CSCI 310 GUI & GAME

Project 3: C# Console Game

Goal: In this project, you will design and implement an interactive console-based game using C#. The game should run entirely in the console window and provide a responsive and engaging gameplay experience. While the inspiration comes from classic block-based games, you are not allowed to recreate Tetris. You are encouraged to be creative and implement your own unique game mechanics.

Learning Objectives
By completing this project, students will:
- Practice working with C# console input/output, including reading keys and controlling console display.
- Implement real-time game logic.
- Design game state management, including handling starting, running, and game-over states.
- Apply OO design and programming for a complete game.

Requirements:
Regardless of your chosen game type, every interactive console game shares fundamental components. Consider these abstractions as you design and build. Your game must include the following functional features:

- **Game Interaction**
    - Use keyboard input for player control or value entry.
    - Design and Implement a start menu or prompt (e.g., “Press any key to start the game”).
    - Design Implement a game-over panel or message showing the final score.

- **Gameplay Mechanics**
    - The game must have dynamic and real-time updates (e.g., moving objects, timers, falling pieces, scoring, game values, or enemies).
    - Enforce playable boundary constraints: there is a playable area, for showing values, objects, or etc. Showing what things depends on your game types.
    - Can allow actions such as moving, rotating, shooting, or dropping items, depending on your game concept (optional)

- **Game State and Score**
    - Track score, level, or other relevant game metrics.
    - Update the game state based on player actions and game rules.
    - Provide restart and quit functionality at game-over.

        - Start/End States: Consider how your game begins (e.g., immediate start) and how it might conclude (e.g., achieving a goal, running out of lives). While full overlay UIs are not required for this simplified approach, a simple console log or on-screen text for "Game Over" is sufficient.
        - Reset Mechanism: Provide a way to reset the game to its initial state (e.g., by pressing a specific key like 'R'), allowing for replayability.

- **Technical Requirements**
    - The game must be written in C# and run in the *console* application.
    - Do NOT use any graphical libraries (e.g., Unity, WinForms, WPF) or APIs, it will NOT work.
    - Implement smooth gameplay, avoiding freezing or blocking input unnecessarily.
        - You may need to use async input handling or Console.KeyAvailable loop for non-blocking gameplay.

- **Scoring & Feedback**
    - Progress Tracking: There should be at least one indicator to show the game progress.
        - Ex: score based on targets hit, items collected, or time survived.
    - Visual Feedback (optional): Provide clear visual feedback to the player if needed (e.g., objects disappearing upon impact, score updates, or simple color changes).

Suggestions
- Creativity is encouraged: your game can be puzzle-based, action-based, simulation, or other genres suitable for a console interface.
- Focus on **correct and smooth gameplay**, even if graphics are minimal.
- Pay attention to **edge cases** such as game-over conditions and simultaneous key presses.
- You are welcome to improve our class' sample console games.

Grading Policy (100pts + 15pts)
- Fulfill above requirements (70pts)
    a. Game functionality and playability
    b. Input handling and responsiveness
    c. Smooth operations and game flow
- Visuals Design (10pts)
- Code organization and readability (10pts)
- Uniqueness and creativity (10pts)
- Bonus (15pts) – very fun and creative

Presentation (20pts)
- Prepare for a 5 minute presentation (i.e., 5 minutes for presentation + 1 minutes Q&A)

- Show how you develop the game
- Individual contributions if you have a team
- Demonstrate your game and show you follow the requirements

Developer Notebook (30pts)

- Follow the specification and document your development well.

Each item will be graded based on:

- Well-completed (100%)
- Completed (85-100%)
- Developing (75-85%)
- Progressing (60-75%)
- Under-developed (under 60%)

Individual contribution:

Grading will be also based on individual contributions in the team. Team members who contribute almost equally will receive the same contribution scores.

Submission:

- Submit your program, developer notebook, and presentation to BrightSpace.