University of Bristol
MSc in Data Science
EMATM0048: Software Development: Programming and Algorithms (SDPA)

Coursework Assessment (100%)

This unit assessment asks you to apply the skills and tools that you've learned throughout the unit, on a selection of different tasks. General guidelines are as follows:

- **Deadline:** 13:00 (UK time) on Wednesday 12th January 2022.
- **Rules:** Don't share your code with anybody. You are welcome to discuss questions with other students, but don't share the answers. The experience of solving the problems in this project will prepare you for real problems in data science (and life).
- **Support:** You are not alone! Book drop-in sessions, office hours, participate in the forum and talk to your classmates. If you're ever feeling overwhelmed or don't know how to make progress, email me or one of our TAs for help.
- **Plagiarism**: Your work must not be plagiarized. More information is available here: https://www.bristol.ac.uk/students/support/academic-advice/plagiarism/
- **Advice:** Develop your answers incrementally. To perform a complicated task, break it up into steps, perform each step on a different line, give a new name to each result, and check that each intermediate result is what you expect.

Part 1: Software Development (45%)

This part will require you to design, implement, test, and debug a text-based Tron board game using an object-oriented approach with appropriate relationships between classes (considering concepts of inheritance, encapsulation, and polymorphism).

Background:

The Tron game is inspired by the famous 1982 movie *Tron* (https://en.wikipedia.org/wiki/Tron). It is designed based on the motorbike game in the movie. This game can be seen as another version of the Atari game *Snake*.

The basic version of Tron is based on two players/bikers with two colours/ids, say players 1 and 2 for instance. The two players are moving in turns on a grid board that is surrounded by walls. A player can move in four directions (up, down, left, right). As a player moves, it leaves a trail on the spaces it has traversed, which acts as a wall. The game ends if at least one player crashes into the wall, another player or previously visited cell. The player who survives the longest win.

Task description:

In this assignment, you are required to write a text-based Python program using object-oriented programming to simulate the Tron game. Your program should do the following:

A. Create a game board of size m*m (m >3)

B. Ask players for their move each turn
C. Display game board after each legal move
D. End the game if the move is illegal and announce the winner

An example game board is below:

```
#########
#1       #
#        #
#        #
#        #
#       2#
#########
```

This game board is empty and is an example of a default game board that you will get to start with. In the initial version of the game, assume player 1 will play their turn before player 2. Example of expected text-based interaction with the user:

```
>> Enter the board size
4
>> Board of size (4x4) created with default locations
############
# 1|   |   |  #
#  |   |   |  #
#  |   |   |  #
#  |   |   | 2 #
############

>> Entre the move of player 1
R
>> Enter the move of player 2
L
##########
#X| 1 |   |  #
#  |   |   |  #
#  |   |   |  #
#  |   |2 |X #
##########
```

This part of the assignment must be text-based; hence it is not allowed to use turtle, pygame or any other API or external library for implementation.

Step 1: Create your classes

Your program should have at least two classes as follows:

- **Board class** contains attributes to store games states (in two-dimension array) and players information. The Board class should also contain methods to manage the game and print the board.
- **Player class** contains all information related to a player such as position, colour/ID, etc.

Your program must contain appropriate methods to check whether each move is legal (not out of borders, colliding with another biker or previously visited cells). Other functionalities such as changing the default layout for players can be set through any of the classes too.

Step 2: Run your code

Create the main script *(main.py)* to run the classes you implemented. Import your classes into your main script. When run, *main.py* should output a series of prompts to the console, which allows the users to interact with your game. If the move is illegal the game ends and the winner is announced. Your program should display the board after every legal move. Your final submission of main.py works as a demo of your program with different scenarios.

Step 3: Extend your program

A. Modify the program to allow players to move simultaneously, so games can end in a tie if they are on a collision path.

