Name: Jack Kroll Mark: /50

**1. Brief introduction \_\_/3**

My feature for SubTerra is to code the reusable player tools class. The tools will be used by the player and will consist of, but not be limited to, the following. A pickax that can destroy walls, a weapon of some kind that can inflict damage, and a flashlight.

My job is to make and help integrate these tools into the generated game levels. The items will be generated with each level in random locations where the player can then acquire them and use them to win the level. My responsibility is to make these reusable Items in Unity and to test them for errors. I will also be responsible for making sure the items interact with the environment as intended.

# Use case diagram with scenario \_\_14

## Use Case Diagrams

1#

Player

Togging inventory

<<extend>>

<<include>>

<<extend>>

Player Uses Flash

Light.

Players Screen is

Illuminated.

Select Flashlight

2#

Player

Togging inventory

Player Uses/Holds

Pickaxe

<<extend>>

<<extend>>

Select Pickaxe

## Scenarios

**Scenario 1 (1st Use Case Diagram)**

**Name:** Illuminate Player Screen

**Summary:** The Player toggles their inventory and then selects Flashlight. Because the player will always have a flashlight in their inventory, they will automatically be able to hold/use it in all cases. However, the flashlight may not illuminate the player's screen if its battery level is 0.

**Actors:** Player

**Preconditions:** The player wants to execute the commands to illuminate the screen or use/hold the flashlight.

**Basic sequence:**

**Step 1:** Player presses the key to toggle “T”, for instance.

**Step 2:** The base class options for the reusable and non-reusable tools will appear in the form of a key “F” for flashlight.

**Step 3:** The player selects flashlight.

**Step 4:** The flashlight will be used/held by the player.

**Step 5:** The player’s screen is illuminated.

**Exceptions:**

**1st Exception:**

**Step 1:** The flashlight has no more battery left.

**Step 2:** The player's screen is not illuminated.

**2nd Exception:**

**Step 1:** The player doesn’t select the flashlight.

**Step 2:** The player's screen is not illuminated.

**Post conditions:** The player’s screen is illuminated.

**Priority:** 2\*

**ID:** J01

**Scenario 2 (2nd Use Case Diagram)**

**Name:** Using/Holding tools that you don't start with. This case example is (Pickaxe).

**Summary:** The Player toggles their inventory and then selects a pickaxe. Because the player will not always have a pickaxe in their inventory, they either will be able to use/hold it, or if they don’t have it, they won't be able to use it.

**Actors:** Player

**Preconditions:** The player wants to execute the commands to use the pickaxe (tool).

**Basic sequence:**

**Step 1:** Player presses the key to toggle “T”, for instance.

**Step 2:** The base class options for the reusable and non-reusable tools will appear in the form of a key “P” for pickaxe.

**Step 3:** The player selects the pickaxe option.

**Step 4:** The pickaxe will be used/held.

**Exceptions:**

**1st Exception:**

**Step 1:** The Player does not have the pickaxe in their inventory.

**Step 2:** The player doesn’t use/hold the pickaxe.

**2nd Exception:**

**Step 1:** The player doesn’t select the pickaxe.

**Step 2:** The player doesn’t use/hold the pickaxe.

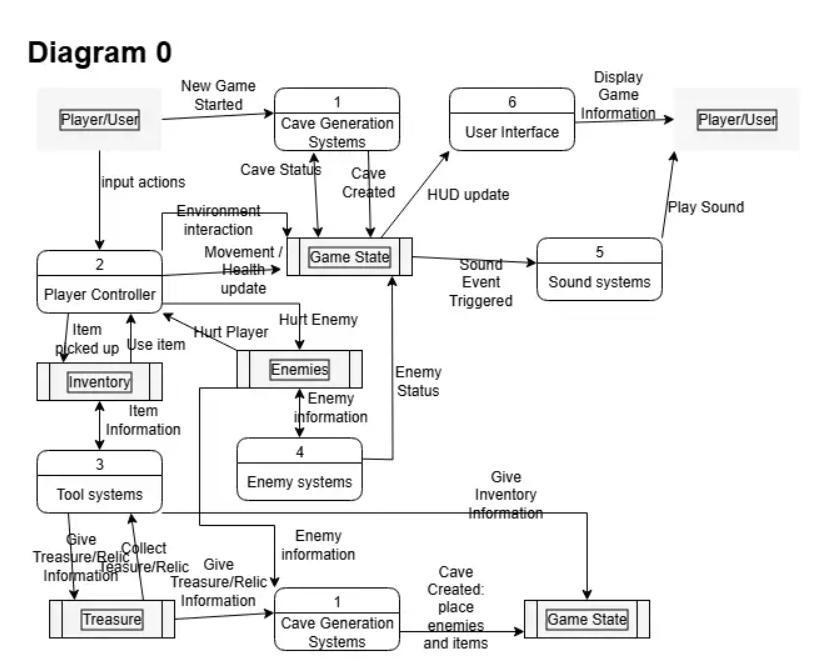
**Post conditions:** The pickaxe will be used/held.

