# Final Project Log:

## Week 1-2 (18/01/23 – 01/02/23):

Aims:

* Create a demo of the game.
* Create a random nxn dungeon.
* Create an AI player that can navigate from the start to the end.

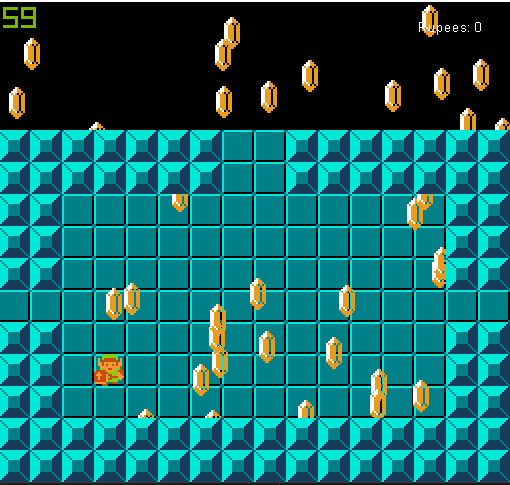
### Day 1 (20/01/23):

* Created room layout, moving player, and collectible rupees.
* Walls are Sprites that are added manually, and overlap with each other.
* 5 pixels per frame = 300 pixels per second?



Goal for next day: Turn room layout into a matrix. Add walls to list using this matrix.

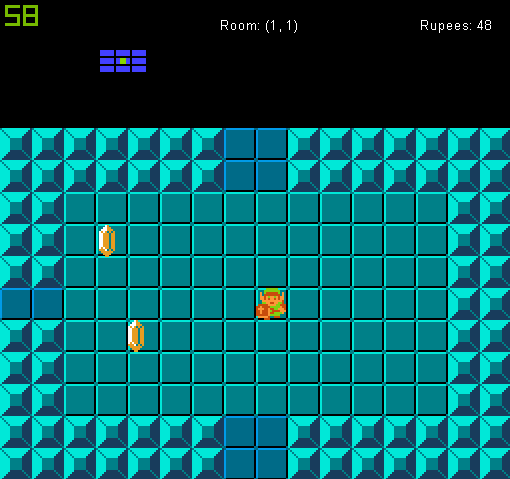
### Day 2 (21/01/23):



* Walls are now determined by 2D arrays.
* Rooms have 4 flags stating where their doors are.

Tomorrow: Transition between different rooms.

### Day 3 (22/01/23):



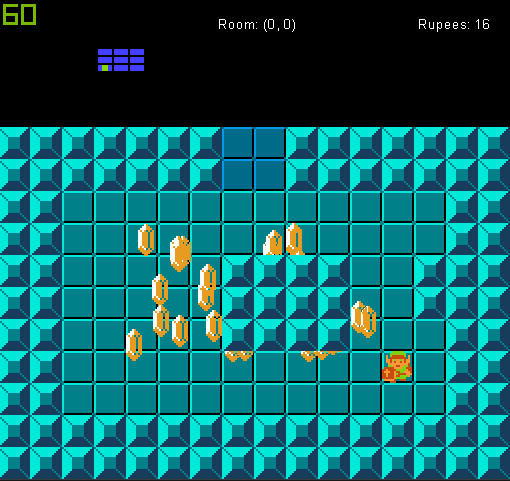
* Rooms are now stored in a matrix. Rooms are connected via 4 links that they have to neighbouring rooms. Inefficiencies from copied code removed.
* Minimap added, shows layout of dungeon, and link’s current room.
* Made a 3x3 maze with 2 branching paths.

Minimap dimensions:

* Link/triforce: 3x3 -> 9x9
* Rooms: 7x3 -> 14x6
* Room gap: 1 wide -> 2

Tomorrow: Start working on A\* to navigate to doors.

