Name Katelyn “K” Atkinson Mark /50

# Brief introduction /3

My feature for the Hot Dog Jones Video Game is the main character functionality and operations. The main character is a hot dog called “Hot Dog Jones”, herein referred to as HDJ. This means I will be responsible for creating HDJ’s movement, states, and ammo.

**Task A -- Movement:** It is my responsibility to ensure HDJ interacts appropriately with the obstacles and enemies placed in our video game’s world. Since this is a top-down video game, every time HDJ enters a new part of the world map (e.g. a checkpoint in a maze), he will grow larger according to the factors laid out in Task C. HDJ must collide with obstacles and enemies in the world map, entering the idle state with the former and possibly entering death state with the latter.

**Task B -- States:** HDJ will have four states, including 1) an idle state with a bobbing motion, 2) a running state (depending on our sprite, legs may or may not be included), 3) a death state, and 4) a growing state. For each state, I will be creating a state machine with functions and objects usable from the scripts that control player-enemy interaction, ammo firing, and death/health.

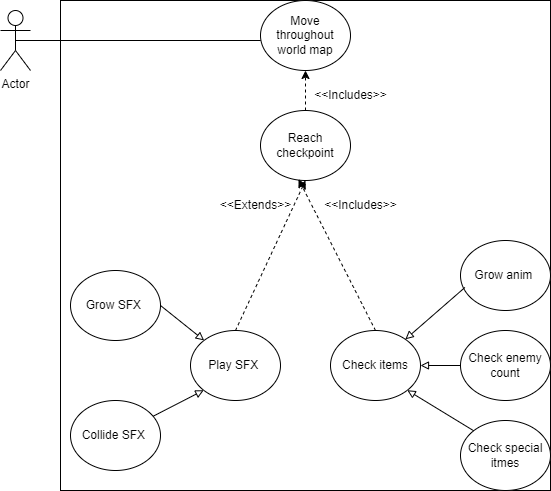
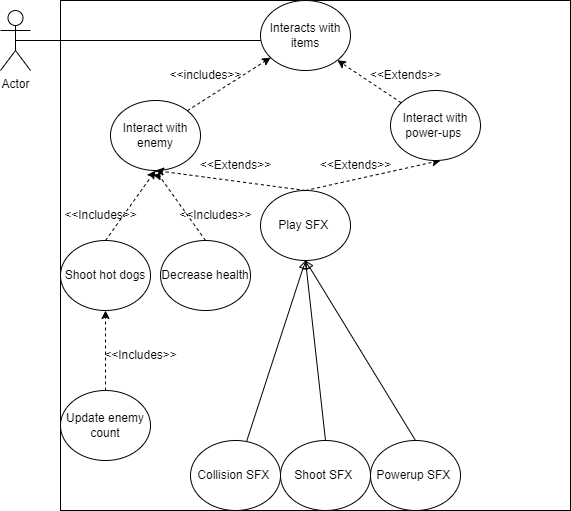
**Task C -- Ammo:** When the base level is entered upon start-up of the game, HDJ will be a generic hot dog size. To kill an enemy, he will need to shoot three hot dogs from his ammo at an enemy.

Let’s get to it.

# Use case diagram with scenario

# \_\_14

**Use Case Diagrams**

****

**Scenarios**

**Scenario 1 (1st UCD):**

**Name:** Move throughout world map.

**Summary:** HDJ moves forward, backwards, left, and right on the map, running into obstacles (no health decrease on impact) and reaching checkpoints.

**Actors:** HDJ.

**Preconditions:**

**-**Player game object has been created and added to the scene.

**-**State machine is made for player with animations.

**-**World design made with checkpoints.

**Basic sequence:**

**Step 1:** A player script control F, B, L, and R movement from arrow keys.

**Step 3:** Player moves throughout map moving around obstacles.

**Step 4:** Reaches checkpoints.

**Exceptions:**

**Step 5:** Player reaches checkpoints to grow or previous checkpoint if they died.

**Step 2:** A button other than arrow keys or space bar are pressed: ignore input.

**Post conditions:** World map HUD is updated with player location.

**Priority:** 1\*

**ID:** C02A.1

\*The priorities are 1 = must have, 2 = essential, 3 = nice to have.

