Graeme Holliday Mark \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/50

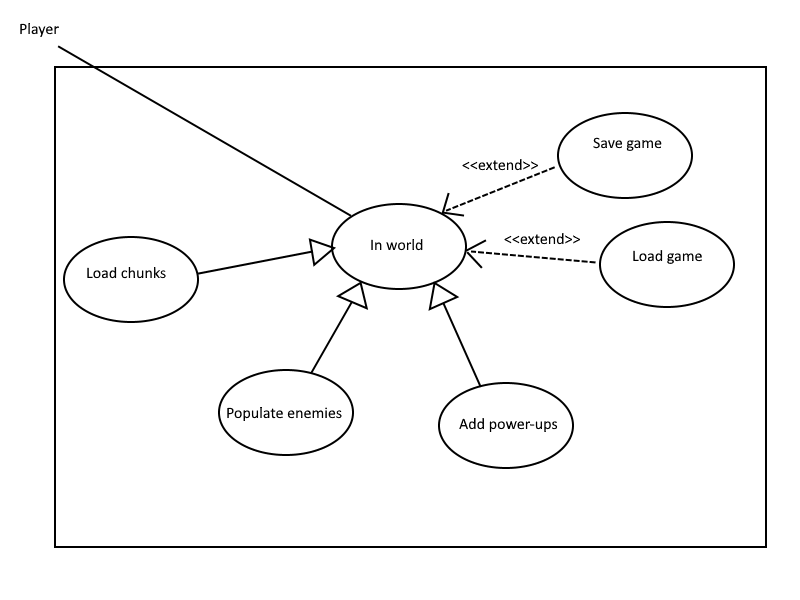
## Brief introduction \_\_/3

My feature is the world generation. The most important characteristics of my task are:

* “Infinite” world generation (limited by physical constraints only)
* Random generation that maintains a realistic feel (probably through use of cellular automata or some noise function variant)
* World must be traversable (pathfinding algorithm ensures each chunk is traversable after generating or it must be replaced)
* Tile-based, but with enough detail to not look overtly pixel art focused
* World is generated in chunks, or some NxN grid of tiles grouped together
* Chunks have biomes associated with them, which affect the textures of the tiles and the types of terrain created
* Chunks must be dynamically loaded and unloaded based on the player’s position in the world