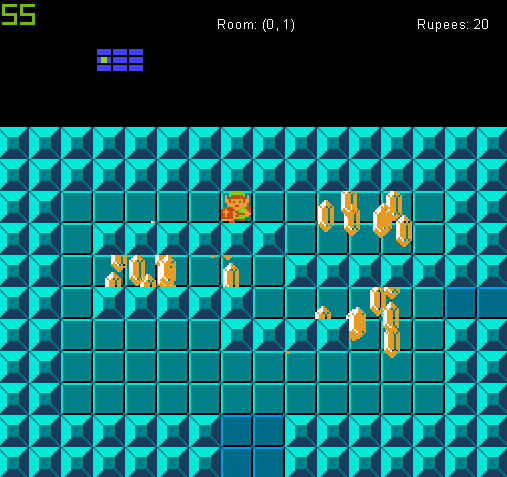
### Day 4 (23/01/23):



* Started working on A\* navigation.
* Arcade’s in-built algorithm was functional but failed to account for walls somehow and was too restricting.
* Began work on my own algorithm that I can tailor to my liking.
* Uses a grid of nodes rather than a list.
* Reaches goal, but not very well.
* Takes a very roundabout path due to no heuristic.
* Does not seem to care about walls, tries to ram through them.
* If it tries to go through 3 or more walls the game crashes.

Tomorrow: Try and fix A\*. Check if neighbours are correct, walls are not added etc.

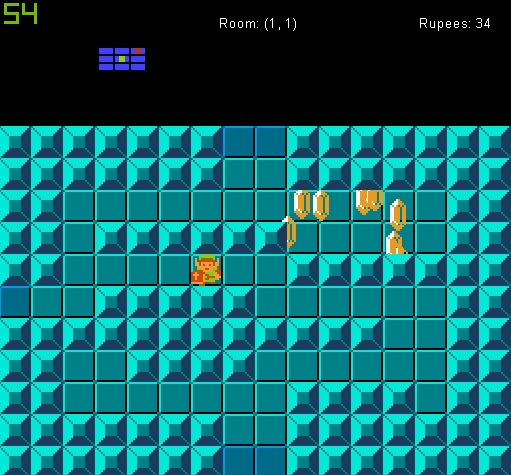
### Day 5 (25/01/23):



* Fixed A\* algorithm, it now always goes to the correct location, taking the fastest path (this was due to not removing nodes from open set), and without going through walls (due to misaligning the graph creation).
* Link will now restart A\* again when travelling to a new room, meaning he will soon be able to fully navigate between rooms.
* Split room and A\* related functions and classes into other files to structure the code. Will likely do this with the player in the future as well.

Tomorrow: Make Link locate a door to walk towards to initiate A\* without me specifying his goal.

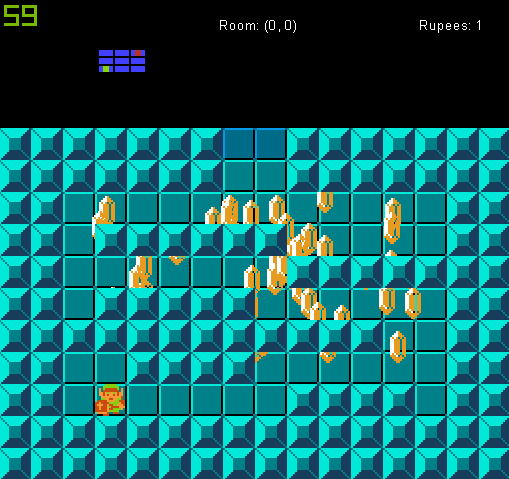
### Day 6 (26/01/23):



* A\* now restarts automatically when entering a new room.
* Will target a random door in the room that isn’t the one Link just came from. Stops when at a dead end.
* Added a marker to show where the final room should be.
* Made a GitHub repository to backup my files.

Tomorrow: Implement a depth-first search to reach the goal.