B. Extend your program so the user has an option to play against a computer program instead of entering the move for each of the two players. The computer player must inherit from the player class. Hence, you will have a parent player class and two children's classes: human and computer player. The computer player can choose the next move randomly from the 4 directions.

C. Implement a smart computer player when the computer finds the best non-suicidal move. The following are examples of smart strategies to implement:
    - Exploring directions that have fewer trails.
    - Check ahead for dead ends.
    - You might think of implementing a search strategy.

Step 4: Errors and exceptions handling

Initially, you may wish to assume that all user input is error-free. In this step, you will need to change your code to handle possible errors. Most errors occur in Null values, negative values, and non-integer inputs. Make sure you handle these errors with the right exceptions. Examples of errors to be handled:

- Move that is not listed as one of the 4 directions.
- Repeated player ID/colour.
- A negative value of the initial location of a player

Step 5: Documentation

Create a **Part1.md** text file in your ~/project folder that explains your game project. This file should include your Project Title and a description of your project design, classes, methods and other important

details. If unfamiliar with Markdown syntax, you might find GitHub's Basic Writing and Formatting Syntax helpful.

Your **Part1.md** file should be at least several paragraphs in length and should explain what your project is, what each of the files you wrote for the project contains and does, and if you debated certain design choices, explaining why you made them. Ensure you allocate sufficient time and energy to writing a README.md that documents your project thoroughly.

Bonus question (5pts)

Build another version of the game to be played on a hexagonal grid, instead of a square.

Part 2: Algorithm Analysis (15%)

The city council is developing a new area and wanted to sort the house numbers in a specific street such that houses with odd numbers are on one side and with even numbers on the other side of the road. The layout should be as illustrated in the below picture.

The council want to sort given random numbers of houses to produce such a layout. Given a subset of the list of house numbers as an array of integers (both odd and even), sort them in such a way that the first part of the array contains odd numbers sorted in descending order, and the remaining portion contains even numbers sorted in ascending order.

Example:
Input: {1, 2, 3, 5, 4, 7, 10}
Output: {7, 5, 3, 1, 2, 4, 10}

Input: {0, 4, 5, 3, 7, 2, 1}
Output: {7, 5, 3, 1, 0, 2, 4}

Write a script to implement this sorting method. Discuss clearly what is the complexity of your sorting algorithm with clear reference to your code blocks? ? Make sure your sorting algorithm handles all possible cases.
Hint: you may choose to utilize the sort() built-in function in your program.

Part 3: Data Analytics (40%)

Step 1: Crawl a real-world dataset

Find an interesting external (remotely-hosted) dataset to extract and analyze. Your dataset can be either extracted from APIs and/or via web scraping. ***Reading data from non-external (i.e., local) files is not accepted for this task***. Extracted data must contain at least 5 columns and 150 rows. Save your extracted data in a CSV format.

Examples of data sources include:

- Twitter data
- RSS feeds
- NewsAPIs
- YouTube API
- https://www.gapminder.org/: world progress
- Wikipedia pages
- Nasa website
- Foursquare API
- Space Weather live https://www.spaceweatherlive.com/en/solar-activity/top-50-solar-flares.html
- Echo Nest API

A well-maintained list of APIs can be found here https://github.com/discdiver/list-of-python-api-wrappers

Explain in markdown cells: *Where does the data come from? What are the variables of interest? How was the data scraped/collected?*

Step 2: Perform data preparation & cleaning

- Load the dataset into a data frame using Pandas
- Explore the number of rows & columns, ranges of values, etc.
- Handle missing data, if any.
- Perform any additional steps (parsing dates, creating additional columns, etc.)

Explain in markdown cells: *steps to prepare, clean your data*

Step 3: Perform exploratory analysis and ask questions

- Explore your data, examples are as follows:
  - Compute the mean, sum, range, and other interesting statistics for numeric columns
  - Explore distributions of numeric columns using histograms etc.
  - Explore the relationship between columns using scatter plots, bar charts, etc.
- Ask at least three interesting questions about your dataset. Your questions may have sub-questions. The quality of your questions is assessed based on the complexity and interestingness. Trivial and basic questions will not give you full marks.
- Answer the questions either by computing the results using Numpy/Pandas or by plotting graphs using Matplotlib. Perform grouping/aggregation wherever necessary. You can use

hypothesis testing and basic modelling (such as regression models) to answer the proposed questions.

