CPSC 230 Computer Science I (Instructor: Dr. T. Qi)

Final Project – Adventure

Total Score: 100 pts + 10 pts (Bonus)

Goal: Implement an adventure game using Python programming **and** prepare a presentation to demonstrate your program and results.

Deliverables:

1) **A working Python program (50%)**
   - **Implement** based on the "**Python Program Instructions**" (**Pages 2-5**)
   - **Submit**
     - Your code as "**Advanture_YOURNAME.py"**
2) **Final presentation (50%)**
   - **Prepare** a final presentation based on the "**Final Presentation Instructions**" (**Page 5-6**)
   - **Present** your final presentation with a live demo **during the class**
   - **Prepare a demo (video recording)** and **narrate** your working Python program.
     - Run through all possible configurations of your game:
       - For the **First-Step Game (Password Guessing)**, demonstrate different outcomes, including correct guesses, incorrect guesses, and any hints provided.
       - For the **Second-Step Game (Four Puzzles)**, show the outcomes of all four puzzles, including both winning and losing scenarios for each.
   - **Submit**
     - Your Demo as MP4 **"Demo_YOURNAME.MP4" (20%)**
     - Your final presentation as a PDF **"FinalPresentation_YOURNAME.PDF" (30%)**

Grading Criteria:

1) Your project will be graded based on how well your Python program executes and completes the assigned tasks (see **Python Program Instructions**).
2) This is your chance to demonstrate the skills and knowledge you've gained in the course! **You are expected to apply concepts covered in lectures to complete the project**.
3) If you use any code or resources not covered in class (e.g., code from the internet or generated by AI), **you must clearly acknowledge these sources in your final presentation slides**, clearly specifying which parts of your code are not original. **Failure to do so may raise academic integrity concerns and impact your grade.**
4) **You may be asked to explain your code during the presentation**. Your ability to clearly explain and demonstrate an understanding of your code will contribute to your grade.

Python Program Instructions

Create a script **Adventure.py**

0. Greeting Player----------

1. Print a welcome message.
2. Ask for the player's name and store it in *user_name*.
3. Initialize the player with 4 lives (*user_lives*) and 0 coins (*user_coins*).

1. First-Step Game: Password Guessing----------

- **Objective:** Guess a 3-letter password following specific rules, with hints provided.
- **Function:** Implement the game in a function called *password_guessing*(), returning True (win) or False (loss).

Rules:

1. **Password Definition:**
    - Predefine a password, which is a 3-letter word:
        - 2nd letter is a vowel (a, e, i, o, u).
        - 1st and 3rd letters are consonants.
        - All lowercase.
    - Store the password in *password*.
2. **Gameplay:**
    - The player guesses each letter of the password, stored in *user_guess*.
    - Display “123” as a placeholder for the password.
    - Ask the player to specify which letter to guess by typing '1', '2', or '3' and then ask for their guessed letter.
    - After a correct guess, reveal the letter in the corresponding position (e.g., “d23”).
    - Use *replace*() to update the guessed password.
3. **Hints for Incorrect Guesses:**
    - Provide **one clue at a time** after each wrong guess:
        - **Clue 1**: “It is a vowel/consonant.”
        - If they guess vowel/consonant correctly, display **Clue 2**: “It’s next to [*some letter on the left of their guess in A-Z*].” (e.g., for 'b', the hint could be "The letter is next to a").
4. **Attempts:** Player has 5 attempts to guess all letters.
5. **Winning Condition:** If the player guesses the password within 5 attempts, they proceed to the next game without losing lives.
6. **Losing Condition:** If they fail to guess the password:
    - Ask if they want to try again:
        - Yes: Lose 1 life and continue with current progress.
        - No: Lose 2 lives and proceed to the next game.

- End the game if all lives are lost and print a goodbye message (you may use exit() to end the program).

7. **Life Tracking:** Inform the player of the remaining lives **before and after** each decision.

Bonus (up to +3 pts)

- Randomly generate a new 3-letter password each time (one random vowel and two random consonants; vowel in a different position each time).
- Generate dynamic hints based on the player's guesses (vowel/consonant status and letter position).
- **Hints**:
  - Use random.randint(a, b) to generate random characters by their indices (random integers).
  - Use the string method .index() to access the position of letters. For example, to get the index of "e" in the variable *my_string* = "hello", you may use my_string.index("e").

2. Second-Step Game: Four Puzzles----------

Welcome the player to the Four Puzzles game and reward them 100 coins. Update *user_coins*.

Gameplay:

1. Ask the player, "Which direction do you want to go, [*name*]?" with options: 'N', 'S', 'E', 'W'.
   - If the input is invalid, prompt again until a valid direction is provided. Handle both uppercase and lowercase inputs.
2. Each direction leads to a different puzzle (**a function for each puzzle**). The player can earn or lose coins and lives, except for direction 'W', where they use coins on treasures.
3. Once a puzzle is completed, **it cannot be replayed,** but **the player can continue to play other puzzles**. However, once they choose 'W' **or** have completed the other three puzzles**, the puzzle of W is the last puzzle played**, and the **win/loss of the entire advantage game is up to the result of Puzzle West**. Your program should keep track of which puzzles the player has played.
4. Always display the player's remaining lives and coins before each puzzle.
5. End the game if the player loses all lives and print a goodbye message (you may use exit() to end the program).