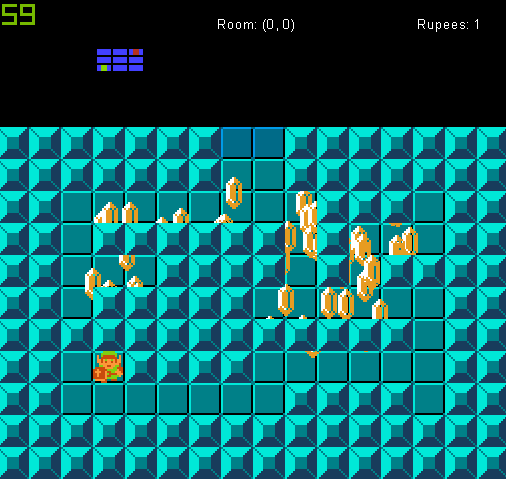
### Day 7 (29/01/23):



* Moved much all of Link’s AI into a player class to organise the code better, and keep all of the functionality self-contained.
* Began preparing for Link to choose between his own objective.
* Currently does not start moving automatically as the way A\* worked previously no longer matters.
* Began implementing the depth-first search.

Tomorrow: Continue with depth-first search. Hopefully help guide it to the end, backtracking may be difficult.

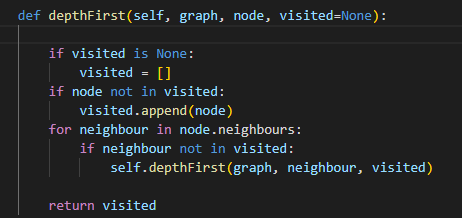
### Day 8 (30/01/23):



* Transferred all map related code to the Map class, tidying up the main file even further.
* Now that the map itself is an object consisting of rooms, it can be traversed using a depth-first search more easily.

Tomorrow: Implement depth-firs search using the Map class.

### Day 9 (31/01/23):



* Implemented depth-first search that returns a list of rooms in the order they should be visited by Link.
* There are no repeats, meaning that the path Link needs to take to backtrack is not accounted for.

Tomorrow: Find a way to include the backtracked path, and/or use the depth-first search to determine Link’s next move.

## Week 3-4 (01/02/23 – 15/02/23):

Aims:

* Implement depth-first search fully, keeping track of room history.
* Implement parametrised dungeon generation.

### Day 10 (02/02/23):

Idea:

When a branch is reached, add room to a stack?

Choose one door .

Straight line to next room. Add to stack.

Another branch is reached, at this to stack.

Choose door.

Dead end. Pop stack, return to last room.

Check other door.

Straight line, add to stack.

Dead end, pop from stack and return to last room.

Return again, now all doors have been checked. Pop from stack.

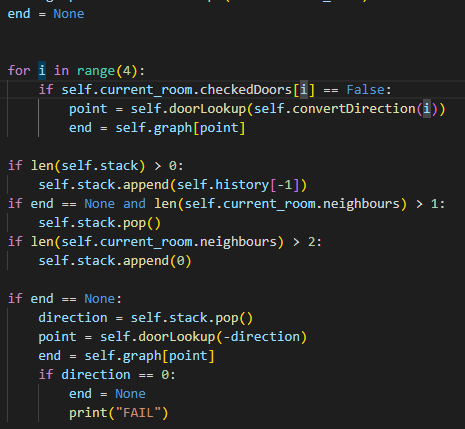
Pop stack again, go through straight room to Original branch.

Can now continue with other door.

If a branch is found, start adding every room to the stack. Add a room to the stack if it has more than 2 doors, or if the stack already has something in it.

Keep track of which of a room’s doors have been explored. When backtracking to a previous room, this can be used to look at the other path.

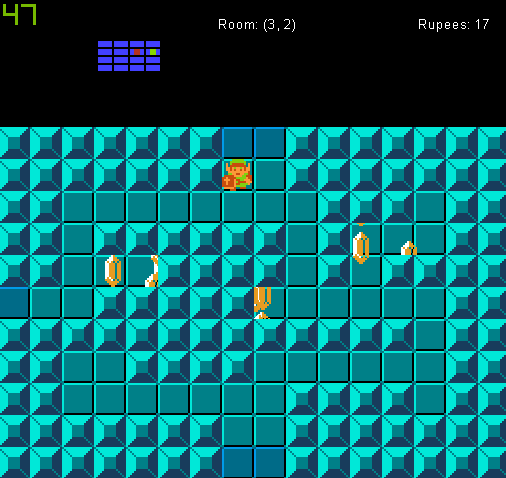
### Day 11 (03/02/23):



