FOODFINDER

(AI FOR VIDEOGAMES PROJECT)

AUTHOR: Federico Maglione 962587

SUMMARY

1 GAME CONCEPT

Like any living being, the NPCs present in the game have the goal of eating to survive. To do this they have to search for food along the map. Each NPC has a percentage of energy (which identifies life). Whenever an NPC eats a food, it feeds its energy.
Each time an NPC makes a move, the energy decreases. Therefore, to survive, they must search for food as quickly as possible.

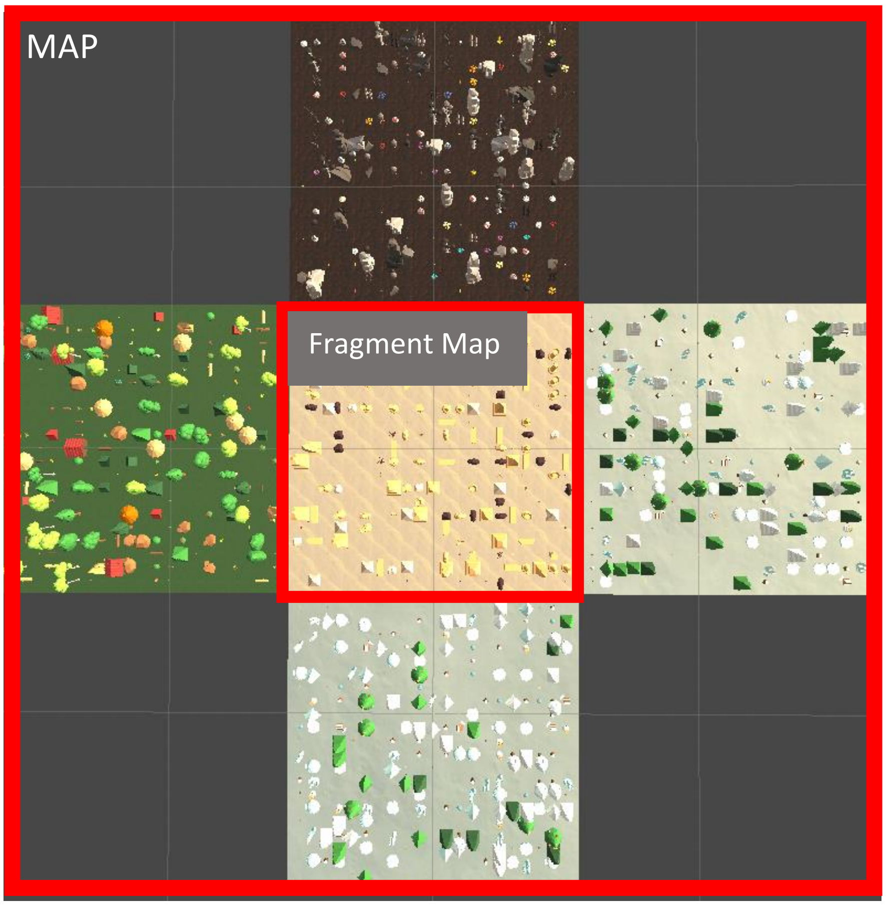
To find food, each NPC has physical characteristics, such as field of vision, speed of movement, weight, height etc. These characteristics have an impact on the search for food. For example, if an NPC is heavier but has a greater field of vision than the others, it will tend to identify the food that surrounds it before the others but with a slower movement speed.
If an NPC is faster but not very tall, it will tend to move faster but will have a harder time identifying the food around it.

When the energy runs out, the NPC can rest, recover a percentage of energy and start looking for food again. This action is allowed only once per NPC, after which the NPC dies.

