Name\_Tyler Gehring \_\_\_\_\_\_\_\_\_\_\_\_\_ Mark \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/50

## Brief introduction \_\_/3

My Feature for the SubTerra Project will be creating a non-reusable Item base class along with several non-reusable item objects that inherit from the base class (tnt, batteries, food, ect). Along with the class structure, I will be creating the unity objects and visual functionality. The “Base Tool Class” Issue: <https://github.com/tylergehring/SubTerra/issues/4> and the “Adding Tools” issue <https://github.com/tylergehring/SubTerra/issues/5> Highlight some of the acceptance criteria (although it is general and the tool class structure has been separated into two base classes – reusable and non-reusable items).

## Use case diagram with scenario \_\_14

### Use Case Diagrams

A diagram of a tool system

AI-generated content may be incorrect.

### Scenarios

* **Name:** Collect Tool

**Summary:** The player grabs a tool

**Actors:** Player

**Preconditions:** There exists a tool object in the cave

**Basic sequence:**

**Step 1:** The player goes to collect the tool

**Step 2:** Identify that the tool is in front of them and is still usable

**Step 3:** Add the tool to the player data

**Exceptions:**

**Step 1:** the tool isn’t close enough to the player

**Step 2:** the tool isn’t usable

**Post conditions:** The tool object has successfully been added to the player’s data

**Priority:** 3\*

**ID:** Issue 4 and 5 on [github](https://github.com/tylergehring/SubTerra/issues)

* **Name:** Use TNT Tool

**Summary:** The player uses the TNT Tool

**Actors:** Player

**Preconditions:** There exists at least one TNT object in the player’s data

**Basic sequence:**

**Step 1:** The player goes to use the TNT

**Step 2:** Identify that the player has a TNT object

**Step 3:** Identify the surrounding area of the player

**Step 4:** Destroy the surrounding area of the cave

**Step 5:** Remove the TNT object from the player’s data

**Exceptions:**

**Step 1:** the player doesn’t contain a TNT object

**Step 2:** the player is by a part of the cave that can’t be destroyed

**Post conditions:** The TNT object has successfully been used and the surrounding cave has been destroyed.

**Priority:** 3\*

**ID:** Issue 4 and 5 on [github](https://github.com/tylergehring/SubTerra/issues)

* **Name:** Use Food Tool

**Summary:** The player uses the Food Tool

**Actors:** Player

**Preconditions:** There exists at least one Food object in the player’s data

**Basic sequence:**

**Step 1:** The player goes to eat the Food object

**Step 2:** Identify that the player has a Food object

**Step 3:** Identify the players health

**Step 4:** Increase the health of the player

**Step 5:** Remove the Food object from the player’s data

**Exceptions:**

**Step 1:** the player doesn’t contain a Food object

**Step 2:** the player already has full health

**Post conditions:** The Food object has successfully been used and the player’s health has increased.

**Priority:** 3\*

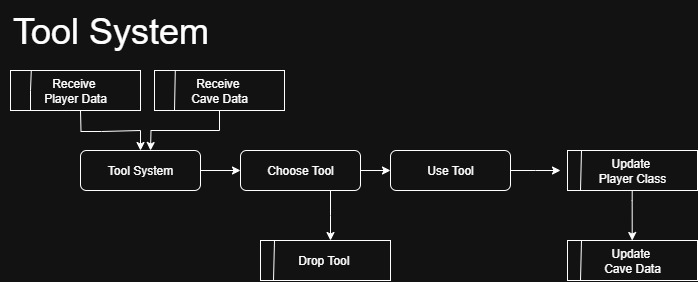
**ID:** Issue 4 and 5 on [github](https://github.com/tylergehring/SubTerra/issues)

## Data Flow diagram(s) from Level 0 to process description for your feature \_\_\_\_\_\_\_14

### Data Flow Diagrams

A diagram of a game

AI-generated content may be incorrect.

****

### Process Descriptions

Tool System:

The tool system will be designed using an object-oriented approach with two primary base classes, representing the two categories of tools: reusable and non-reusable. All tool objects will inherit from these base classes, overriding core methods as needed to satisfy their specific functional requirements.

For the **non-reusable tool base class** (the primary focus of this champion), the design will allow ingestion of relevant data from both the player class (e.g., health status, available tool inventory) and the surrounding environment (e.g., cave structures, tool locations). Using this data, a player can activate a tool instance by invoking its dedicated use() function. This function will be implemented within each specific tool subclass, ensuring that its behavior modifies the previously ingested data in a tool-appropriate manner.

Upon successful execution of the use() function, a post-processing step will update the player and environment data stores with the resulting changes. This architecture supports modularity, flexibility, and repeatability: the system can be thoroughly tested by varying input parameters and observing the resulting outputs across different tool types.

### Acceptance Tests \_\_\_\_\_ 9 :

**Test Non-Reusable Item Feature**

To validate the functionality of the **NonReusable Item** feature, we will conduct a series of controlled tests using mocked player and cave data. The test will repeatedly generate random data within realistic bounds to simulate a variety of gameplay conditions.

**Test Steps:**

* Generate mock player data (e.g., health, inventory contents, status effects).
* Generate mock cave/environment data (e.g., cave structures, resource nodes).
* Instantiate a non-reusable tool object and execute its use() function with the mocked inputs.
* Capture the resulting outputs and write them to an output file.
* Compare the observed results against expected outcomes to verify correct functionality.

**Expected Output Characteristics:**

* Player state is updated appropriately (e.g., health reduced, inventory item consumed).
* Cave/environment data reflects any intended changes (e.g. altered cave structure).
* No references to the consumed tool remain in the player’s inventory.
* All updates are written correctly to the output file with no data loss or corruption.
* System remains stable under repeated test runs with varied input parameters.

## Timeline \_\_\_\_\_\_\_\_\_/10

### Work items

|  |  |  |
| --- | --- | --- |
| Task | Duration (PWks) | Predecessor Task(s) |
| 1. Implement NonReusable Class | 6 | - |
| 2. Add Tool to Unity | 2 | 1 |
| 3. Visual Design | 2 | 1,2 |
| 4. Sound | 2 | 1,2 |
| 5. Testing | 3 | 1,2,4 |
| 6. Documentation | 1 | 5 |
|  |  |  |
|  |  |  |

### Pert diagram

### Gantt timeline

A grid with yellow squares

AI-generated content may be incorrect.