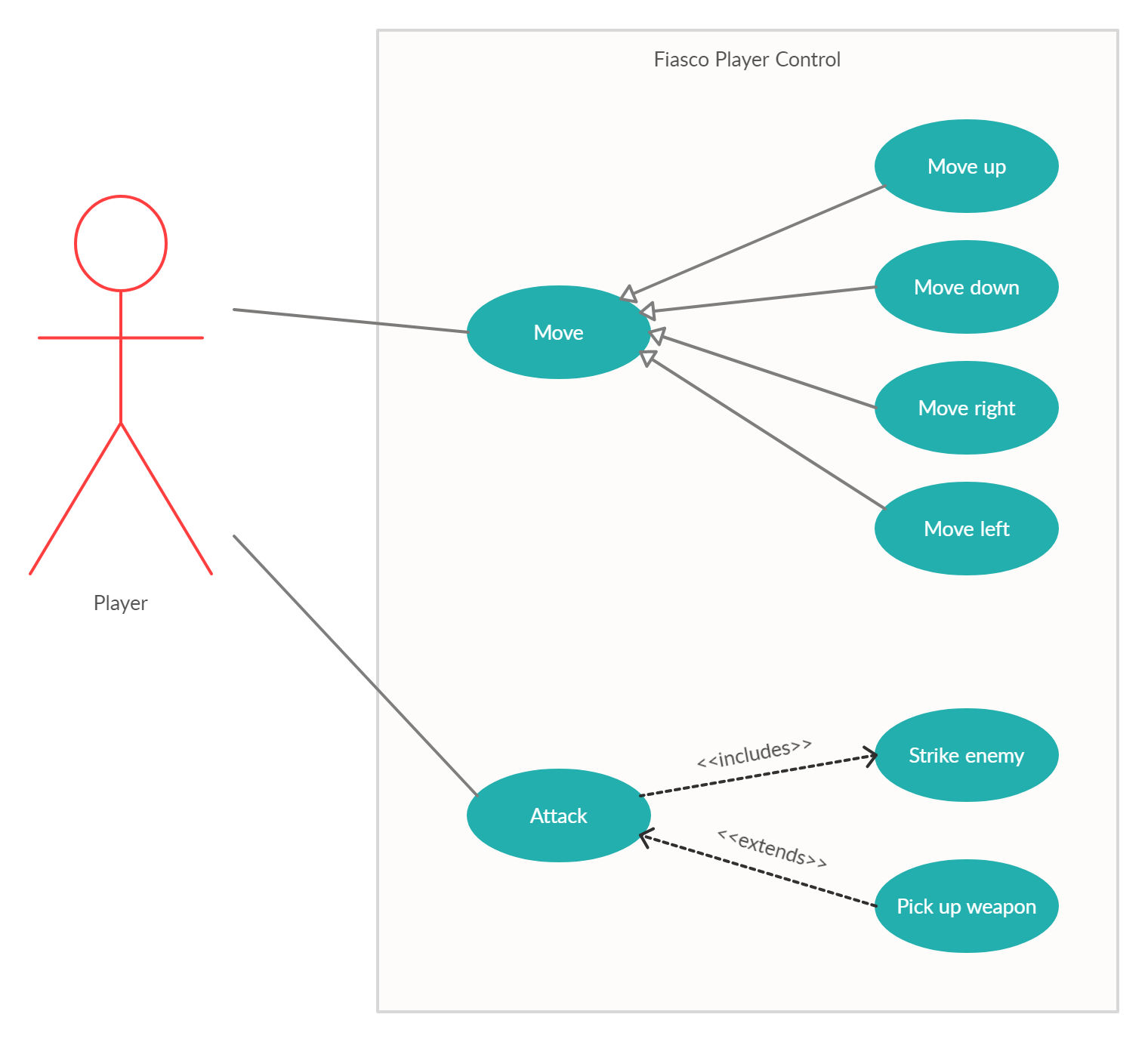
Name: Winston Oswald-Drummond Mark \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/50

## Brief introduction \_\_/3

I’m Team Lead #4 for this project, and I’ll be handling player movement and attacking (specifically when the player attacks enemies, not the reverse). Our game, Fiasco, is a top-down 2D story/adventure game. This means there will be no jumping. For movement, all the player will be able to do is move up, down, left, and right. I plan on making this happen when the user presses w, s, a, or d respectively on their keyboard. I plan on dealing with collisions (between players and walls or enemies) by setting up “rigidbodies” on Unity. As for attacking, the user will be able to do this by pressing another keyboard button (maybe r), so long as an enemy is in range. When the button is pressed, assuming an enemy is in range, the currently equipped weapon will be used to deal damage. Also, if the enemy is killed and they drop a weapon, the player will pick that weapon up.

