


## FINAL PROJECT: “THE EVEN COLOURFUL GAME”

	Col 0	Col 1	Col 2	Col 3
Row 0	0	0	2	1
Row 1	4	1	0	2
Row 2	1	0	0	2
Row 3	2	1	0	1



### A. GENERAL PROJECT DESCRIPTION

You are asked to implement a Python program, allowing the user to play the “Even Colourful Game”. This game can be played as a single player game (style “solo”) or two players where the user plays “against the computer” (style “AI”). The program should give the option to the user to play several games, possibly playing each game in a different style. At the end of each game, the program presents to the user some totals and some colorful visuals associated to the game end-results, and invites the user to play more games. At the end of all games some general totals are shown.

For each game, the program will create an initial board based on data from one of three provided files “boardX.csv”, where X is a number, 1 2 or 3. A board will contain only digits (0 to 9) in each cell, organized as a square matrix (of dimension 4x4, 5x5, etc, not larger than 10x10). See Figure 1.

	Col 0	Col 1	Col 2	Col 3
Row 0	0	0	2	1
Row 1	4	1	0	2
Row 2	1	0	0	2
Row 3	2	1	0	1

*Figure 1 – Example of a board 4x4 (corresponding to the provided file board2.csv). A direct printing of a lists of rows with this data would be `[[0,0,2,1],[4,1,0,2],[1,0,0,2],[2,1,0,1]]`*

That is, you are provided with three possible initial board data files in comma separated format. Each boardX.csv file contains information to create one initial board. All boards are square: they have the same number of rows as columns. The user will be able to choose among these various boards for each game (by providing the number 1, 2, or 3)<sup>1</sup>.

The game unfolds as text based dialog with the user. At the end of each game, the program will show some images to the user<sup>2</sup> and save such images as well as jpg files. These images will be based on the game results, and also relying on the provided csv file “colorcoding.csv”. See more details below. **You should also check the sample runs provided, which are considered part of the problem description.**

<sup>1</sup> Note: You may create additional initial board files if that helps you debug your program. The project may be marked with different boards than those provided. **Make sure that you allow that your program at least works with three possible board files named exactly as described.**

<sup>2</sup> You will be asked to import the cmpt120ImageWeek9 module with image manipulation functions. If you encounter problems with the function showImage(...) you may just save the images and not show.

## **B. TOPICS THAT THIS PROJECT INCORPORATES**

With this project you will be working with:

- Reading text (csv) files
- Manipulating one, two and three dimensional lists, (the three dimensional lists have the same structure which we have used to represent images with RGB pixel lists)
- Working with single and nested loops
- Text interaction with the user
- You will use the CMPT120imageWeek9.py module.
- The code is required (and will be most useful for you) to be done in a modular way, defining your own modules and functions, both productive and non-productive.

## **C. DESCRIPTION OF THE GAME IN MORE DETAIL**

**You are highly recommended to implement this project in stages**, first ensuring that the game works well in some basic way/s, before you incorporate additional features. Stage 1 next is a suggestions for you to start implement the game<sup>3</sup>.

### **STAGE 1 [17 points]**

**One game only, 'solo' style game, following the sample runs dialog except that the printing of the board is just directly printing the 2 dimensional list of rows (as in the caption in Figure 1), no images generation, no user validation, no additional features.**

Ask the user the initial information and then show an initial board directly by printing the list of rows. Then have the user choose how many turns they would want to play (the maximum possible turns is the (number of rows -1); for example a maximum of 3 turns in a 4x4 board). The user can change one digit in the board per turn. After the user plays all the turns (i.e. changes at most as many digits as turns), the game is over and the program determines the winner (user or computer).

At the end of the game, if all rows and all columns in the board add up to an even number, then the user wins the game, otherwise the computer wins. If the user wins a game, then the user is added points to their total points. If the user loses the game, these points are subtracted from the user's total points. Points calculated as the integer part of calculating the maximum value among all the numbers in the "final line" and the "final column" divided by the number of turns that the user took. See Figure 2.

The game ends with the program informing who the winner is, the points added (or subtracted). See more examples in the sample runs.

---

<sup>3</sup> You may also want to define some functions and not implement them until later, but at least preview what the functions would do and where they would be called from, initially just including a print in the function definition to show you (as programmer, in the debugging stage) when the function is executed.

	Col 0	Col 1	Col 2	Col 3
Row 0	1	0	4	8
Row 1	4	1	0	2
Row 2	1	0	3	2
Row 3	7	1	0	7

The 'final line' with the sum of all columns is: [13, 2, 7, 19]

The 'final column' with the sum of all rows is: [13, 7, 6, 15]

*Figure 2. A final board, and corresponding final line and final column. This is not a winning board for the user, since some numbers in these lists are not even. The points in this case would be  $19/n$ , where  $n$  is the number of turns the user played.*

### AI STYLE GAME. [7 points]

The AI style game incorporates that “the computer” has the opportunity to change a digit in the board, also one digit per turn, **immediately after the user does a digit change**. The “AI” strategy should only be to randomly choose a position (row, column) in the board, and change the digit in that position to a random digit (potentially leaving the original digit). This may avoid that the user wins (but not necessarily). The same calculation of points applies at the end of the game as when played with ‘solo’ style: adding the points to the user if the user wins, or subtracting the points otherwise.