* Depth-first search now implemented and working.
* Link will choose a path when reaching a branch, then once he reaches a dead end will return and check the other path.
* If all paths are exhausted he will return to original branch and stop. This shouldn’t happen normally as he should fine the exit before this ever happens, but may be a problem later.
* Still needs testing with bigger mazes that have branches off of branches.

Tomorrow: Start working on maze generation. Will need to use it to create bigger mazes in order to fully test depth-first search. <https://www.algosome.com/articles/maze-generation-depth-first.html>

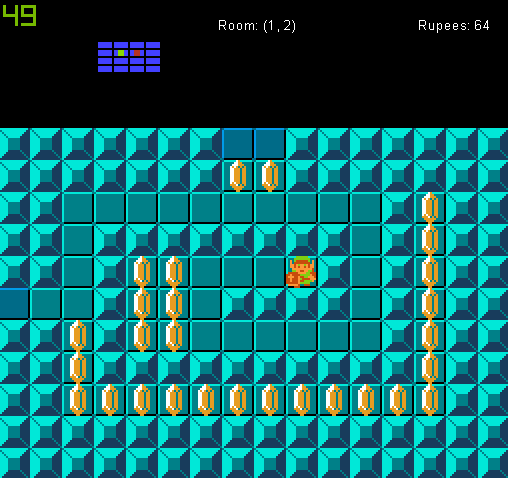
### Day 12 (06/02/23):



* Implemented depth-first maze generation.
* Creates fully traversable mazes, with any size.
* No longer need to define room connections manually in any way.
* Depth-first traversal still works, but breaks when a dead end is reached early on before a branch. Likely need to add to just check first room again? Shouldn’t be able to reach a dead end without already reaching a branch.
* Depth-first traversal could break when multiple paths are added. Will definitely need to implement A\* as well.

Tomorrow: Start making the rooms unique from each other. Have their own rupee list, and own floor plan. Make several floor templates that the rooms will pick from.

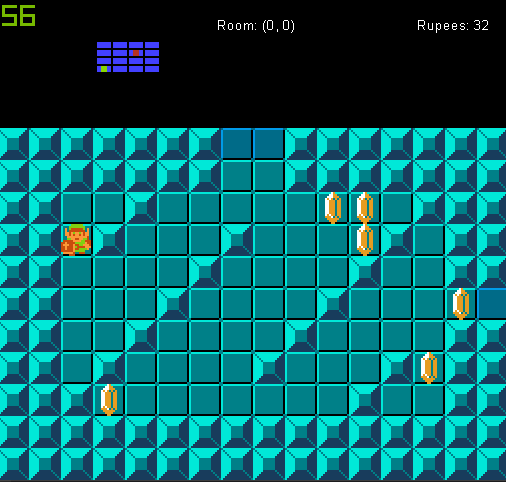
### Day 13 (07/02/23):



* There are now 14 predefined templates for rooms to use, all stored in a list for easy access. Currently, they are randomly chosen between to make each room feel special.
* Rupee lists now belong to individual rooms rather than the game itself. This means that each room can have its own amount, and they will disappear when leaving the room. They will not reappear when returning to the room if they have been collected. Currently, every floor tile spawns a rupee.
* Enemy list has been made, yet to be implemented.
* Much obsolete code has been deleted.

Tomorrow: Iron out how rupees are spawned. Begin working on targeting rupees as their own objectives.