## Use case diagram with scenario \_\_14

### Use Case Diagrams



### Scenarios

**Name:** In world

**Summary:** The world is initialized and loaded into the player’s view

**Actors:** Player

**Preconditions:** Player is approaching a location that is not yet loaded, or game has just begun.

**Basic sequence:**

**Step 1:** Generate the chunk using the random generation algorithm

**Step 2:** Verify chunk is traversable using pathfinding algorithm. If not, repeat step 1.

**Step 3:** Place chunk full of tile entities into the game world.

**Step 4:** Populate newly created chunk with enemies

**Step 5:** Place power-ups into the new chunk

**Exceptions:**

**Step 1:** Player chooses to save and quit game.

**Step 2:** The game state is saved and written to storage.

**OR**

**Step 1:** Player loads a previously saved game.

**Step 2:** Instead of generating a random chunk, load in the previously generated chunk using save data.

**Post conditions:** Chunk populated with enemies, power-ups, and tiles is visible on the screen.

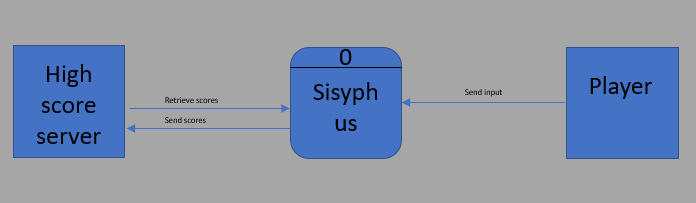
**Priority:** 1

**ID:** C01

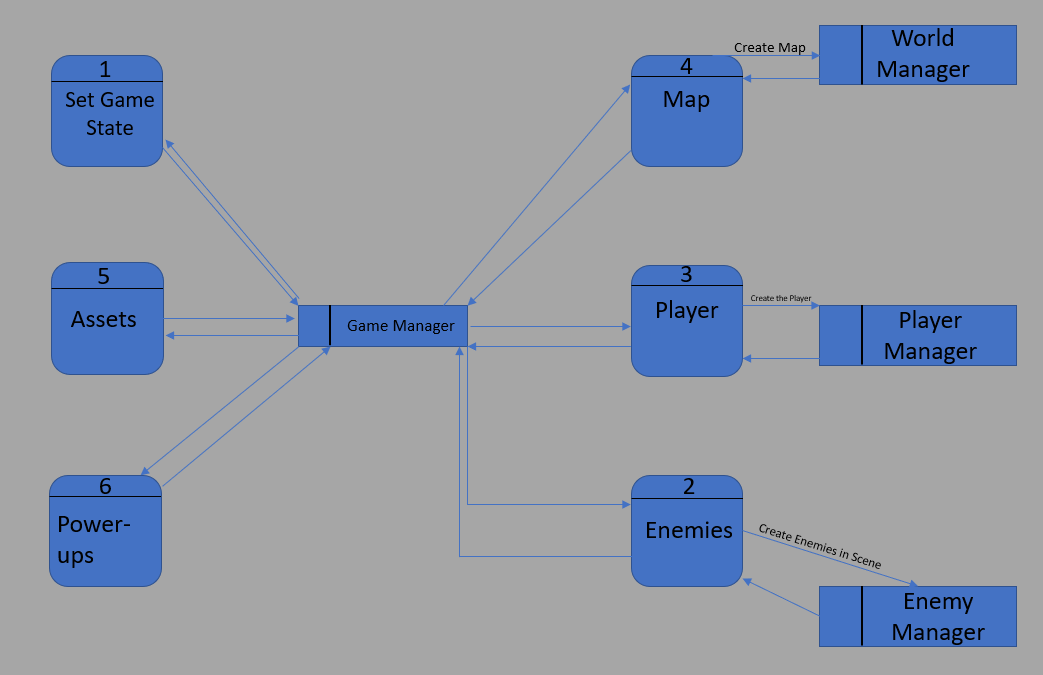
## Data Flow diagram(s) from Level 0 to process description for your feature \_\_\_\_\_\_\_14

### Data Flow Diagrams

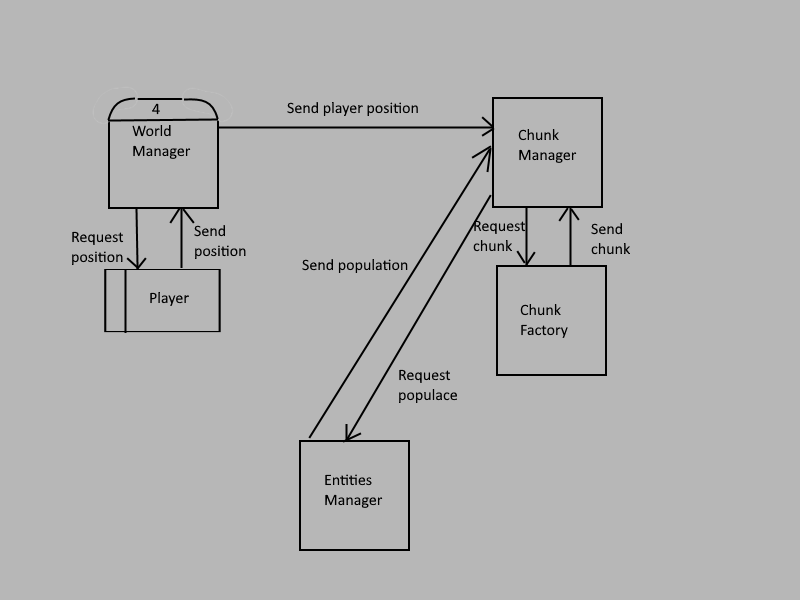
### Level 0



### Level 1



### Data Flow Diagram



### Process Descriptions

Process 4 (Generate Map):