### NEXT STAGES: ADDITIONAL FEATURES. See below and the sample runs for further details.

You are suggested to incorporate these features gradually.

- Allowing the user to play several games, and including general totals. [4 points]
- Showing the board in a tidier way, with titles as in Figure 2 above and in sample runs, (as opposed to just printing the board as the list of lists of numbers) [3 points]
- Generate and show /save the images associated to the “final line” and “final column” (more details below) [10 points]
- Include user validation of at least number of turns, row and column position, and of one string value, making sure that the value is within the required ranges when appropriate, and ask the user to type again if incorrect. Preview that the user asks 0 turns for one game (not modifying the board). See the sample run 3, with focus on validation. [6 points]
- Add an option (ask the user when asking if ‘solo’ or ‘AI’) that increases the number of turns that can be played per game. For example, in the case of a board of dimension 4x4 allow up to  $(16-1) = 15$  turns (instead of the basic limitation to  $(4-1) = 3$  turns, together with providing the option of the user typing -1 as row number to stop the turns before reaching the maximum [3 points]

### TEXT FILES DESCRIPTION<sup>4</sup>

#### **FILE boardX.csv**

This file will have in the first line, the board number of rows (which is the same as the number of columns), and then one row per line (comma separated) with the digits. The board in Figure 1 above corresponds to the file in Figure 3 next.

	A	B	C	D
1	4			
2	0	0	2	1
3	4	1	0	2
4	1	0	0	2
5	2	1	0	1
6				

Figure 3 –Provided board2.csv file, here shown in excel. The format is comma-separated strings.

#### **FILE: colorcoding.csv.**

There is a line header and then each line contains a number and then 3 RGB values (between 0 and 255, extremes included). Values were generated randomly, so it is possible that different numbers have the same color associated. For example, in Figure 4, line 4 has the color coding for number 2, representing the RGB color [102,225,178]

	A	B	C	D
1	number	R	G	B
2	0	11	55	146
3	1	176	179	45
4	2	102	225	178
5	3	0	172	247
6	4	35	111	246
7	5	100	181	110
8	6	121	8	147

Figure 4. – Provided board.csv file, here shown in excel. The format is comma-separated strings.

### **HOW THE TWO IMAGES SHOULD BE CREATED**

The 'final line' associated to the last board in the game generates an horizontal "line board", the 'final column' generates a "vertical board"

For the horizontal board image, the colors are, from left to right, the colors corresponding to the numbers in the 'final line', according to the file colorcoding.csv. For the vertical board image, the colors are, from top to bottom, the colors corresponding to the numbers in the 'final column', according to the file colorcoding.csv. The black lines separating the colors in the images were created as 10 pixels wide. The number of pixels per square is asked to the user, but recommended to be between 40 and 60 (or smaller to debug). See Figure 5.

---

<sup>4</sup> Note that your project may be marked with different files contents than the ones provided, but will have the same format and file names.

The 'final line' with the sum of all columns is: [13, 2, 7, 19]  
The 'final column' with the sum of all rows is: [13, 7, 6, 15]

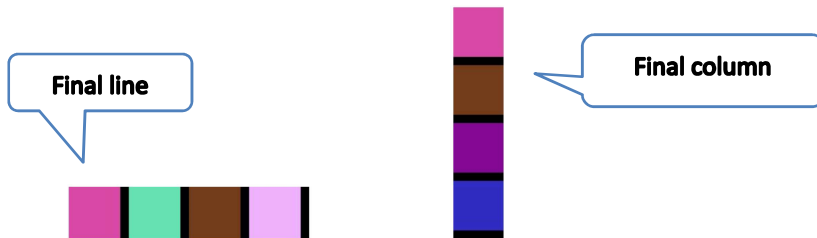


Figure 5 The final line and column and corresponding images. (this corresponds to the sample run 2.3). Notice how the numbers 13 and 7 have the same color in both images, as expected.

**D. DIALOG WITH AND OUTPUT TO BE SHOWN TO THE USER: SEE THE SAMPLE RUNS!!!**

The information that your program asks from and shows to the user should be analogous to and in the same order and detail as in the sample runs. Ask if in doubt.

**E. WHAT YOU ARE PROVIDED**

- This description
- The files board1.csv, board2.csv, and board3.csv
- The file colorcoding.csv
- The file cmpt120imageWeek9.py
- Sample runs
- A helper.py file with some recommended functions to be developed
- Videos and/or additional sample runs may be posted later for further clarifications

**F. WHAT YOU NEED TO SUBMIT**

- Three Python files: main.py, createBoardImages.py and cmpt120imageWeek9.py
  - createBoardImages.py should include the functions that you will use for creating the images at the end of the game
  - Your main file should implement the game, dialog with the user, creating the board, determining winning points, etc. For the creation and manipulation of the images, import the module that you create createBoardImages
  - The cmpt120imageWeek9.py file that you use
- The associated reflection survey

*If you have any questions consult with the Teaching Team. **Make sure that you check email and Canvas announcements in case that additional clarifications are provided.***

*End of description of the "Even-Colourful game" final project*