### Day 14 (08/02/23):



* Amount of rupees is now determined by “rupee density”. Each tile has a chance of spawning a rupee, based on the density of the room assigned during map generation. 1 density means every floor tile will have a rupee.
* Link can now target rupees and collect them. When entering a room, he will decide which rupees he wants to collect, and add them to a list. The likelihood of each being added is determined by his “rupee affinity”. He will then collect each of them, in order of which he is currently closest to. He may also collect rupees he does not care about if they are in his way.
* Sometimes his paths can be inefficient, due to targeting rupees on the other sides of walls since they seem closer, and when he has to take a split path, backtracking may take a while. However the system still works perfectly fine, and this is only really an issue when there are many rupees in the room.

Tomorrow: Tidy up rupee collection function, make movement its own function that the depth first search and rupee collection both use. Use a system of “states” to determine which task he is currently focused on.

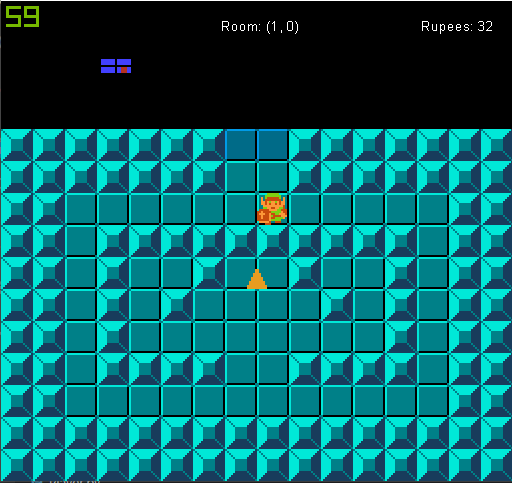
### Day 15 (09/02/23):



* Rupee collection, depth-first traversal, and movement are now all separate functions that can coexist at the same time.
* Link can now clear a whole dungeon while collecting all rupees on the way with no input from the player.
* Depth-first search is now even more broken, now breaking when reaching the end. Also still doesn’t work with starting rooms and repeated branches.

Tomorrow: Fix depth-first search, add an actual dungeon goal with the Triforce and all.

### Day 15 (10/02/23):



* Added Triforce, cannot be collected yet, but spawns in its own room with no treasure.
* End goal is now dynamically calculated, being the room that is furthest from the entrance.
* Minimap red dot now also tracks where the end goal is.

Tomorrow: Add Triforce collection state, make collecting it end the game. Fix depth-first search.

### Day 16 (13/02/23):



* Depth-first search is now fully fixed, will now explore every room in the maze before returning to the exit.
* Changed how rupee target works, no longer re-targets rupees, stopping infinite loops from happening.
* Only needs Triforce collection and enemies and “game” portion is complete.

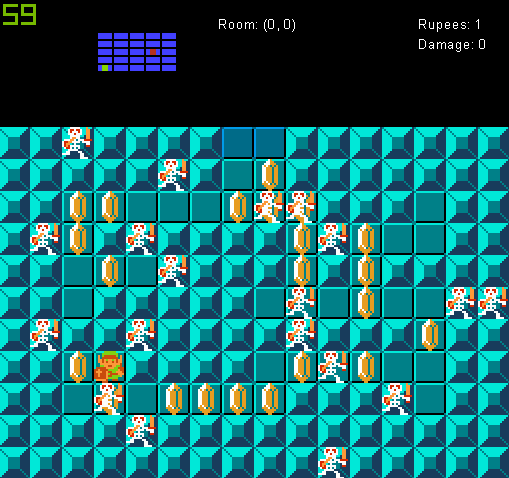
Tomorrow: Add in Triforce collection.

## Week 5-6 (15/02/23 – 01/03/23):

Aims:

* Add in enemies.
* Get dungeon generation fully complete, with all necessary parameters.
* Decide what statistics need gathering.

### Day 17 (15/02/23):



* Triforce collection done, Link will pick it up automatically when reaching final room and the game will end.
* Enemies have been added. Every room has its own list of enemies, they are currently spawned in just as rupees are. Enemies have their own class.
* Enemies die on contact with the player. They deal damage to Link based on his combat ability.
* Enemies currently move around randomly with no respect for collision. Their speed is determined by a parameter independent of the player.

Tomorrow: Make enemies account for walls.

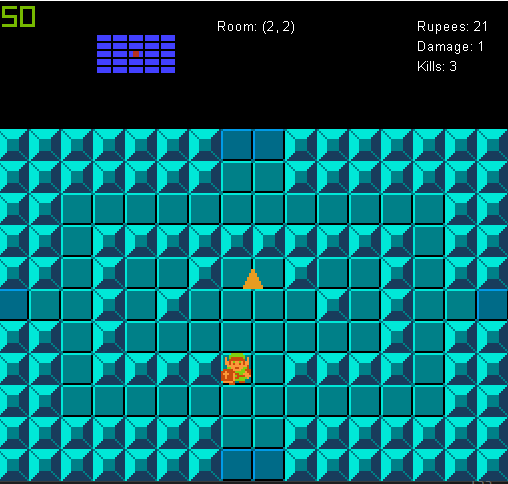
### Day 18 (20/02/23):



* Enemies will now check if a movement is valid by choosing it, stopping them from walking into walls.
* Kill count is now tracked.
* Link will successfully target all enemies in the room and kill them.
* Game doesn’t crash.

Tomorrow: Add in extra doors.

### Day 19 (21/02/23):



* Extra doors are now added in manually, allowing for multiple paths.
* Currently done completely randomly.

Tomorrow: Make it so triforce only has one entrance. Make it so number of 3 and 4 door rooms are counted.

### Day 20 (27/02/23):

Parameters:

* Rupee = +1
* Kill = + 2
* Damage = -4
* Room entered: + 10?
* Every frame: -0.1?

Idea:

* Store every room in a list.
* Shuffle the list.
* Add a door to each room if it has 2 doors.
* Continue until quota is met or at end of list.
* Iterate through list again, add door to each room if it has 3 doors.
* Continue until quota is met or at end of list.

## Week 7-8 (01/03/23 – 15/03/23):

* Split maze up into quadrants, assign parameters on a quadrant-by-quadrant basis.
* Think of a target of what the player parameters should actually produce:
* A player who really likes killing enemies should have many enemies placed along his path to the goal. Don’t place enemies outside that path because then he will waste time killing enemies before getting to the end, and it will get boring.
* Should be a time penalty for taking a long time, ensuring that the gameplay that is experienced is dense.
* Players who ignore treasure and enemies will finish faster so they need content spread out across the whole path, and will need fewer extra doors to make sure they don’t get right to the end immediately. We want lots of rooms to be explored.
* Door parameters can just be the number of doors added to each quadrant. Allows finer control while also simplifying implementation.

### Day 21 (03/03/23):

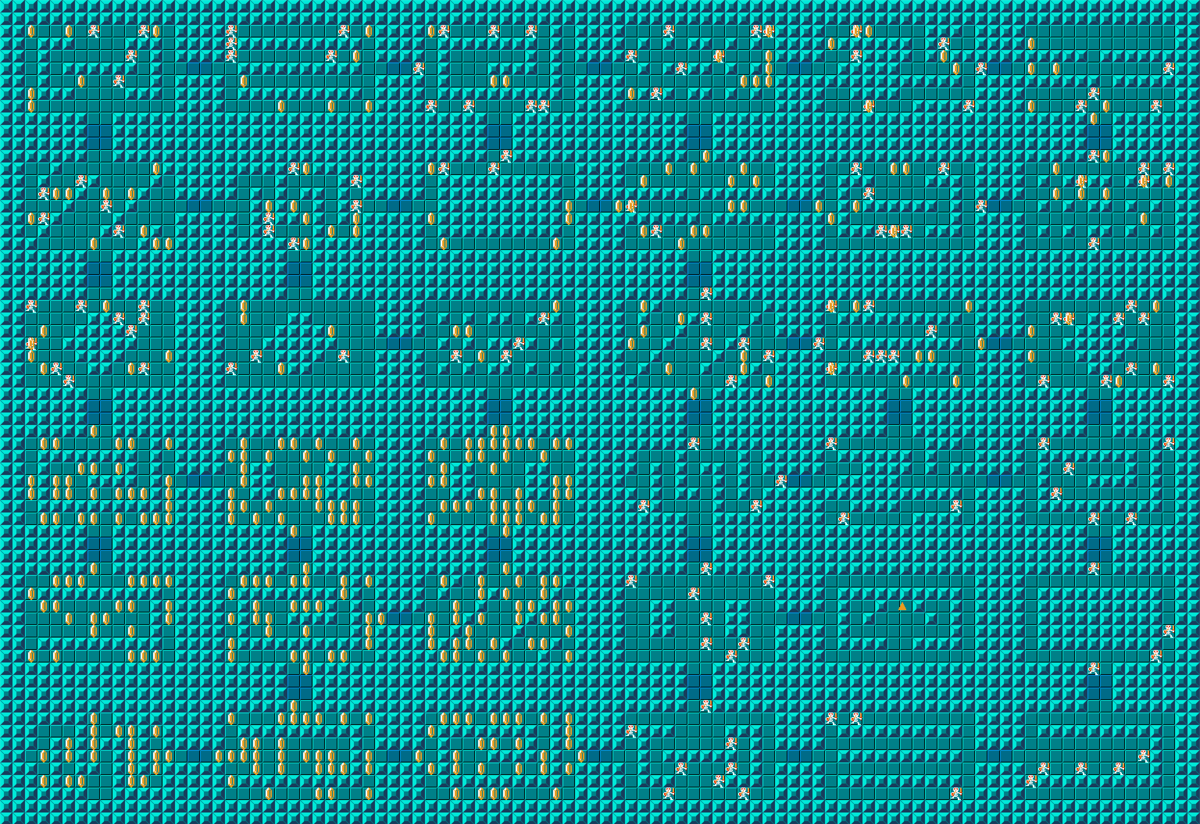
* Rooms are now divided up into quadrants.
* Each quadrant has a certain amount of extra doors allocated to it, rooms are given these doors in order.
* Rooms are shuffled so there is no bias towards the bottom left rooms.

Tomorrow: Only deduct from the door counter if there wasn’t already a door there.

### Day 22 (04/03/23):

* Doors are now only added if there wasn’t already something there. No doors are wasted.
* There are now separate rupee and enemy densities for each quadrant. Consider making these densities a flat number per room, rather than a chance per tile?
* Triforce room can no longer spawn extra doors. This minimises the chance of getting an easy line to the exit due to lucky door placement.
* Link now uses a heuristic to explore, which informs his decision making on which door to pick. This makes mazes with many doors much quicker consistently.

### Day 23 (06/03/23):



* Added tilemap generator for the dungeon.
* Exports map to png after the dungeon is made, before the game starts.

## Week 10 (22/03/23 – 29/03/23):

Aims:

* Make a system for rerunning games once they are completed. Create new game class?
* Figure out fitness function, what makes the most accurate score?
* Find a way to carry over random seed between games. This ensures the maze is the same, but anything parameterised will still be different.
* Implement Bayesian optimisation using sklearn: https://hyperopt.github.io/hyperopt-sklearn/
* Figure out which parameters can be cut. Ideally, the optimisation should have 10 or less parameters. The shorter the run time, the more we can afford.
* Genetic algorithm would be more suitable for many parameters, but would take far too long.

### Day 24 (23/03/23):

* Added restart function to reset game back to starting state, using given parameters.
* Due to randomisation, mazes are completely different each time.

### Day 25 (24/03/23):

* Set RNG seed at start of program.
* Seed is reset every time the game is completed, ensuring that the maze is the same.
* Only differences in the new dungeon are a result of the parameters.

### Day 26 (25/03/23):

* Commented all code up to this point.

### Day 27 (26/03/23):

* Rupee = +1
* Kill = + 2
* Damage = -8
* Room entered: + 10?
* Every frame: -0.1?

### Day 28 (27/03/23):

* Fitness function done
* Output to csv done
* Need to remove 2 quadrants so there are only 6 parameters.
* Make actual parameters written to file