bugs in memory_starter.py
  - You must identify and fix bugs you find in the file. Save these in a file called memory_bugs.txt. It should look something like this. Note that these may or may not be actual bugs in the program.
    - ```
      Memory Bugs

      1. No prompt for the number of columns
      ```
      ```

```
2. Score for players is not updating properly
3. And so on...
```

- The number of bugs is unknown (to you).
- A bug is anything that makes the gameplay experience not match expectations.
- Expectations are based on the following Requirements Specification
  - The game starts with a welcome message
  - The user is prompted to enter the number of columns on the game board
  - The user is prompted to enter the number of rows on the game board
  - The user is prompted to enter the number of players, which will be named Player 1, Player 2, etc.
  - Game loop
    - Prior to each prompting during game play, the score for each player is printed followed by the game board (see below)
    - The current player is prompted to input an x, y coordinate
    - If the user ever enters the coordinates of a card that is face-up already, they are notified of the error and need to try again
    - If the spot is valid the card is flipped face-up.
    - Process repeats for a second card
    - If a match is detected the player scores a point and goes again
    - If a match is not detected the two cards are flipped face-down and the next player goes
    - This continues until all possible matches are made
  - After all matches are made, one of two possible messages are displayed
    - If it is a tie, then it is so stated along with an indication of the players who tied
    - If there is a single winner then it is so stated along with a message of who the winner is
- You do not need to complete a Software Development plan for this file

Notes

- Example game board. Notice the spacing for each cell. Two cards are currently face-up.

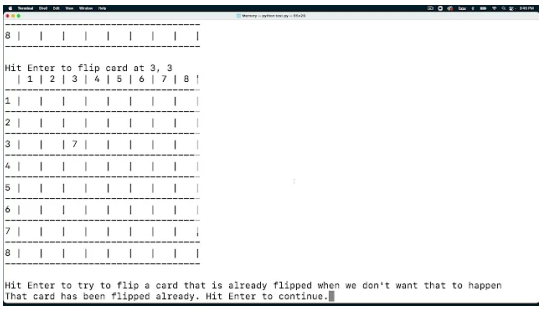
```
Player 1: 0
Player 2: 0
Player 3: 0

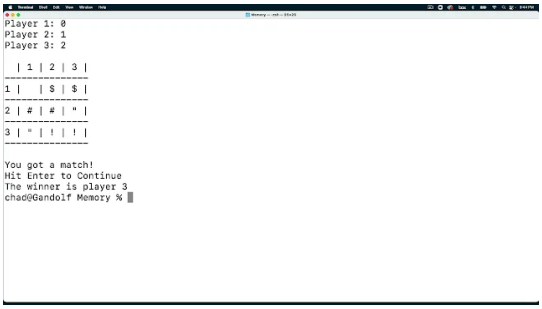
  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
---------------------------------
1 |   |   | ? |   |   |   |   |   |
---------------------------------
2 |   |   |   |   |   |   |   |   |
---------------------------------
3 |   |   |   |   |   | 6 |   |   |
---------------------------------
4 |   |   |   |   |   |   |   |   |
---------------------------------
5 |   |   |   |   |   |   |   |   |
---------------------------------
6 |   |   |   |   |   |   |   |   |
```

```
-----------------------------------
7 |   |   |   |   |   |   |   |   |
-----------------------------------
8 |   |   |   |   |   |   |   |   |
-----------------------------------
```

- Testing this can be a little tricky since the cards are scattered randomly. You can comment out your line of code that randomizes things to make life easier while you test. You can take it as a hint that a single line of code can handle the randomization. Of course, you can do it another way if you'd like.

Demo

- Test
  - 

- Game play
  - 

Task 2 - Recursion Fun

Starter Code: **assn13_task2_starter.py** (https://usu.instructure.com/courses/681553/files/81729294/download?download_frd=1)

Do not modify any existing code in the starter code file. Change the file name to task2.py. Only add function definitions that you need to solve the two problems. You must solve each problem using recursive function calls.

aggienacci series is a number pattern similar to the fibonacci series. Aggienacci values are only defined for positive integers including zero. The following is the definition of aggienacci values:

```
ag(0) = 0
ag(1) = 1
ag(2) = 2
ag(x) = (ag(x-3) + ag(x-2)) / ag(x-1)
```

The binary search portion should be self explanatory. Make sure to handle the count of recursive function calls properly.

A software development plan is not required for this task.

Points	100
Submitting	a file upload
File Types	zip

Due	For	Available from	Until
Dec 10, 2021	Everyone	-	-

Unit 7

Criteria	Ratings		Pts
Task 1: Software Development Plan Software design (no testing necessary) for the public methods in GameBoard and MemoryCard classes. Create UML diagrams for the entire class.	**5 pts Full Marks**	**0 pts No Marks**	5 pts
Task 1: MemoryCard class definition All specified public methods are defined properly.	**10 pts Full Marks**	**0 pts No Marks**	10 pts
Task 1: GameBoard class definition All specified public methods are defined properly.	**15 pts Full Marks**	**0 pts No Marks**	15 pts
Task 1: Score board display The score board and game board are displayed properly	**10 pts Full Marks**	**0 pts No Marks**	10 pts
Task 1: test.py file runs properly	**5 pts Full Marks**	**0 pts No Marks**	5 pts
Task 1: starter code bugs found All starter code bugs are documented in the appropriate file. Note: you do not need to document the solution, just the bug.	**15 pts Full Marks**	**0 pts No Marks**	15 pts
Task 1: Multidimensional list used properly The GameBoard class should use a multidimensional list to track the game board state.	**10 pts Full Marks**	**0 pts No Marks**	10 pts
Task 1: starter code fixed The starter code should be modified to resolve all bugs and match the requirements specification. It is not necessary to document the fixes in another file.	**15 pts Full Marks**	**0 pts No Marks**	15 pts
Task 2: aggienacci implemented correctly Correct output and use of recursive function calls	**5 pts Full Marks**	**0 pts No Marks**	5 pts
Task 2: binary search implemented correctly Correct output and use of recursive function calls. Count of function calls correct	**10 pts Full Marks**	**0 pts No Marks**	10 pts
			Total Points: 100