OOP Project(Word Guess Game)

Anyone found in copying the project from any other group from any source, both the groups will get F grade in Course as well as in the Lab.

For your information project will be evaluated by only one panel committee including Course + Lab instructors to check the quality of code and plagiarism.

Note: Carefully read the following instructions.

- **2 students are allowed per group. No cross-section is allowed.**
- **Submit your roll number in each submission.**
- **Using all concepts of OOP is mandatory. And concepts should be used correctly.**
- **Your code should be clear and add exceptions where required [to avoid system to crash or accept incorrect data/options]**
- **Naming Conventions should clarify the purpose of variables and functions being used.**
  - **Names of Classes should start with a Capital Letter.**
  - **Names of Variables should start from a Small Letter.**
  - **Identifiers should be named using camelCase.**
- **Project is to be submitted as a zip file.**
- **Use 3 file structure for implementation**
- **Submit the whole project by one member of the group.**
- **There must be a block of comments at start of each function; the block should contain brief description about functionality of code.**
- **Use understandable name of variables.**
- **Proper indentation of code is essential.**
- **Make a Microsoft Word file and paste all of your C++ code with all possible screenshots of every task output in MS word and submit .cpp file with word file.**
- **Make separate .cpp files for all tasks and use this format 22F-**

1234_22F-1235_Filename.cpp/.h.

- First think about statement problems and then write/draw your logic on copy.
- After copy pencil work, code the problem statement on MS Studio C++ compiler.
- At the end when you done your tasks, attached C++ created files in MS word file and make your submission on Google classroom. (Make sure your submission is completed).
- Please submit your file in this format 22F-1234_22F-1235_Project.
- First Product Deliverable contains structure of classes and submitted on [15/11/24], files should have data to check and evaluate each module.
- Final Product Deliverable completed and integrated with all files submitted on [01/12/24], files should have data to check and evaluate each module.
- Submission information
    - Submission will be through google classroom. Lab teachers will make portals for submission.
    - You need to submit the final project code [write roll numbers of group mates in a comment at top] txt files and readme file contain the information about modules completed and the sequence to run the program.
- Use windows Form to get bonus marks

GOOD LUCK ☺

Project statement

Game Overview:

In this word-guessing game, the player will search for words hidden in a **2D grid of letters**. The words can be located consecutively in **rows, columns, or diagonals**. The player has two difficulty modes, **Easy and Hard**, and each mode **has three levels** with **varying requirements for word length** and number of words to find. The player has five total chances to complete all three levels across the selected difficulty mode.

Game Objective:

The objective is to guess the required number of words on each level by extracting words of the specified length from a 2D grid. Each mode and level introduces unique challenges and constraints based on word length and word count.

Major entities along with their functionalities are described below:

Divide the modules in start with your group member and everyone is responsible to his own completed modules. Make sure you combine the project from time to time to avoid any integration issues later.

You need to design the hierarchy of classes to be defined in the system first. All the components should be implemented in OOP concepts. [constructers, Inheritance, virtual function, friend-class/functions, static data members, polymorphism, operator overloading, three files' structures etc.]

Game Modes:

1.Easy Mode:

- **Word Length:** 2 to 4 characters.
- **Levels:**
    - Level 1: Guess 3 words.
    - Level 2: Guess 5 words.
    - Level 3: Guess 7 words.
- **Data Source:** easy_words.txt containing 50 words of lengths 2-4.

2.Hard Mode:

- **Word Length:** 5 to 7 characters.
- **Levels:**
    - Level 1: Guess 3 words.
    - Level 2: Guess 5 words.
    - Level 3: Guess 7 words.
- **Data Source:** hard_words.txt containing 50 words of lengths 5-7.

Core Game Mechanics:

Grid Setup: A 2D array is populated with random letters, with hidden words scattered across rows, columns, and diagonals.

Word Guessing: Players must locate and enter words they find in the grid. Words must follow the length constraints specified by the mode and level.