## Use case diagram with scenario \_\_14

### Use Case Diagrams



### Scenarios

**Name:** Move

**Summary:** The player moves up, down, left, or right using the WASD keys.

**Actors:** Player.

**Preconditions:** Game has been initialized and user has pressed play.

**Basic sequence:**

**Step 1:** Accept keyboard input.

**Step 2:** Call less general use case based on which key was pressed (w, a, s, or d).

**Exceptions:**

**Step 1:** A key other than w, a, s, or d is pressed: Do nothing.

**Step 2:** No keys are pressed: Do nothing.

**Step 3:** Multiple opposing keys (like w and s or a and d) are pressed at the same time: Do nothing.

**Step 4:** Multiple non-opposing keys (e.g. w and a) are pressed at the same time: Call all relevant use cases.

**Post conditions:** Specific use case has been called which caused the player to move (unless something was in their way or opposing keys were pressed simultaneously).

**Priority:** 2

**ID:** WOD01

**Name:** Move up

**Summary:** The player moves up, this use case generalizes to the “Move” use case.

**Actors:** Player.

**Preconditions:** Game has been initialized, user has pressed play, and afterwards the user pressed the ‘w’ key.

**Basic sequence:**

**Step 1:** Accept keyboard input.

**Step 2:** Input has been determined to be ‘w’.

**Step 3:** Move player up one game pixel.

**Exceptions:**

**Step 1:** There is a wall in the way: Do nothing.

**Step 2:** There is an enemy in the way: Do nothing.

**Post conditions:** Player has been moved up one game pixel (unless something was in their way).

**Priority:** 2

**ID:** WOD02

**Name:** Move down

**Summary:** The player moves down, this use case generalizes to the “Move” use case.

**Actors:** Player.

**Preconditions:** Game has been initialized, user has pressed play, and afterwards the user pressed the ‘s’ key.

**Basic sequence:**

**Step 1:** Accept keyboard input.

**Step 2:** Input has been determined to be ‘s’.

**Step 3:** Move player down one game pixel.

**Exceptions:**

**Step 1:** There is a wall in the way: Do nothing.

**Step 2:** There is an enemy in the way: Do nothing.

**Post conditions:** Player has been moved down one game pixel (unless something was in their way).

**Priority:** 2

**ID:** WOD03

**Name:** Move right

**Summary:** The player moves right, this use case generalizes to the “Move” use case.

**Actors:** Player.

**Preconditions:** Game has been initialized, user has pressed play, and afterwards the user pressed the ‘d’ key.

**Basic sequence:**

**Step 1:** Accept keyboard input.

**Step 2:** Input has been determined to be ‘d’.

**Step 3:** Move player one game pixel to the right.

**Exceptions:**

**Step 1:** There is a wall in the way: Do nothing.

**Step 2:** There is an enemy in the way: Do nothing.

**Post conditions:** Player has been moved one game pixel to the right (unless something was in their way).

**Priority:** 2

**ID:** WOD04

**Name:** Move left

**Summary:** The player moves left, this use case generalizes to the “Move” use case.

**Actors:** Player.

**Preconditions:** Game has been initialized, user has pressed play, and afterwards the user pressed the ‘a’ key.

**Basic sequence:**

**Step 1:** Accept keyboard input.

**Step 2:** Input has been determined to be ‘a’.

**Step 3:** Move player one game pixel to the left.

**Exceptions:**

**Step 1:** There is a wall in the way: Do nothing.

**Step 2:** There is an enemy in the way: Do nothing.

**Post conditions:** Player has been moved one game pixel to the left (unless something was in their way).

**Priority:** 2

**ID:** WOD05

**Name:** Attack

**Summary:** The player damages an enemy.

**Actors:** Player.

**Preconditions:** Game has been initialized, user has pressed play, and afterwards the user pressed the attack key.

**Basic sequence:**

**Step 1:** Accept keyboard input.

**Step 2:** Make sure keyboard input is the attack key.

**Step 3:** Strike the nearest enemy.

**Exceptions:**

**Step 1:** There is no enemy within range: Do nothing.

**Step 2:** A key other than the attack key was pressed: Do nothing.

**Post conditions:** A nearby enemy has been damaged (if there was one in range of the player).

**Priority:** 2

**ID:** WOD06

**Name:** Strike enemy

**Summary:** Nearby enemy takes damage.

**Actors:** Player.

**Preconditions:** Game has been initialized, user has pressed play, and afterwards the user pressed the attack key.

**Basic sequence:**

**Step 1:** Accept keyboard input.

**Step 2:** Keyboard input has been determined to be the attack key.

**Step 3:** Decrease the health of the nearest enemy.

**Exceptions:**

**Step 1:** The health of the enemy has dropped to zero or less: Kill the enemy.

**Step 2:** There is no enemy within range: Do nothing.

**Step 3:** A key other than the attack key was pressed: Do nothing.

**Post conditions:** A nearby enemy has been damaged (if there was one w range of the player).

**Priority:** 2

**ID:** WOD07

**Name:** Pick up weapon

**Summary:** Player picks up weapon that was dropped from a slain enemy

**Actors:** Player.

**Preconditions:** Game has been initialized, user has pressed play, and afterwards the user killed an enemy that dropped a weapon.