**Scenario 2 (2nd UCD):**

**Name:** Interact with world objects.

**Summary:** HDJ moves through the map to fire unlimited hot dogs at enemies to grow at checkpoint.

**Actors:** HDJ, hot dog ammo, enemies, power-ups.

**Preconditions:**

**-**Enemy spawner scripts, sprites, and movement implemented.

**-**Hot dog ammo fires, collides, and vanishes enemies properly.

**-**World map has randomly placed power-ups.

**Basic sequence:**

**Step 1:** Player moves with arrows to face the enemy or power-ups.

**Step 2:** Space bar is pressed to fire stream of hot dogs.

**Step 4:** Update enemy count and health.

**Exceptions:**

**Step 5:** Player collides with enemy before killing it, resulting in decrease in health.

**Step 6:** No growth if died and sent back to previous checkpoint, enemy count lost.

**Step 3:** A button other than arrow keys or space bar are pressed: ignore input.

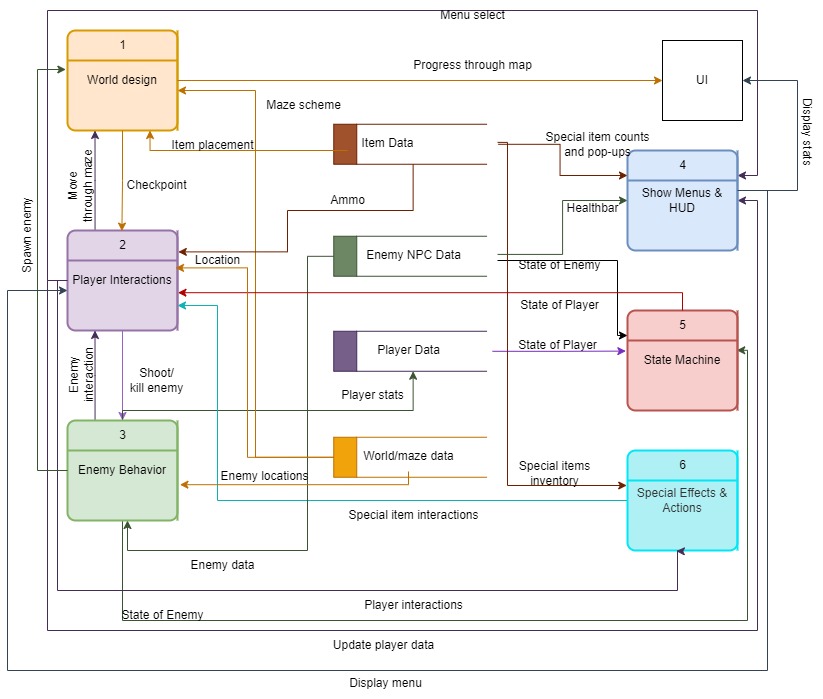
**Post conditions:** Player grows from number of enemies killed. Or the health bar decreases when hit by enemy.

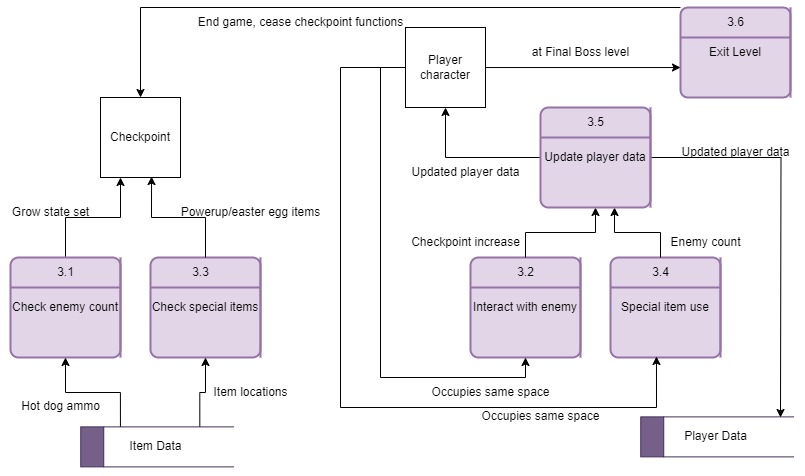
**Priority:** 2

**ID:** C02A.2

# Data Flow Diagram(s) from Level 0 to process description for your feature 14

**Data Flow Diagrams**

****

****

**Process Descriptions**

Interact with enemy:

