Name\_\_\_\_Dylan Card\_\_\_\_\_\_\_\_\_\_ Mark \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/50

[**Instructions**: Remove everything that is not a heading below and fill in with your own diagrams, etc.]

## Brief introduction \_\_/3

I will oversee all the menus’ features and the error handling system. This includes the starting menu of the game, level selection menus for replay-ability, inventory menus, and general settings menus. Each menu will have several options that can be selected each of which will have a corresponding effect on the game world. The in-game menu will also have the ability to handle errors such as the player becoming stuck or the correct items not being available or lost for some reason.

## Use case diagram with scenario \_\_14

[Use the lecture notes in class. Ensure you have at least one exception case]

Example:

### Use Case Diagrams

### Scenarios

**[You will need a scenario for each use case]**

**Name:** Start Game

**Summary:** The player uses the menu to start the game.

**Actors:** Player.

**Preconditions:** Game has been initialized.

**Basic sequence:**

**Step 1:** Display button to be selected.

**Step 2:** Accept input of a single [Start] button.

**Step 3:** Begin game from the beginning.

**Exceptions:**

**Step 1:** A button other than [Start] is selected: Either open the [Level Selection] menu and begin the game from the selected level or open the [Help] menu depending on the button selected.

**Step 2:** The [Exit] button is selected: exit game.

**Post conditions:** The game has begun.

**Priority:** 1

**ID:** M01

### Scenarios

**[You will need a scenario for each use case]**

**Name:** Access/View Inventory

**Summary:** The player views and can use items stored in an inventory.

**Actors:** Player.

**Preconditions:** Game has been initialized and play has begun.

**Basic sequence:**

**Step 1:** Display [Inventory] button to be selected.

**Step 2:** Player selects the inventory button.

**Step 3:** Display the inventory of the player.

**Step 4:** Give the player the option of using an item.

**Step 5:** Use item and close menu.

**Exceptions:**

**Step 1:** The player selects [Close] and does not use an item: close inventory menu.

**Post conditions:** An item has been used by the player.

**Priority:** 2

**ID:** M02

### Scenarios

**[You will need a scenario for each use case]**

**Name:** Error Handling

**Summary:** The player notices an error in the game that is then resolved by the error handler selected from the in-game menu.

**Actors:** Player.

**Preconditions:** Game has been initialized, play had begun, and an error has occurred.

**Basic sequence:**

**Step 1:** Display button to be selected.

**Step 2:** Player notices an error and selects the [Error] button.

**Step 3:** Accept input describing the error.

**Step 4:** Resolve the error in the most efficient manner.

**Exceptions:**

**Step 1:** The player selects that there is an error when there is not: repeat the error fix as if there were an error.

**Step 2:** The error is outside the scope of the error handler: record the progress of the player and reinitialize the game to a point in the players progress just before the error.

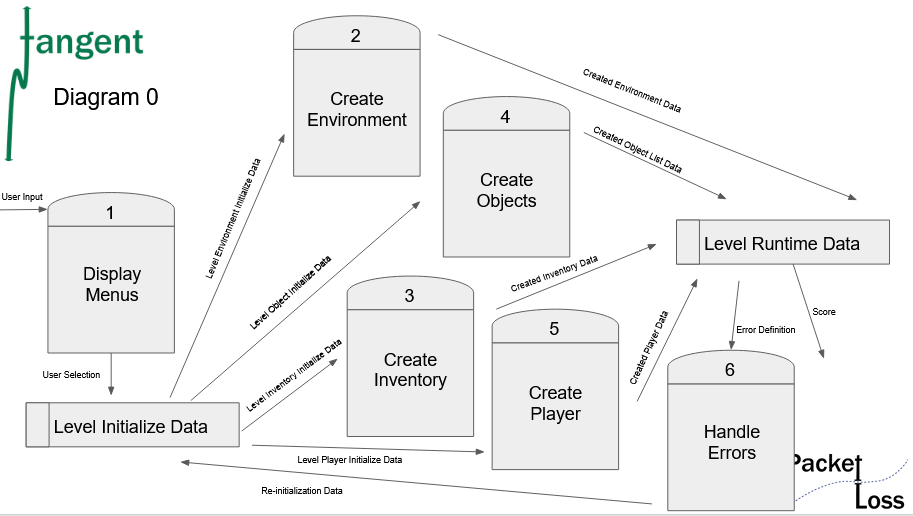
**Post conditions:** The error has been resolved.