**Basic sequence:**

**Step 1:** Accept keyboard input.

**Step 2:** Keyboard input has been determined to be the attack key.

**Step 3:** Decrease the health of the nearest enemy.

**Step 4:** Enemy has been killed.

**Step 5:** Enemy has dropped a weapon.

**Step 6:** Have the player equip the weapon.

**Exceptions:**

**Step 1:** The enemy’s health is still above 0: don’t kill enemy or drop/pick up any weapons.

**Step 2:** The enemy was killed but it didn’t drop a weapon: Inventory remains unchanged.

**Post conditions:** A nearby enemy has been damaged (if there was one in range of the player).

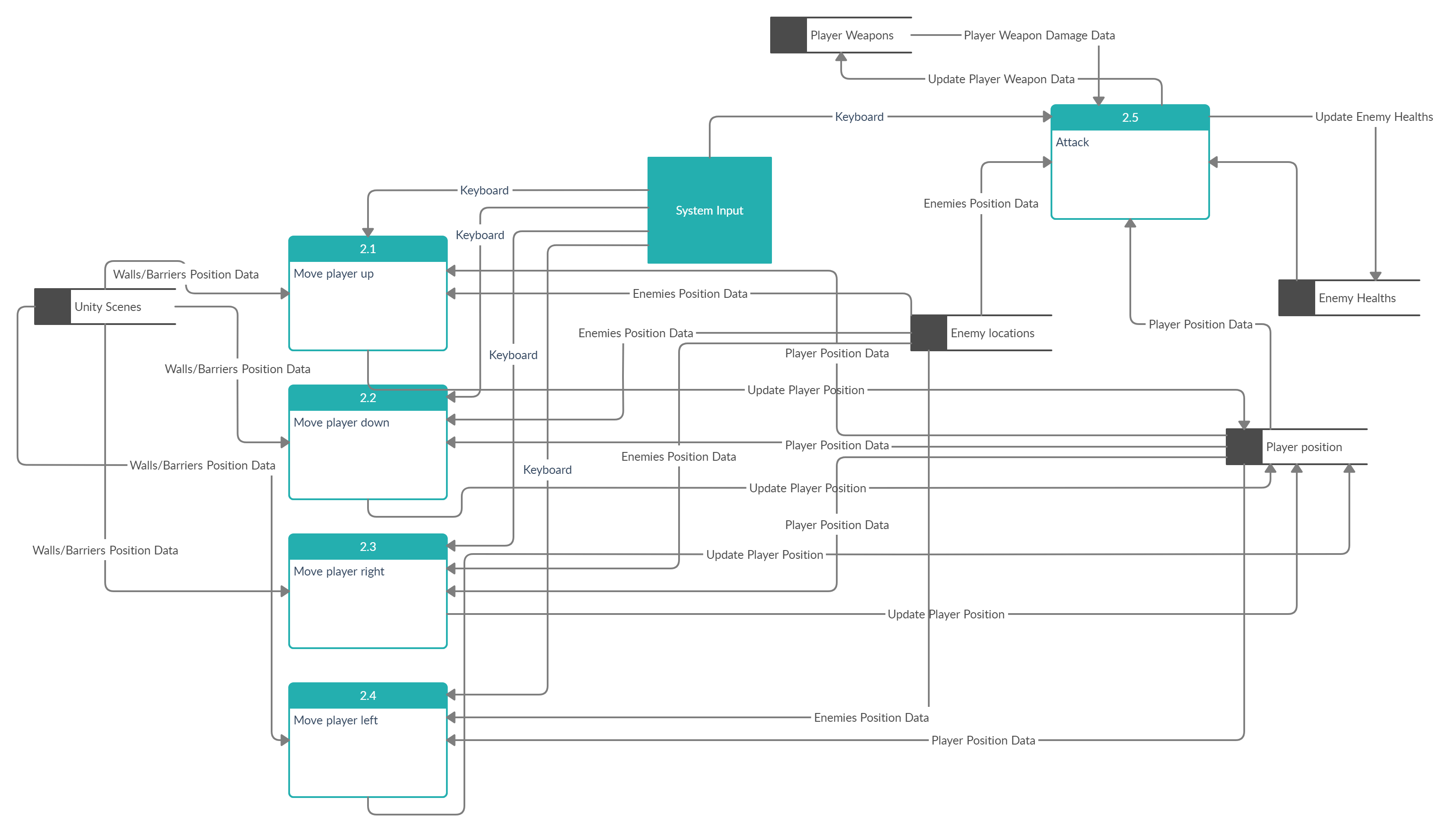
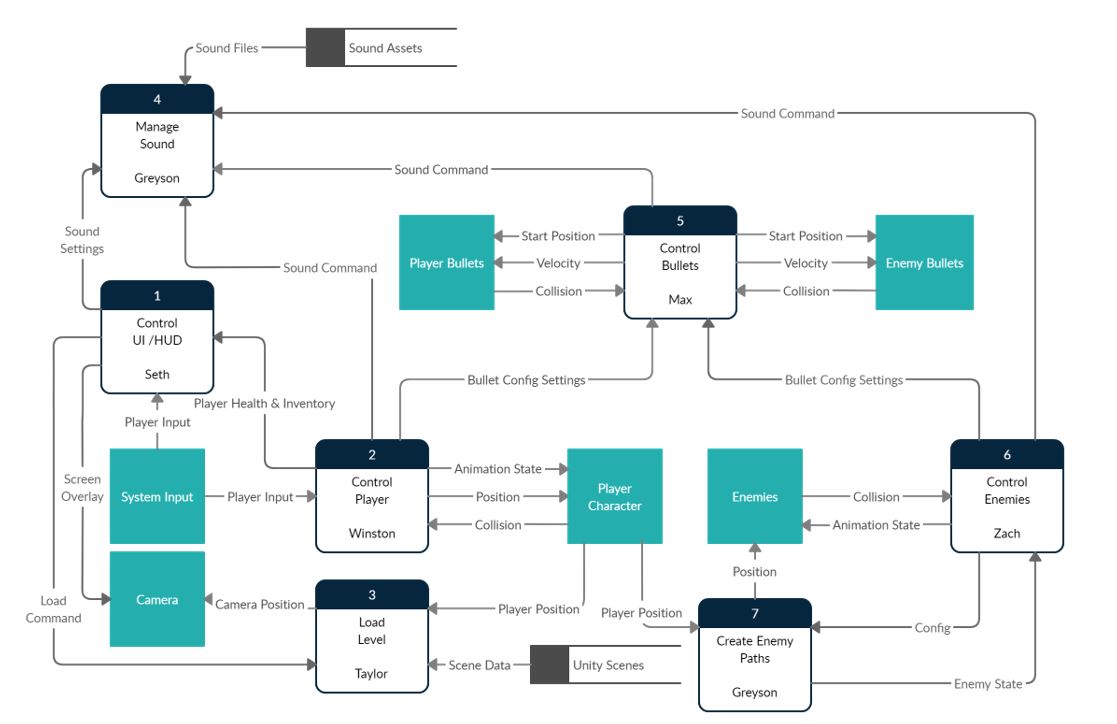
**Priority:** 3