**Priority:** 2\*

**ID:** J02

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

## Data Flow Diagrams



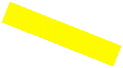


Diagram for the Tool Systems.

3

Tool Systems/Manager

Giving Tool Attributes

Asking for Tool Info and how to use them.

3.1

Base Tool Class

Giving Tool Attributes

Giving Tool Attributes

Getting Tool Attributes.

3.3

Non-reusable Tool Class

3.2

Reusable Tool Class

Diagram for Reusable Tool Class.

3.2

Reusable Tool Class

Defining Tool Attributes

Defining Tool Attributes

Defining Tool Attributes

Pickaxe Attributes

Flashlight Attributes

Other tool Attributes



**Process Descriptions**

Decision tree

Player triggers tool system manager?

│

|── No -> Do nothing

│

└── Yes ->

Tool available in the player's Inventory?

│

|── No -> Display message “You don’t have that tool” and end getting attributes.

│

|── Yes ->

Get attributes from the correct class (reusable or non-reusable)

│

|── No -> Display Error message to player.

│

|── Yes ->

Select the correct tool in the reusable player class.

│

|

| Flashlight battery = 0?

│ |── Yes -> Set Illumination attribute to ‘false’.

│ |── No -> Set Illumination attribute to ‘true’

│

Object updated successfully?

|── Yes -> Update object state.

|--─ No -> Display an error message

How this works in a few steps.

1st Step: The tool system manager is triggered by the player to get the attributes/data set of a certain tool.

2nd Step: The attributes/data are acquired from the (reusable tool class).

3rd Step: Those attributes are then used in the game on the visible objects that the player can see.

Note: The data for the tool and how it works will be organized into 2 classes that both inherit from the base class.

# Acceptance Tests \_\_\_\_\_\_\_\_9

To test the reusable tools, I will perform several tests in a controlled environment to ensure the tools behave as expected and no errors occur during gameplay. I will do this by utilizing Unity's Test resources and creating a mock environment where I can put in a bunch of random values/circumstances to different tools to see if there are any errors.

I will test the Pickax by giving it random wall types, seeing if there are any errors with it being used at a specific speed or location. Then log the results.

For the Flashlight, I will test its functionality by inputting random player commands to make sure there are no input handling errors that affect my class. Then log the results.

For a weapon or other tool, I will also test input handling by the user in a mock environment, simulating the game and logging the results of random input and positioning.

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

## Work items

|  |  |  |
| --- | --- | --- |
| Task | Duration (PWks) | Predecessor Task(s) |
| 1. Planning and preparation for Tools integration and design. | 3 | - |
| 2. Base class structure programming. | 2 | 1 |
| 3. Reusable Tools Class programming. | 2 | 2 |
| 4. Add class to Unity. | 2 | 2, 3 |
| 5. Add Visual design. | 1 | 2, 3 |
| 6. Add Sound | 1 | 4,5 |
| 7. Documentation | 2 | 6 |
| 8. Final Testing and Integration | 3 | 7 |

## Pert diagram

A diagram of a network

AI-generated content may be incorrect.

## Gantt timeline

Key

Duration (PWks) With no slack.



Slak.



Tasks

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | 2 |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | 2,3 |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | 2,3 |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | 4,5 |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  | 6 |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  | 7 |  |  |

1

2

3

4

5

6

7

8

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Duration (PWks)