Explain in markdown cells *Outline in detail your entire analysis. Explain your insights clearly*. You should include a short description of what the intent /interpretation of each plot is.

Step 4: Summarize and write a conclusion using markdown cells

- Write a summary of what you've learned from the analysis
- Share ideas for future work on the same topic using other relevant datasets/sources

Example Projects

Refer to these projects for inspiration:
Analyzing your browser history using Pandas & Seaborn by Kartik Godawat
Understanding the Gender Divide in Data Science Roles by Aakanksha N S
Practical Business Python https://pbpython.com/
Examples from Kaggle: https://www.kaggle.com/eliasdabbas/youtube-data-api
https://www.kaggle.com/deffro/eda-is-fun
Analyzing Worldwide Cuisines https://towardsdatascience.com/analyzing-worldwide-cuisines-with-python-and-foursquare-api-e63455c14246

Code commenting and documentation

Your code must be documented appropriately using docstring comments. Document as you go, not all at the end. One strategy is to write a brief comment about the "one thing" that the function is supposed to do when you declare it, then refer to that comment while implementing it: this helps maintain focus when implementing the function and helps stop yourself from making it do more than the 'one' thing it is supposed to do.

Each class and each method should have at least one docstring which can be brief.

Version Control

You must use version control to keep track of your progress during implementation, using Git. You should perform regular commits as you implement features and fix errors. Your commit comments should reflect the context of the changes that were made in each commit—a fellow developer can always diff the contents to see exactly what changed but that does not provide the context. You should try to ensure each commit comment contains a short subject line. Ensure that the assignment is private, i.e., cannot be seen by people other than yourself. Failure to do that will be considered plagiarism.

Submitting your coursework

Your submission will comprise your entire version control repository for the three parts. Your repository must contain all source files required to run your program.

Your submission must include your code for each part: part 1, part 2, part 3.

- **Part 1:** Includes different Python files for implemented classes, in addition to part1.md file explained in part 1.
- **Part 2:** Jupyter Notebook file to implement and run the sorting algorithm. Answer the questions in the markdown cells.
- **Part 3:** Jupyter Notebook file that includes code and explanation in the markdown for each step. The collected data saved in Step 1 must be submitted too.
- **GitHub markdown (.md file):** to explain any required instructions for running your code.
- If you have used any additional code, beyond standard Python packages, then you will need to include an additional subdirectory containing additional code required to run your scripts. The authorship of any such code should be clearly stated in GitHub markdown.

For submitting your work on Blackboard, you will need to submit a zip file with the name ("UOBusername_EMATM0048). Your zip file should contain the latest version of your GitHub repository along with a text file of your GitHub repository link. You will need to add "**SDPA-UoB**" as a collaborator to your repository. Please note ***you must NOT change your remote repository once you added SDPA-UoB as a collaborator.***

Grading

Your grade will be assessed on both correctness and style.

- ***Correctness***: How well the program works *according to specifications*. *How good is the quality of your analysis*.
- ***Style***: The quality of the code and documentation.

Your code should be well-written and well-commented. You are encouraged to format your code per Python style guidelines. It should be clear enough for another programmer, such as the course staff, to understand and modify if needed. Quality code is expected to meet our style requirements:

- Every function written is commented, in your own words, using a docstring format that describes its behaviour, parameters, returns, and highlights any special cases.
- There is a comment at the top of each code file you write with your name, section, and a brief description of what that program does.
- All requirements for each of the parts are met.
- If version control is not submitted/not maintained regularly or made public, 10 pts will be deduced.