**ID:** WOD08

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

### Data Flow Diagrams (you may need to zoom in a bit to read everything):

**Player Control - Winston**



### Process Descriptions

**Move player up:**

WHILE (in game){

IF (keyboard\_input is ‘w’) {

player\_Yposition += 1; //move up 1

//collisions will be handled using rigidbodies

}

}

**Move player down:**

WHILE (in game){

IF (keyboard\_input is ‘s’) {

player\_Yposition -= 1; //move down 1

//collisions will be handled using rigidbodies

}

}

**Move player right:**

WHILE (in game){

IF (keyboard\_input is ‘d’) {

player\_Xposition += 1; //move right 1

//collisions will be handled using rigidbodies

}

}

**Move player left:**

WHILE (in game){

IF (keyboard\_input is ‘a’) {

player\_Xposition -= 1; //move left 1

//collisions will be handled using rigidbodies

}

}

**Attack:**

WHILE (in game){

IF (keyboard\_input is attack key) {

IF (enemy is within range)}

nearest\_enemy.health -= player\_damage;

IF (enemy.health <= 0){ //if the enemy was killed

enemy\_death(nearest\_enemy);

pickup\_item(); //if item was dropped, pick it up

}

}

}

}

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

**Player movement:**

Run a script that simulates pressing random keys, excluding the keys that are meant to do something like move the player, 1000 times, and make sure nothing happens. I’ll have it return true if no move functions are called and return false otherwise.

Run a script that presses random move keys 1000 times, and make sure player movement works fine afterwards. If ever a player’s position does not move when it’s supposed to, or it moves in the wrong direction, it will return false. Otherwise, it will return true.

I’ll press different combinations of multiple move keys at the same time and check to make sure it does the right thing (nothing if opposing keys are pressed, move in a diagonal direction of only non-opposing keys are pressed).

I’ll purposefully run into walls/barriers and enemies to make sure that the player stops moving once it reaches these things.

I’ll tap each move key very quickly and check to see that it still works. I’ll then do the same but I’ll hold the keys for a very long time rather than tap them quickly.

**Attacking:**

Run a script that simulates 100 random attacks on different enemies, and then have the script check to see that the enemy healths are updates properly and that they are killed off if their health drops to zero or less. At the end, it’ll print a message telling me if the attacks were done properly or not.

I’ll test with killing both enemies that are meant to drop weapons and enemies that aren’t supposed to drop weapons. I’ll make sure that only those enemies who are meant to will drop weapons, and that the player will automatically pick those weapons up. Once things seem to be working, I’ll run a script that kills 100 random enemies and checks each time to see if an item dropped when it was supposed to and if that item was picked up. If anything ever goes wrong, it will return false. If things work properly, it will return true.

I’ll attempt to attack enemies that aren’t within range, and make sure nothing happens.

I’ll attack all enemies and make sure the correct amount of damage is being dealt.

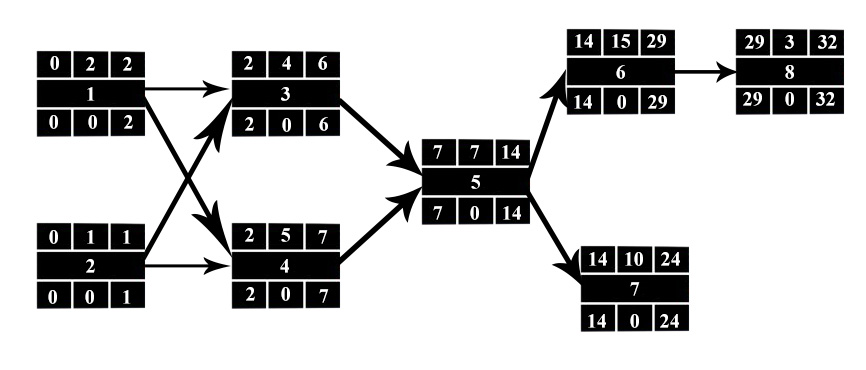
I’ll run a script that places an enemy by the player, and then simulates pressing 1000 random keys, excluding the attack key. At the end, the script will check to see if the enemy’s health has changed at all. If the enemy’s health is the same, no attacks occurred and everything worked well, so it’ll return true. Otherwise, it will return false.

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

### Work items

|  |  |  |
| --- | --- | --- |
| Task | Duration (Hours) | Predecessor Task(s) |
| 1. Requirements Collection | 2 | - |
| 2. Program Design | 1 | - |
| 3. Player Movement | 4 | 1,2 |
| 4. Player Attacking | 5 | 1,2 |
| 5. Inventory (picking up weapons) | 7 | 3,4 |
| 6. Visuals (if time permits) | 15 | 5 |
| 7. Testing | 10 | 5 |
| 8. Potentially Testing Visuals | 3 | 6 |

### Pert diagram



### Gantt timeline

