Name: Ferris Hammes-Buehler Mark \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/50

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

## Brief introduction \_\_/3

My feature is the player character, known in-game as the “Hero”. When starting an ascension, the user chooses one of three Hero classes for that run. There will be at least three different Hero classes, those being “Fighter”, who has a shield-bash, “Rogue”, who can climb metal pipes, and “Wizard” who has a ranged attack.

The player character needs to be able to move laterally, jump, attack, and use their ability.

## Use case diagram with scenario \_\_14A diagram of a game Description automatically generated

### Scenarios

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

**Name:** Attack

**Summary:** The user uses the player controller to attack.

**Actors:** User.

**Preconditions:** Player model and player controller have been initialized; user made a valid selection in ascension setup.

**Basic sequence:**

**Step 1:** Determine what class of Hero the user selected.

**Step 2:** Begin attack animation.

**Step 3:** Create an area that detects collision with enemies.

**Step 4:** If area collides with an enemy, increase score by a set amount.

**Step 5:** Delete that area after a set amount of time.

**Step 6:** End attack animation.

**Exceptions:**

**Exception 1:** [attack] is pressed while the animation is still running:

**Step 1:** Ignore input.

**Exception 2:** [attack] is pressed while the Hero Class is Wizard:

**Step 1:** Begin attack animation.

**Step 2:** Create an area that detects collision with enemies and walls.

**Step 3:** Move that area away from the player model until it collides with an enemy or wall. If it collided with an enemy, increase score by a set amount.

**Step 4:** Delete that area.

**Post conditions:** Calculated value is displayed.

**Priority:** 2

**ID:** C01

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

### Data Flow Diagrams

A diagram of a company

Description automatically generated

### A diagram of a computer program Description automatically generated

### Process Descriptions

Instantiate Player Scene:

Character controller default moved into main scene.

Attach Character Controls:

FOR EACH control in Settings

IF control has a key bound to it

Key pressed -> action

END IF

END FOR EACH

Attach Class Abilities:

IF user selection is valid

FOR EACH control in settings -> class specific

IF control has a key bound to it

Key pressed -> action

END IF

END FOR EACH

Player controller animations / model set to correct objects for that class.

END IF

Place Character in Level:

WHILE player location is NOT grounded AND location is NOT part of the terrain

Randomly reassign location between a set X and a set Y (handled in level pieces).

END WHILE

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

To determine whether this function works correctly, we run the following test:

* Input:
  + Class selection: random integer between 1 and the number of options.
  + Key bindings: set to random inputs.
  + Level section: a small area. Enemy to far left, enemy to close left. Barricade to right, pipe above.
* Press buttons in the following sequence:
  + Attack left
  + Attack left
  + Shield bash right
  + Jump
  + Climb
* Output the position of the player, as well as the position of any enemies and integrity of the barricade.
  + If the controller is a Fighter, the barricade should be broken, the close enemy should be dead, and the player should be no higher. If so, success! If not, fail.
  + If the controller is a Rogue, the barricade should be intact, the close enemy should be dead, and the player should be at the top of the pipe. If so, success! If not, fail.
  + If the controller is a Wizard, the barricade should be intact, both enemies should be dead, and the player should be no higher. If so, success! If not, fail.

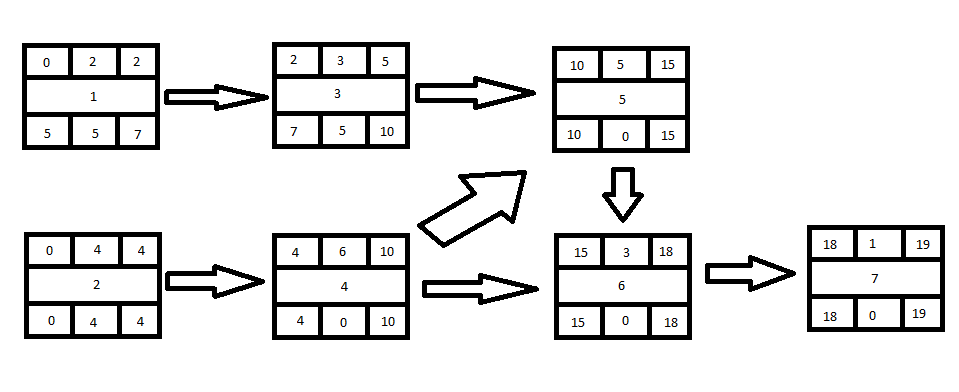
The number of times we run this test increases the likelihood of catching an error.

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

### Work items

|  |  |  |
| --- | --- | --- |
| Task | Duration (PWks) | Predecessor Task(s) |
| 1. Requirements Collection | 2 | - |
| 2. Sprite Design | 4 | - |
| 3. Controller Design | 3 | 1 |
| 4. Level Creation | 6 | 2 |
| 5. Programming | 5 | 3, 4 |
| 6. Testing | 3 | 4, 5 |
| 7. Installation | 1 | 6 |

### Pert diagram



### Gantt timeline

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