**Priority:** 3

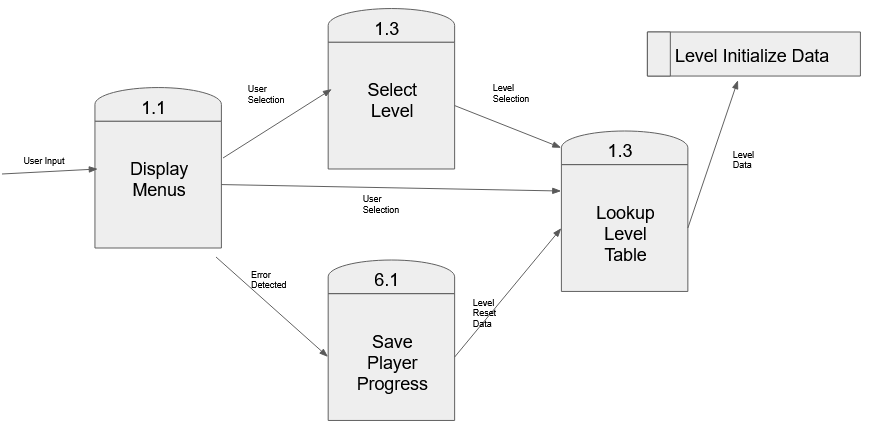
**ID:** M03

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

[Get the Level 0 from your team. Highlight the path to your feature]

Diagram 0 

### Data Flow Diagrams



### Process Descriptions

2

Display Menus:

Accept player menu selection

If player selection ( MAIN) //application started or reset

Display MAIN

If player input is HELP\_MENU

Display HELP\_MENU

If player input is not start game

Call Select Level

Elseif player selects start game call Lookup Level Table for lvl 1

Elseif game is running and player select menu

Display GAME\_MENU

If player selection is help display HELP\_MENU

If player selection is error call SavePlayerProgress

Else display HELP\_MENU

Exit help menu when player selects EXIT\_HLEP\_MENU

Elseif player inputs exit

Application.Quit();

Select Level:

Wait for player input

While player selects a level other than one

Prompt for the level passcode

If Passcode is correct call Lookup Level Table for selected level

Else Prompt for another selection

Save Player Progress:

Save player inventory matrix

Save level objects, problems, and solutions matrices

Call lookup level table with the saved play data to reinitialize the level

Lookup Level Table:

If called with no additional input

Lookup level one initialization data

Store data to call initialization functions

If called with specific level input

Lookup specified level initialization data

Store data to be called by initialization functions

If called with specific level input and level matrices

Do not access the level table

Send the given information to the level initialization data store

This will cause the level to be initialized with the same flagged problems and solutions as well as the player progress to that point

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

[Describe the inputs and outputs of the tests you will run. Ensure you cover all the boundary cases.]

A script will be written to call each of these processes individually. Each process will be tested 100 times. The script will then check to see if the process for which it was designed to test has successfully completed its function. If the process acted correctly then the script will record a success in an output file. If the process was not successful then the script will record a failure in the output file. This script will be modified to run in an encapsulated environment designed to test each process individually. The processes will be accepted if the number of failures in each output file is > 4.

An additional script will be written to check that the save player progress function is working properly. This script will create random numbers of objects that would not normally need to be saved and will randomize the objects that appear, such that problems will not always have answers available, to test how the process can handle unexpected cases. If the process if able to successfully store the intended matrices, based on the problems present table, and send the correct data to re-instance the level it will be considered a success.

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

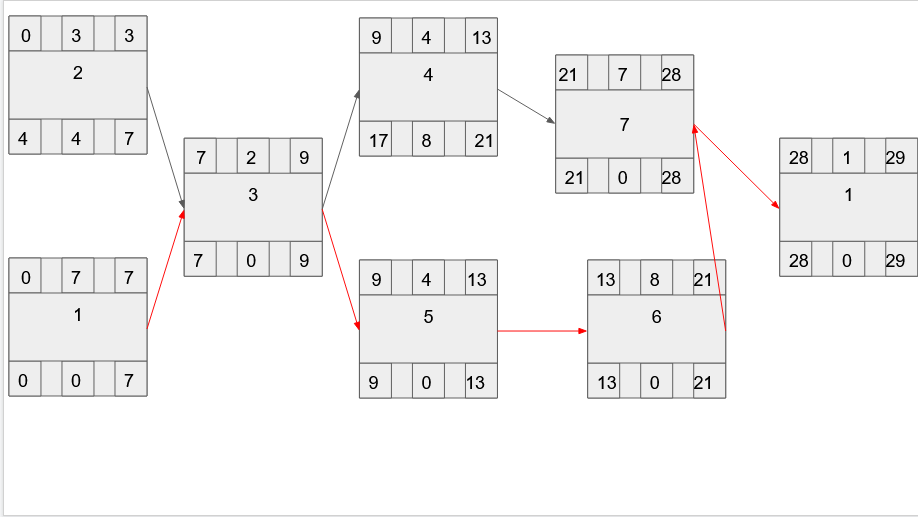
[Figure out the tasks required to complete your feature]

Example:

### Work items

|  |  |  |
| --- | --- | --- |
| Task | Duration (Hrs) | Predecessor Task(s) |
| 1. Design Menu Functionality | 7 | - |
| 2. Create Menu Visual Design | 3 | - |
| 3. Implement Start Menu Design | 2 | 1,2 |
| 4. Implement Level Selector | 4 | 1,2,3 |
| 5. Implement Inventory Menu | 4 | 1,2,3 |
| 6. Design Error Handler | 8 | 2,3,5 |
| 7. Testing | 7 | 3,4,5,6 |
| 8. Installation | 1 | 3,4,5,6,7 |

### Pert diagram



### Gantt timeline

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  | 1,2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  | 1,2,3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  | 1,2,3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |  |  |  | 2,3,5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 3,4,5,6 |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 7 |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |  |  |  |  |  |