Puzzle North: Dice Rolling ----------

- Roll a 12-sided die (using random.randint()), and ask the player to guess the number.
- If their guess is **less than** 5 from the correct number, their gain is based on how close their guess is to the correct number (e.g., they gain 5 coins if they guess it correctly; they gain 4 if off by 1, and so on).
- Otherwise, they lose 5 coins and 1 life.

- Implement this in the function puzzleN_dice(), returning True/False.

Puzzle South: BACKWARDS ----------

- **Randomly** generate a four-letter word (all lower cases; letters **may repeat**)
- Show the word to the player and ask them to type it backward
    - Correct answer: Gain 5 coins.
    - Incorrect answer: Lose 5 coins and 1 life.
- Implement this in the function puzzleS_backwards(), returning True/False.

Puzzle East: Sum of Digits ----------

- **Randomly** generate a four-digit number (**no** repeated digits)
- Present a four-digit number and ask the player to sum its digits. For example, for the number 1234, the sum is 1+ 2 + 3 + 4 = 10. **You must use a for-loop** to calculate the sum.
    - Correct sum: Gain 5 coins.
    - Incorrect sum: Lose 5 coins and 1 life.
- Implement this in the function puzzleE_sumdigits(), returning True/False.

Puzzle West: Maximize Your Treasure----------

- The player faces a treasure chest with five items: "sword" (40 coins), "shield" (35), "potion" (25), "gold" (50), and "gem" (60). **The goal is to collect as much treasure as possible without exceeding the coins they have**.
- Use a **dictionary**, *inventory*, to store the items (strings) and their values (integers). Track collected items in a **list**, *treasures*, and total value in *treasure_value*.
- Ask the player if they'd like to take a draw on the next treasure in the *inventory*.
    - If they type 'yes', **randomly** select an item from the inventory and show its value to them.
        - If they can afford it, deduct the item value from *user_coins*, and remove the item from the *inventory*. Update *treasures* and *treasure_value*.
        - If they cannot afford it, **they lose the game and 1 life**. You should not update *treasures* and *treasure_value*.
    - If they type 'no', check if they have won the game:
        - If the remaining coins are less than the lowest item value in the *inventory*, **they win!**
        - Otherwise, **they lose the game and 1 life**.
- Implement this in the function puzzleW_treasure().

Bonus (up to +7 pts)

Below are two **optional** extensions to the basic task of Puzzle West, as described above. You may come up with your own extension; to be eligible, **you must discuss it with your instructor beforehand** to ensure the scope and difficulty are appropriate. You are only allowed **two extensions in total**. You may earn up to 3 pts for one extension and up to 7 pts for two.

Extension #1: Magic Items & Set Bonuses

- Add 3 – 5 magic items with hidden effects, which are only revealed after purchase.
- Collecting certain **sets** (more than one magic item) gives extra coins or extra lives.
- Minimum requirements (and be creative!):
    - Identify hidden effects for each magic item; effects can be positive, negative, or random (e.g., lower/raise item initial coin cost)
    - Implement at least two different set bonuses:
        - e.g., collect "ruby" + "emerald" -> + 10 coins; collect all three magic items -> +1 life, etc.
- Adjust the win/loss logic in the basic task so that this bonus would work

Extension #2: Carry-Weight Constant

- Add a backpack and set the capacity (e.g., 50 lbs).
- Each item now has both value and weight; extend *inventory* to store {item: (value, weight)}
- Track *total_weight* of your backpack and prevent taking an item if it would exceed capacity, even if coins allow.
- **Win condition**: maximize value without busting either coins or weight.

Final:

- Overall game success in the adventure depends on their performance in Puzzle West.
- Print the player's result (Win/Loss) and list their collected treasures, remaining lives, and coins (and backpack weight if applicable).
- End the game.

Final Presentation Instructions

Prepare a final presentation (**5 minutes max**) to showcase your project. Use Google Slides (**maximum 10 slides**), and **save your presentation as a PDF**.

Slide 1: Title Page

- Include the project title, your name, course code, and date.

Slide 2: Project Completion Overview

- **First-Step Game: Password Guessing**
    - Did you complete this game? (Full / Partial / Incomplete)
    - Did you implement any bonus features? (Yes / No)
- **Second-Step Game: Four Puzzles**
    - Report the status of each puzzle (North, South, East, West): Full / Partial / Incomplete
    - Did you implement any bonus features? (Yes – How Many? / No)

Slide 3: Design of the First-Step Game

- Describe the design choices you made for the Password Guessing game.
    - How did you structure the code?
    - Any **special methods or logic** you used (those are not introduced in this course)?

Slides 4-7: Design of the Four Puzzles

- For each puzzle (North, South, East, West):
    - Explain how you approached and implemented the puzzle.
    - Highlight any unique design choices or challenges.
- If you have implemented the extension(s), you must describe the game logic and implementation for **each** extension.

Slide 8: Challenges

- What challenges did you face during the implementation process? How did you address and overcome these challenges?
- Share the most interesting or valuable things you learned during the project.

Slide 9: Improvements

- If you had more time, how would you improve your project? Feel free to propose new features or ideas that could enhance the game's fun and engagement.

Slide 10: Acknowledgments

- Mention anyone who helped you with your project.
- If you used external resources (e.g., websites, tutorials), include the links.
- If you used AI tools to assist with your program, specify which parts you applied them.