Chances: Players have 5 total chances to complete all levels within the selected difficulty mode.

Game Flow

1.Game Initialization:

- The Game class displays the project name, clears the screen, and then displays the main menu.
- **Main Menu Options:**
  - **1. Start Game:** Begins gameplay by allowing the player to select Easy or Hard mode.
  - **2. Option:** Displays game instructions and rules.
  - **3. Highest Score:** Shows the top 5 scores from each mode (loaded from a text file).
  - **4. About Us:** Displays information about the project members.
  - **5. Exit:** Exits the game.

2.Level-Based Gameplay:

- The selected Difficulty mode (Easy or Hard) initializes the grid with a size based on the level:
  - **Level 1:** 10x10 grid
  - **Level 2:** 15x15 grid
  - **Level 3:** 20x20 grid
- **Grid Allocation:** The grid is dynamically allocated by allocateGrid(size) based on the level.
- **Word Validation:** Player guesses are validated using the WordValidator object.
- **Score Update:** The ScoreTracker updates the player's score after each successful word guess.

3.End of Game:

- After all levels are complete, the player's score is checked against the top 5 scores.
- If it qualifies, the ScoreTracker updates the list of top scores and saves it to the

text file.

Note: You can take assumptions where needed but use proper relationships in classes.

1. **Change Console Text Color:**
   - **Windows**: #include <windows.h>, use SetConsoleTextAttribute.
   - **Linux/Unix**: Use ANSI escape codes (e.g., "\033[1;32m" for green).
2. **Clear Console Screen:**
   - **Windows**: system("CLS").
   - **Linux/Unix**: system("clear").
3. **Add Delay (Pause Execution):**
   - **Windows**: #include <windows.h>, use Sleep(milliseconds).
   - **Linux/Unix**: #include <unistd.h>, use usleep(microseconds).
   - **Cross-Platform**: #include <chrono> and #include <thread>, use std::this_thread::sleep_for(std::chrono::milliseconds(milliseconds)).

Example 10x10 Grid (Easy Mode, Level 1)

```
C A T X Y B E E T S
A P E B A T C U P I
B I G R A T P I E T
L O W H E N M A T S
O W L S I N B U G S
R U N C A T T O Y E
B E A R L I P P O W
S I N K P E N C O W
A N T R U G C A R D
F O X T E N P I G H
```

Select character:
Input row: 0
Input column: 0
Selected character at position [0,0] = C

Select character:
Input row: 0
Input column: 1
Selected character at position [0,1] = A

Select character:
Input row: 0
Input column: 2
Selected character at position [0,2] = T
Word complete remaining words to guess are 2

.
.
.

Note: If word is completed than the blank or white space is replaced to the word.

Words by Length:

- **2-Letter Words**:
  - AP, IT, TO, IN, UP
- **3-Letter Words**:
  - CAT, BAT, RAT, BIG, OWL, RUN, ANT, COW, FOX, TEN, MAT, PEN
- **4-Letter Words**:
  - BEAR, SINK, CARD, HEN, BUGS, TOYS, CUP, LION

Sample Word Placement:

- **Horizontal Words**:
  - Row 1: CAT, BEE, TAP
  - Row 2: APE, BAT, CUP
  - Row 4: HEN, MAT, SINK
  - Row 5: OWL, BUGS
  - Row 6: RUN, CAT, TOYS
  - Row 7: BEAR, LION
  - Row 8: SINK, PEN, COW
  - Row 9: ANT, RUG, CARD
  - Row 10: FOX, TEN, PIG
- **Vertical Words**:
  - Column 1: CAB, OWL
  - Column 4: ANT, HEN
  - Column 5: SUN, BUG
  - Column 7: PEN
  - Column 10: DOG
- **Diagonal Words**:
  - CAT, RUG, BAT

Note: To avoid random characters word, you can use any logic that will help the player to avoid missing the 5 chances.

GOOD LUCK ☺