Alex Agruso, Mauro Alvizo Jr.
Daniel Arreguin, Jake Bram

1 Introduction

1.1 Project Scope

Our program will be a simple powder-toy physics simulator. It will be a standalone desktop application written in C++ using the SFML multimedia library. The UI will consist of a grid of pixels, with each pixel represent a different simulated particles. Each particle has an element which determines how it interacts with other particles. The user will be able to add and remove particles from the board and watch the physics simulation unfold.

2 Functional Requirements

2.1 User Interface

The user interface will consist primarily of two sections: element selection, and the game board. Users will be able to choose which element they would like to use by clicking its respective button. Next, they will click and drag on the game board which will add particles of that element to the screen. The physics simulation runs continuously, unless the user decides to pause it. Finally, the user will be able to save and load board layouts, saving a currently running simulation for later.

2.2 Particles and Elements

Physics interactions occur between particles, with each particle being of a certain element. A particle's element determines how it interacts with other particles.

2.2.1 Element Classes

There are four primary classes of elements: solids, liquids, gasses, and powders. Each is described as follows:
Solid: Static and rigid; is not affected by gravity.
Liquid: Dynamic; flows freely but is affected by gravity and collides with other particles.
Gas: Dynamic; diffuses evenly, is not affected by gravity but still collides with other particles.
Powder: Dynamic; forms piles, is affected by gravity, and collides with other particles.

We currently plan on implementing 20 different elements, with interactions documented in the following sections.

2.2.2 Solid Elements

Wall: Is indestructible and does not interact with other elements. Purely serves as a collision boundary.
Metal: Can be melted by lava, and can be electrified using spark. Electrified metal can also ignite TNT and gunpowder.
Wood: Can be burned by fire or lava, producing smoke.
Glass: Produced when sand touches lava. Produces stone when destroyed with TNT or gunpowder.
TNT: Explodes when touching fire, lava, or electrified metal. Destroys most materials.
Plant: Can be burned by fire or lava, producing smoke. Grows like an amoeba if in contact with wood.
Rock: Can be melted bny lava, and produces stone when destroyed with TNT or gunpowder.

2.2.3 Liquid Elements

Water: Puts out fire, producing steam. Also cools lava into stone. Freezes when contacting ice.
Lava: Very hot, ignites anything that is flammable. Cools into stone when contacting cold objects.
Poison: Flammable, corrupts plant into more poison.
Oil: Flammable, floats on top of water.

2.2.4 Gas Elements

Smoke: Produced by several elements when burned.
Fire: Ignites anything that is flammable. Can be put out using water.
Steam: Produced when water evaporates.
Gas Vapor: Flammable, passively produced by oil.

2.2.5 Powder Elements

Stone: Melts when contacting lava.
Sand: Turns into glass when in contact with fire or lava.
Gunpowder: Explosive, can be ignited by fire and lava, as well as electrified metal.
Ice: Very cold, freezes things upon contact.

2.2.6 Miscellaneous Elements

Spark: Can be used to electrify metal.

3 Non-Functional Requirements

3.1 Event System

Our program will be built on an event system. Any interaction between different components of the program will be handled as an event. There will be many different types of events for our application, many of which will have no common functionality with each other. In order to limit unnecessary abstraction, we will implement the events as an algebraic data type using `std::variant`. We will also use pattern matching with `std::visit`.

3.2 Physics Engine

For stability, the physics engine will be limited to running at 60 ticks per second. Each particle will be represented as an instance of `std::optional` wrapping a particle physics instance. If the particle is empty, or air, it takes the type of `std::nullopt`, while if it is not empty, it takes the type of a standard physics particle. The engine will be single-threaded, but a multi-threaded implementation will be considered if performance is an issue.

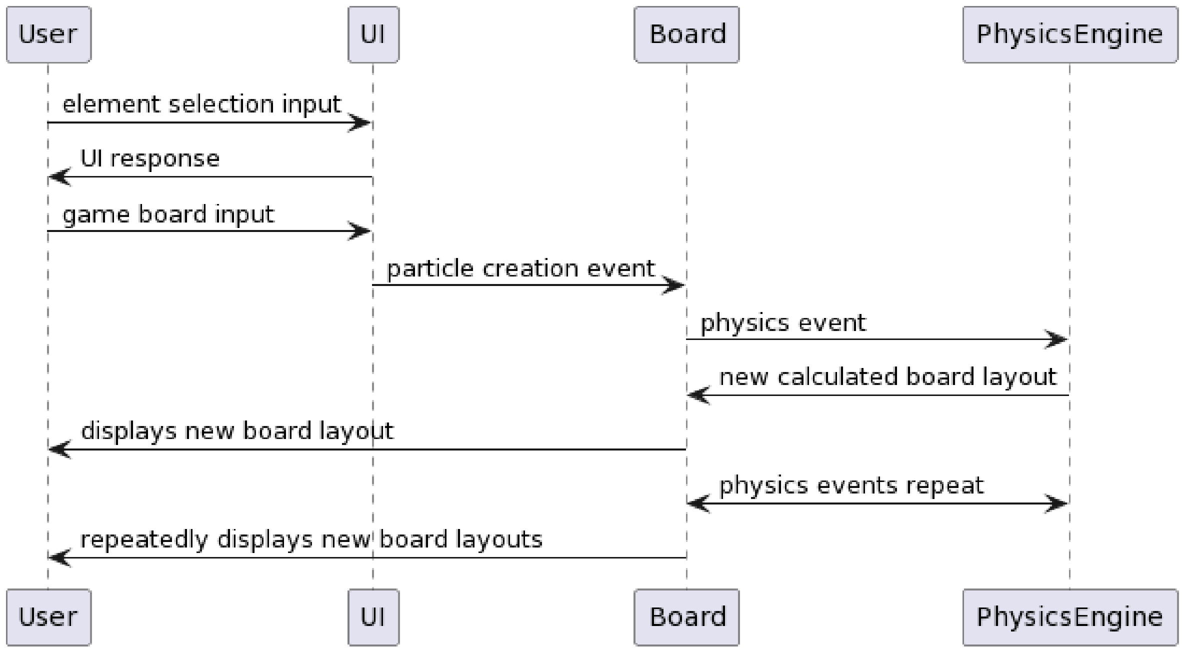
3.2.1 Simulation Size

The physics engine should be able to fulfill the above requirements for a simulation that is 800 pixels wide and 600 pixels tall.

3.3 Platform Compatibility

Our application should be able to run on Windows, Mac, and Linux. It should be able to run on any system with more than 2GB of memory and a multi-core processor.

4 Sequence Diagram



5 Use Cases

5.1 Element Selection

Description: User selects an element.
Events: On the user click, a mouse event is created which is registered by the application. The button then reacts to the event and changes state. Finally, the game board is updated to have the current element that the user has specified.

5.2 Particle Creation

Description: User creates particles on the game board.
Events: When the user clicks and drags on the game board, mouse events will be continuously sent to the game board. These events tell the game board to create new particles where the user's mouse is.
Exceptions: If the specific pixel is already occupied by another particle, then the game board will not overwrite with the user's element.