Project Tasks for Milestone 1: Build the Game Engine Foundations

Your task for Milestone 1 is to explore the basics of rendering objects on screen using Simple DirectMedia Layer 3 (SDL3). While this code may not appear to be related to a game engine, it is laying the foundations upon which you will build out your game engine throughout the course of this semester. Milestone 1 deliverables have team parts and individual parts. As always, you are expected to abide by the University's Academic Integrity Policy (http://policies.ncsu.edu/policy/pol-11-35-01), which includes providing appropriate attribution for all external sources of information consulted while working on the milestone tasks.

There are 125 points available on this milestone. Students enrolled in 481 are required to complete the first 100 points (Tasks 1–5) but may also choose to complete Task 6. Students enrolled in 481 earning scores above 100 will receive a 100 on the milestone (i.e., extra points will not be given as extra credit). Students enrolled in 581 are required to complete all 125 points and will receive a grade as the percentage of the 125 points they earn. All students are required to submit a writeup addressing the tasks of the milestone they have completed.

Game Engine Requirements (Team Submission)

This is the core, reusable code that your whole team will build together. Think of this as the "chassis" and "engine" of a car that everyone on the team will then customize.

- **Task 1: Core Graphics Setup**
    - Create the foundational code to initialize SDL3.
    - Write the functions to create a window (1920x1080) and a renderer in SDL3.
    - Implement the main game loop that clears the screen to a blue color, prepares the scene, and renders it.
- **Task 2: Generic Entity System**
    - Create a general system or class for "entities" or "game objects."
    - The **engine** needs the *capability* to draw and update the position of any given entity. It shouldn't care *what* the entity is, just that it has a position and a texture/shape to render.
- **Task 3: Physics System**
    - Build a generic physics system.
    - This system must include a gravity feature that applies a constant downward acceleration to an object.
    - Crucially, the strength of gravity must be **configurable** (e.g., via a function like `Physics.setGravity(float value)`) and not hard-coded.
- **Task 4: Input Handling System**

    - Create a manager or system that reads the keyboard state using `SDL_GetKeyboardState`.
    - This system should provide a simple, abstract way for the game to check if a key is currently pressed (e.g., a function like `Input.isKeyPressed(SDL_SCANCODE_W)`).
- **Task 5: Collision Detection System**
    - Implement a generic function or system that can detect collisions between two game objects.
    - Using bounding box collision (like with `SDL_HasIntersection`) is the recommended starting point. The system should be able to take two entities and return a `true` or `false` value indicating if they are overlapping.
- **Task 6: Scaling System (581 Required)**
    - Integrate logic into your rendering system to handle two different scaling modes.
    - Implement a mechanism to toggle between **constant size** (pixel-based) and **proportional** (percentage-based) scaling. This toggle should be triggered by a key press, linking to your input system.

Individual Game Requirements (Individual Submission)

This is your personal application that *uses* the team's engine. You are building a specific "custom car" using the shared chassis and parts. Your code will be unique to your game.

- **Task 1: Running the Engine**
    - Your `main.cpp` will call the engine's initialization and main loop functions to get your game window running.
- **Task 2: Specific Entity Implementation**
    - Use the engine's entity system to create three specific objects for *your game*:
        - **A static object:** A wall, a platform, a tree—anything that doesn't move.
        - **A controllable object:** The player character.
        - **An auto-moving object:** An enemy patrol, a moving platform, etc., that follows a continuous, predefined path.
    - The appearance and movement pattern of these objects are unique to your game.
- **Task 3: Applying Physics**
    - Use the engine's physics system in your game.
    - You will need to decide which of your entities are affected by gravity. For example, your player character might be affected by gravity, but a floating enemy might not be.
- **Task 4: Implementing Controls**
    - Use the engine's input system to define the controls for your player object.
    - Your code will check for specific key presses (e.g., W, A, S, D) and call functions to move your controllable entity.

- **Task 5: Handling Collision Responses**
    - Use the engine's collision detection system to check for overlaps between your specific game objects.
    - You must write the code for what happens after a collision is detected. For example:
        - If the player hits the static object, stop the player's movement.
        - If the player hits the auto-moving object, you might reset the player's position or print a message.
- **Task 6: Demonstrating Scaling (581 Required)**
    - Ensure your game correctly demonstrates the engine's scaling feature, toggling between the two modes when the designated key is pressed.

Deliverables, Writeup, and Rubric

This section clarifies what you turn in, and how it is graded.

- **Team Submission (25%):** A single submission for the whole group containing all the shared **Game Engine** code, including documentation of the **engine** design (e.g. where each of the tasks is addressed in the engine). Documentation regarding the design is required to receive credit for the team submission portion.
- **Individual Game Submission (25%):** Each person submits their own project file containing the code for their **Individual Game** which uses the team's engine.
- **Individual Writeup (50%):** Each person writes and submits their own reflection paper. This paper should discuss why you made the design decisions for the **engine** (as a team) and the specific implementation choices you made in your **individual game**.

The grading breakdown is as follows: Tasks 1-5 are each worth 20 points. Task 6 is worth 25 points. Each task split into the Team Submission, Individual Game Submission, and the Individual Writeup.

	No Credit	Half Credit	Full Credit
Task 1	No Game Loop No Window Opens No Writeup addressing task	Window Opens Partial game loop Writeup addresses Task, but nothing substantial	Window Opens and renders objects Full Game Loop with playable game Writing substantially addresses task
Task 2	No entity system No entities in game No Writeup addressing entities	Partial entity system Not all required entities are present Writeup partially	Full entity system All required entities present Writeup addresses entity

		addresses entity system	system
Task 3	No Physics system Game doesn't use physics No writeup discussing physics	Partial physics system or hardcoded values Game uses some physics, or have hardcoded gravity Writeup partially discusses physics system design decisions	Modifiable gravity system which can be applied to entities Game showcases gravity system Writeup discusses physics system design decisions
Task 4	No input system, or system uses SDL events (beyond window close) Game does not permit movement No writeup discussing input system design		Input system exists and does not use SDL events beyond window close even Game allows movement on controlled entity Writeup discusses input system design
Task 5	No collision detection system in engine No collisions are detected in game No writeup discussing collision system design	Partially working collision detection system Collisions are resolved (with some potential errors, or are notified via text) Writeup addresses some collision system design	Working collision detection system in engine Collisions are resolved without errors Writeup discusses collision system design in depth
Task 6	No scaling functionality in engine Game cannot scale correctly No Writeup discussing scaling system design		Scaling functionality in engine Game scales correctly Writeup addresses scaling system design