Tat Tanprasert 6710545636

# ByteFactory

(name pending)

## 1. Project Overview

A factory building roguelike automation game. At its core it is a game about producing the largest numbers possible. And levels with exponentially higher number goals must be met using your limited factory space.

## 2. Project Review

The game is reminiscent of “Factorio” and other factory-like games. However, unlike these games that have incremental progress. ByteFactory will be about completing levels in a way similar to the roguelike genre, using limited space with gameplay focusing only on design and reorganization. The game is also reduced to only 1 resource being “numbers” for simplicity.

## 3. Programming Development

### 3.1 Game Concept

The game operates on a small grid system you have to build on. “Generators” will make numbers and then “machines” will apply operations like multiplication, with traversal using conveyor systems, each having their own size required in the grid. The game will provide a random assortment of machines in each level like a roguelike game (still unsure about specifics). A number goal must be reached to complete the level with a higher goal in the next. However, there is only limited space to place down machines, so you must design layouts carefully. Machines should also be configurable with input/output sides to meet your design needs. The selling point of the game is for those who like the intricate designing mechanics of factory-like games combined with the randomness of roguelike games.

### 3.2 Object-Oriented Programming Implementation

Classes that should be used in mechanics include:

**“Byte” class**

Representing the number objects in the game to go on conveyors and be used in machines. With attributes like value and position. And methods like a collision check with other numbers.

**“Conveyor” class**

The conveyors that will move numbers in the game. Attributes like direction and length. Methods like one for moving all numbers on itself.

**“Machine” class**

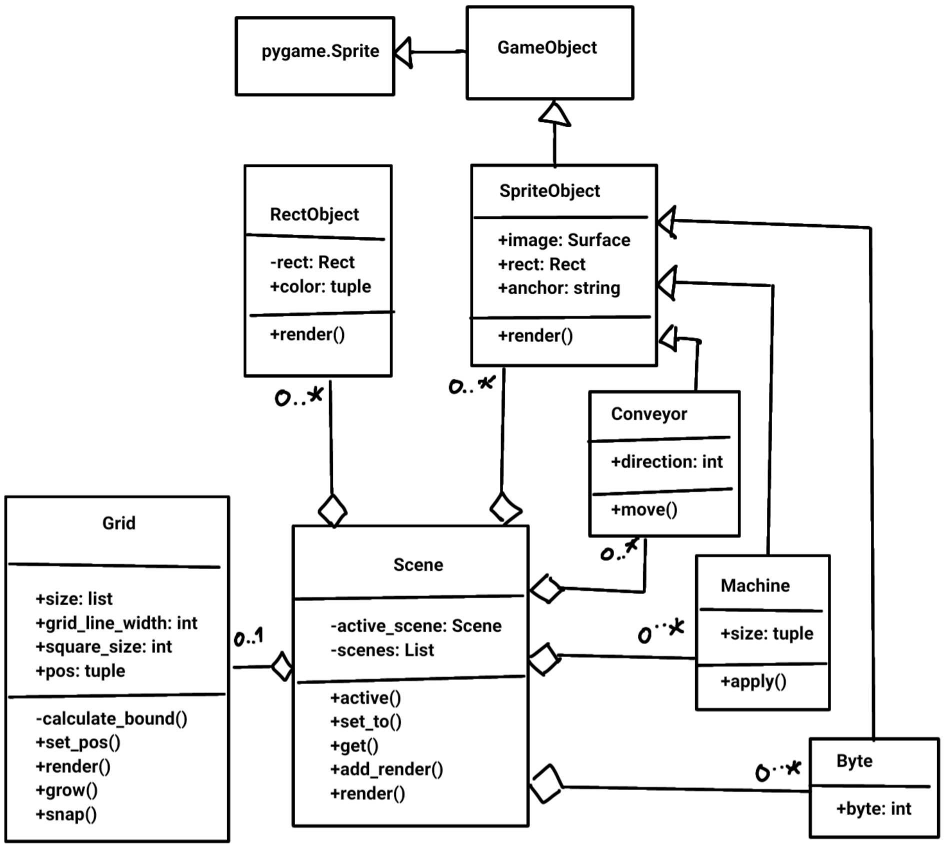
The machines in the game, used for modifying numbers. Attributes like working speed and its operation function. Methods for modifying, inputting, and outputting numbers.

**“Scene” class**

Being the scene for use in the game. Attributes like objects in
scene. Methods like rendering the scene.

**“Grid” class**

Being the grid all objects will go on. Attributes like size and square size. Methods like grow and snap.



(basic general diagram)

There will also be other classes like: GameObject.
Other classes for ease like Display and Text should also be present.

## 3.3 Algorithms Involved

The movement of number objects in the game will be logic based, using the machines. Such as outputting in different directions if a number is greater or smaller than a specified value. The game itself is basically building your own algorithm using provided logical operators. There is no inherent complicated algorithm use though.

## 4. Statistical Data (Prop Stats)

## 4.1 Data Features

Metrics tracked in each level such as:

- Total Numbers produced
- Highest Number produced
- Average Number value produced
- number of Numbers produced at different value levels
- Number production rate
- Average Number bottleneck time
- Average Number time spent on conveyor percentage
- Goal completion rate
- number of rerolls

| | Why have this data | How to get 50 values | Which variable to collect from | How it will be displayed |
|---|---|---|---|---|
| Total Numbers produced | Shows if game progression is working as intended | Play game 5 times | Scene class component variable | Line graph |
| Highest Number produced | Shows strategy reliance on largest number | Play game 5 times | Scene class component variable | Line graph |
| Average number bottleneck time | Shows players how efficient their design is | Play game 5 times | Byte class stop time | HIstogram |
| Goal completion rate | Seeing this over multiple levels can show issues in player's strategy or the game's balance | Play game 5 times | Scene class component variable | Line graph |
| Average number produced | Shows player how spread out their current design is | Play game 5 times | Byte class value | Line graph |

## <u>4.2 Data Recording Method</u>

Data will be stored in a JSON file.

## <u>4.3 Data Analysis Report</u>

Stats related to factory efficiency will be displayed for the player. A graph will display how values changed over each level

## 5. Project Timeline

| Week | Task |
|---|---|
| 1 (10 March) | Proposal submission / Project initiation |
| 2 (17 March) | Full proposal submission |
| 3 (24 March) | Basic game structure systems |
| 4 (31 March) | More game details and data collection |
| 5 (7 April) | Possible improvement and polish |
| 6 (14 April) | Submission week (Draft) |

Milestone goals:
16 April 50%: complete core game systems
23 April 75%: complete additional game systems
11 May 100%: make additional design choices for completion

## 6. Document version

Version: 5.0
Date: 16 April 2025

| **Date** | **Name** | **Description of Revision, Feedback, Comments** |
|---|---|---|
| 15/3 | Phiranath | Don’t forget to change the italic fonts. The overall concept is good. Make sure you can finish this game before the final submission. 👍 |
| 16/3 | Rattapoom | No further comments 👍 |
| 28/3 | Phiranath | I have little to comment on, but I was wondering whether the data feature will collect only five out of the nine features proposed. 🤔 |