IF health != 0:

Shoot hot dogs

Play ammo SFX

Increase enemy count

ELSE

Set dead state as true

Check enemy count;

IF enemy count > 5 && checkpoint == 1:

Set grow state as true

# Acceptance Tests 9

For all tests described above, the player will be spawned in an automatic run state with hot dog ammo firing at a constant stream. Level 0 of the game is designed in a straight fashion to allow for automation without directional input from arrow keys. The end of the automation will be the player reaching the first checkpoint.

|  |  |
| --- | --- |
| **At Checkpoint Count Enemy Test** | |
| *Input* | *Output* |
| Enemy count increases within range:  Checkpoint 1: > 0  Checkpoint 2: > 5  Checkpoint 3: > 10  Checkpoint 4: > 15 | Player grows by height and width of 4 scale at each checkpoint. A logfile function is used in C# to display if a checkpoint is reached && enemy count is within range. |
| Enemy count < 0:  Meaning there are no enemies being spawned or the enemy count function fails to set any negative value back to the 0-base limit. | Use the debugging log function in C# to catch all occurrences of this happening. However, the only time enemy count can be < 0 is if the enemy loses in Checkpoint 2 and respawns in Checkpoint 1 (Level 0). This should be rectified by checking at each respawn in each checkpoint that enemy count is greater than 0. If it is < 0, set the enemy count to 0. |
| Enemy count > 20:  Meaning the player is inside Checkpoint 4 and has reached the Final Boss. | At the checkpoint 4, the output should use the log function to display to the developer that “enemy count stopped, entering Final Boss area”. As enemies are no longer spawned in checkpoint 4, the enemy count will be reset to 0 for security. However, the maximum enemies a player can kill per checkpoint is 10. After that, the counter loop is exited. |
| **Hot Dog Ammo Test** | |
| *Input* | *Output* |
| The hot dog ammo shooting function will shoot automatically upon start. The changeable variable is: # of hot dogs fired per second  Good input ranges: # of dogs = 1 – 3  Bad input ranges: # of dogs = < 1 || > 3 | A “good to go” message to the console will occur through the log function when the input is within good range. An error message will be thrown if the bad input range is input.  Any bad input will be defaulted to # of dogs = 3. |
| **World Obstacle Test** | |
| *Input* | *Output* |
| Obstacles (such as bricks or boxes to compose the maze, or powerups) are placed in the world map at the end of the automation period. The player is left to spawn in the world and automatically enter the run state to run forwards towards these world obstacles/items. | Collision detection from the debug log function. A flag will be set to active if the obstacle is a power-up. |

# Timeline /10

**Work items**

|  |  |  |
| --- | --- | --- |
| Task | Duration (PWks) | Predecessor Task(s) |
| 1. Player Design and Prototype | 5 | - |
| 2. Entity and States | 6 | 1 |
| 3. Item Superclass Design | 3 | 1 |
| 4. Item Subclasses Design | 4 | 3 |
| 5. Item Interaction Functions | 4 | 2, 4, World Design, Enemy Design |
| 6. Programming | 5 | 5 |
| 7. Documentation | 4 | 6 |
| 8. Testing | 3 | 6 |
| 9. Integration | 4 | 7, 8 |

**Pert diagram**

**A diagram of a diagram

Description automatically generated**

**Gantt timeline**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Slack |  |
| 2 |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Wrk hrs |  |
| 3 |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  | 3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |  |  | 2, 4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 5 |  |  |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 6 |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 7, 8 |  |  |  |  |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |