Player : Blue Warshmallow

Enemy : Red Warshmallow

Controls: Arrow keys to move, mouse left lick to shoot

Ai triggers are placed all over the level, green triggers are for horizontal movement, while red triggers are for jumping.

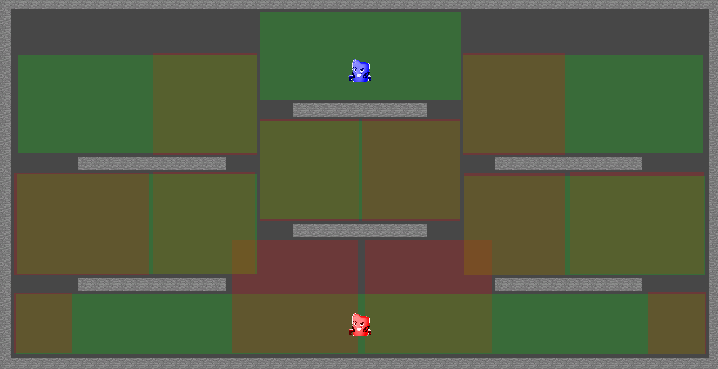


Figure 1 not all triggers end up being used in pathfinding.

Green triggers activate *OnStay* meaning the enemy will move horizontally as long as it’s within said trigger.

Red triggers activate *OnEnter* meaning the enemy will jump only on the first frame it makes contact with the trigger, assuming the enemy was on the ground that is.

The triggers are connected together through a node system.

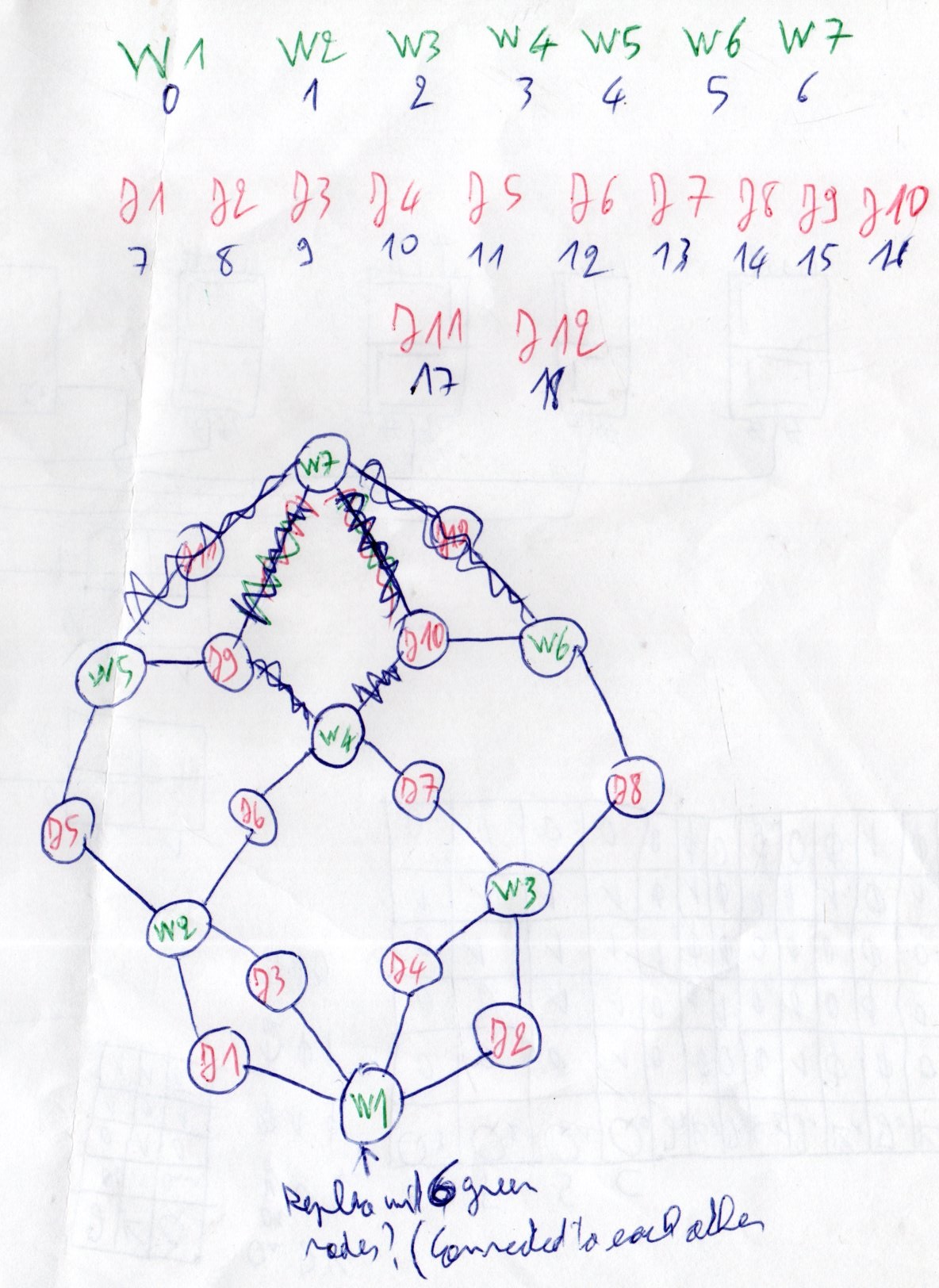
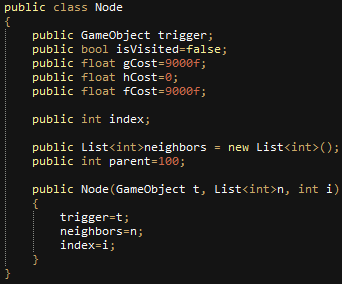


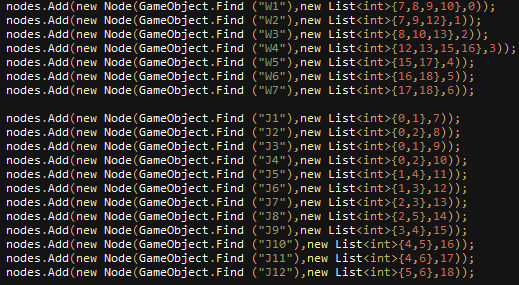
Figure 2 early iteration, not all nodes end up being used.

Every level is going to require its own personalized network of triggers depending on where the enemy will need to jump, walk or drop down.

This is the node class containing a GameObject reference for its associated trigger, a flag signaling whether it has already been iterated over or not, variables for the different cost/distance values, it’s index within the list, a list storing the indices of its neighbors, and the index for its eventual parent node, as well as a constructor.



A number of these nodes are then declared and stored with in a list, one for each trigger in the level and its neighbors.



An A\* search algorithm is used to determine the shortest possible path from the starting point trigger to the target trigger, I will not go into detail as to how the algorithm works as it’s publicly available.

Here is a good source:

<https://www.101computing.net/a-star-search-algorithm/>

AI breakdown:

The pathfinder will locate the green trigger containing the enemy and mark it as the starting node, if enemy ammo is less than 16, then the trigger containing the ammo pick up is marked as the target node, if not, the trigger containing the player is marked as the target node.

The pathfinder will then calculate the shortest possible path from the starting node to the target node, and store the indexes of the path nodes in a list, then it will iterate over that list activating the colliders of the triggers associated with those nodes, while skipping over the jumping triggers that aren’t needed (in the case of a drop down from one platform to the other).

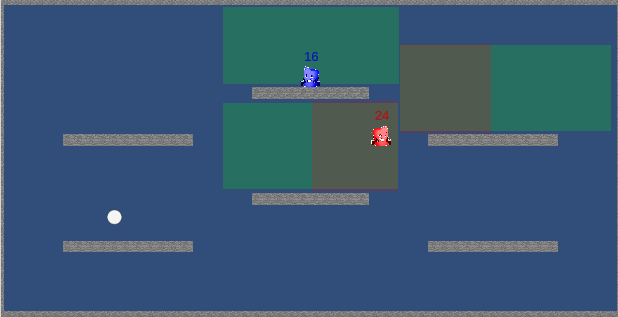


Figure 3 ascending path, all jumping triggers are enabled.

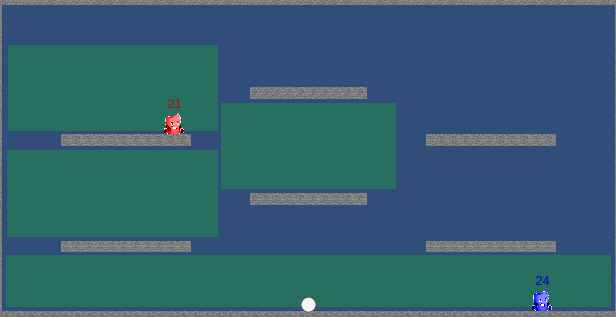


Figure 4 descending path, all jumping triggers are disabled.

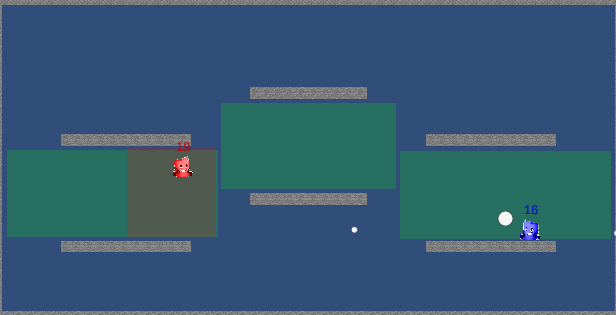


Figure 5 one jumping trigger is enabled while the other is disabled.

The horizontal direction of the enemy is decided by the position of the next trigger relative to the trigger it is currently in.

All colliders are re-activated every 30 frames to refresh the positions of the player and the enemy and recalculate the path, that is why they keep flashing constantly.

If the enemy happens to be inside the same trigger as the ammo pickup, it will go towards it automatically.

The enemy automatically fires at the player if it has a direct line of sight, which is determined using the *Physics2D.Linecast()* function.

The slider in the top left is the “Digression” slider, as opposed to aggression, because the enemy is already very aggressive in it’s default state, the slider’s current max value is 1 because it’s a work in and the AI bugs out with values over 1.

When set to 1, the enemy will not go the node the player is located in, but rather will stay in the one closest to it…

At least that’s what happens in the engine, for some reason a lot of things stop working in the exported builds, including the slider and the player’s double jump, terribly sorry for that, I did my best to fix it to no avail.

Speaking of the latter, a double jump (and possibly even the dash) can be implemented with this AI, but I’ll have to create a third (and fourth) type of triggers for it and the node network will become even bigger.