IF player in ungenerated area OR game has just begun

REQUEST player position from player data store

GET player position

SEND player position to chunk manager

REQUEST new chunk from chunk factory

GET randomly generated chunk

REQUEST population from entities manager

GET chunk populated with entities

END WHILE

Process 4.1 (Create Chunk):

REQUEST biome from chunk manager

GET biome

IF biome is type 1

Bias algorithm in some direction

ELSE IF biome is type 2

Bias algorithm in some direction

ELSE

Bias algorithm in some direction

ENDIF

Generate chunk using weighted algorithm

SEND chunk to chunk manager

Process 4.2 (Verify Chunk)

WHILE no path exists through chunk

Run Process 4.1

END WHILE

Processes 4.1 and 4.2 are not pictured in the data flow diagram, but deal primarily with the interaction between chunk manager and chunk factory. They would both show up if you were to go a level deeper into the data flow diagram. I chose to use these two due to a low quantity of processes on this level.

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

Test 1: Ensure chunks are traversable

This test will be performed after each chunk is generated. Steps are as follows:

* 1. After chunk is generated, send it to the pathfinding algorithm.
  2. If the pathfinding algorithm oks it, the chunk must be traversable.
  3. If the pathfinding algorithm rejects it, the chunk must be discarded and a new one will be created.
  4. If this process is being repeated often (say more than half a time per chunk), the generation algorithm should be revisited and optimized.
  5. This process must be followed for each biome, since the generation algorithm is biased by biome values.

Test 2: Ensure chunks are loading/unloading correctly

This test is to be performed on a device to ensure the chunk manager is doing its job. Steps:

1. After building the project to a device, monitor the device performance as the game is played.
2. If the chunk manager is failing to unload chunks, the game will gradually slow down as more chunks are loaded.
3. If the chunk manager is failing to load chunks, that will be immediately apparent as no tiles will be visible.

Test 3: Ensure the number of entities loaded at a given time is optimal for performance.

This test is also to be performed on a device. It ensures the game is optimized for efficiency and speed. Steps include:

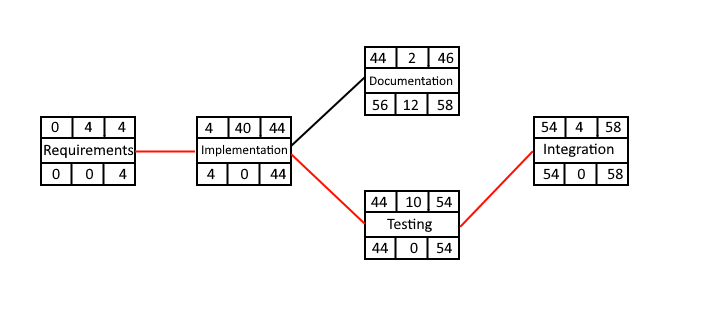
1. Build the project for a device.
2. Play the game for an amount of time (say 5 minutes) while monitoring device performance.
3. Repeat steps a-b 10 times while varying the maximum number of chunks to be loaded by the chunk manager.
4. This test is also limited by visual input. If the player can see a chunk being loaded, then the chunk is being loaded too late.

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

### Work items

|  |  |  |
| --- | --- | --- |
| Task | Duration (hours) | Predecessor Task(s) |
| 1. Requirements Collection | 4 | - |
| 2. Coding/Implementation | 40 | 1 |
| 3. Testing | 10 | 2 |
| 4. Documentation | 2 | 2 |
| 5. Integration | 4 | 3 |

### Pert diagram



### Gantt timeline

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

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 |