This is a basic match-3 game that requires several enhancements and optimizations to improve gameplay and performance. The following tasks should be completed with a focus on maintainable and well-structured code:

Guidelines: Ensure SOLID Principles

As you implement the tasks below, ensure that the codebase adheres to SOLID principles. Each feature should be implemented with clear responsibilities, modularity, and maintainability in mind, using design patterns where appropriate to enhance flexibility and readability.

Task 1: Implement Cascading Gem Drop Logic

Develop logic to prevent unintended matches when new gems drop onto the board. Ensure that when gems are matched and new ones cascade into place, they do not inadvertently create additional matches upon spawning. In cases where it's unavoidable, aim to generate only the minimum necessary matches.

- Cascading gems should fall one by one to fill the empty slots, rather than dropping as a group.

Task 2: Create a Gem Pooling System

Replace the current method of infistantiating new gems with an object pooling system to enhance performance. Reuse "destroyed" gems from the pool when new gems are required on the board. If the pool is empty, instantiate new gems as needed.

Task 3: Special piece - Bomb

Creation:
Matching 4 or more pieces of the same color will generate a bomb piece (AKA special piece).
The special piece will be spawned on the same piece slot that created the 4+ match (user action), independently from the regular pieces' refill logic (Cascading).

- **Special piece color** must be the same as the 4+ pieces that created it, for example matching a 4+ red piece will create a red bomb.

Matching bomb:
Matching a special piece is possible with any other special pieces (e.g bomb with bomb), or with 2 or more other pieces (regular or special) of the same color group.

- Matching with 3 or more regular pieces will create another bomb, following the same logic above (matching 4 or more pieces).

Bomb destruction:
By matching a bomb, it will be destroyed but only after destroying its neighbor pieces group as the following pattern:

		x		
	x	x	x	
x	x	Bomb	x	x
	x	x	x	
		x		

- Allow to set any amount of delay (in seconds) before destroying the neighbor group.
- Allow to set any amount of delay (in seconds) before destroying the bomb piece itself.
    - It can be done with or after the explosion of the neighboring group, but not before.
- The regular pieces' refill logic (Cascading) must start only after the bomb special piece was destroyed.

Task 5: Implement Staggered Gem Drop Animation

When new gems spawn or existing gems drop to fill empty spaces, they all move as a single unit, creating a bulk-drop effect that feels abrupt and visually unappealing. The goal is to create a **staggered drop animation** where gems fall one by one in a cascading motion, giving a smoother and more visually appealing effect.

- **Individual Drop Timing**: Adjust the timing so that each gem falls slightly after the previous one, creating a chain-like staggered effect.
- **Animation Timing**: Ensure that each gem's animation begins as soon as the one above it has dropped into place, or with a slight delay between neighboring gems.
- **Visual Reference**: Check the provided link for an example of the desired effect (e.g., the gem drop effect in *Royal Match* Level 1). Observe how each gem animates individually and the way each gem's fall is triggered in sequence, providing a polished, professional appearance.

Royal Match Level 1 - NO BOOSTERS GAMEPLAY 👑 | SKILLGAMING ✔️

Documentation Request
At the end of development, please include a short description of each feature, system, or major logic component created or refactored during this task list. This summary should explain each feature's purpose, how it integrates with the existing code, and any design patterns or principles used.