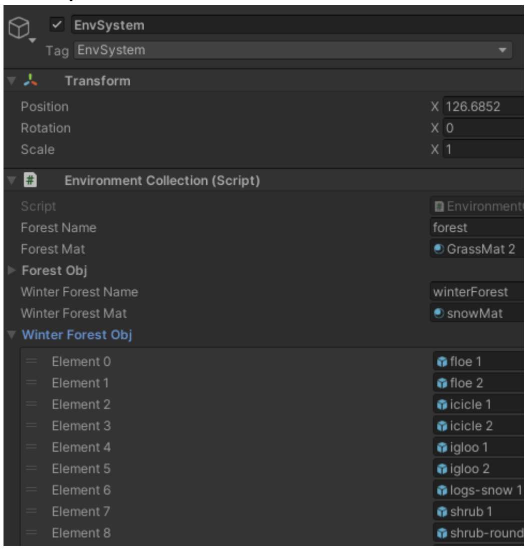
2 GAME ELEMENTS

2.1 ENVIRONMENT

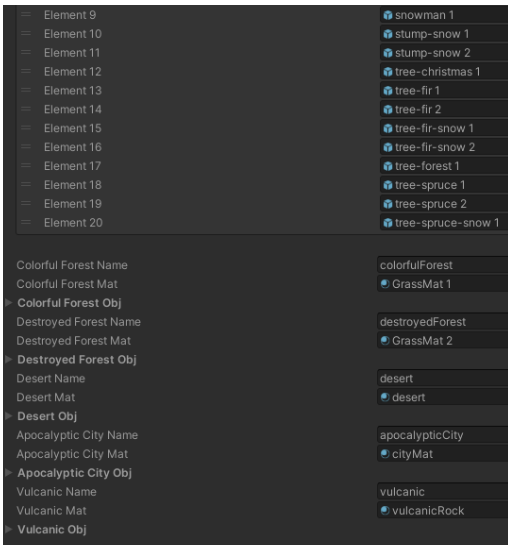
- **The map** is structured with a set of smaller maps which we will call fragment maps. Each fragment map can have a different environment given by a group of possible environments. Like in the example below:



- **Env System**: Game object used to represent the set of possible environments that can be instantiated. Each time a new fragment map is generated its environment is extrapolated from this set.



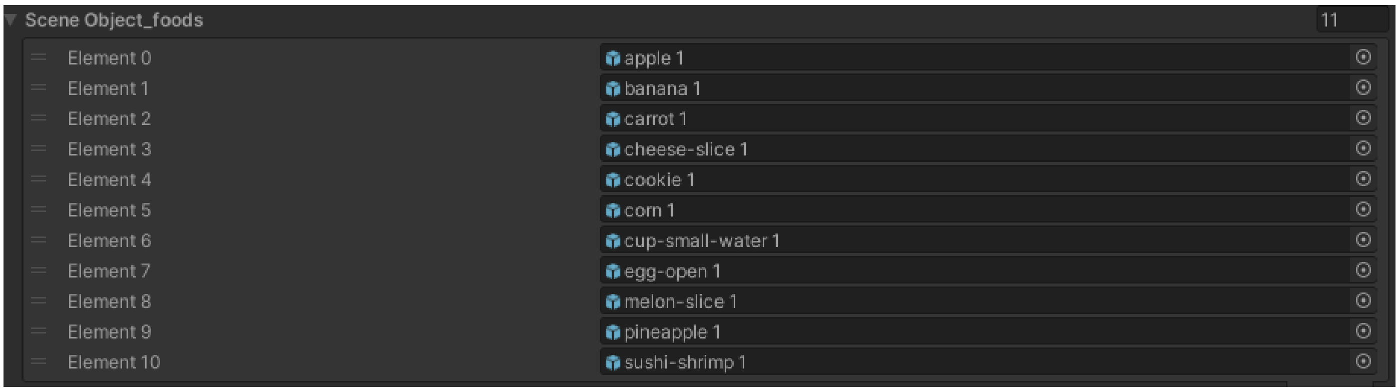
```
✓ EnvSystem
Tag EnvSystem
Transform
Position                         X 126.6852
Rotation                         X 0
Scale                            X 1
Environment Collection (Script)
Script                           Environment
Forest Name                      forest
Forest Mat                       GrassMat 2
▶ Forest Obj
Winter Forest Name               winterForest
Winter Forest Mat                snowMat
▼ Winter Forest Obj
    Element 0                    floe 1
    Element 1                    floe 2
    Element 2                    icicle 1
    Element 3                    icicle 2
    Element 4                    igloo 1
    Element 5                    igloo 2
    Element 6                    logs-snow 1
    Element 7                    shrub 1
    Element 8                    shrub-round
```



```
    Element 9                    snowman 1
    Element 10                   stump-snow 1
    Element 11                   stump-snow 2
    Element 12                   tree-christmas 1
    Element 13                   tree-fir 1
    Element 14                   tree-fir 2
    Element 15                   tree-fir-snow 1
    Element 16                   tree-fir-snow 2
    Element 17                   tree-forest 1
    Element 18                   tree-spruce 1
    Element 19                   tree-spruce 2
    Element 20                   tree-spruce-snow 1

Colorful Forest Name             colorfulForest
Colorful Forest Mat              GrassMat 1
▶ Colorful Forest Obj
Destroyed Forest Name            destroyedForest
Destroyed Forest Mat             GrassMat 2
▶ Destroyed Forest Obj
Desert Name                      desert
Desert Mat                       desert
▶ Desert Obj
Apocalyptic City Name            apocalypticCity
Apocalyptic City Mat             cityMat
▶ Apocalyptic City Obj
Vulcanic Name                    vulcanic
Vulcanic Mat                     vulcanicRock
▶ Vulcanic Obj
```

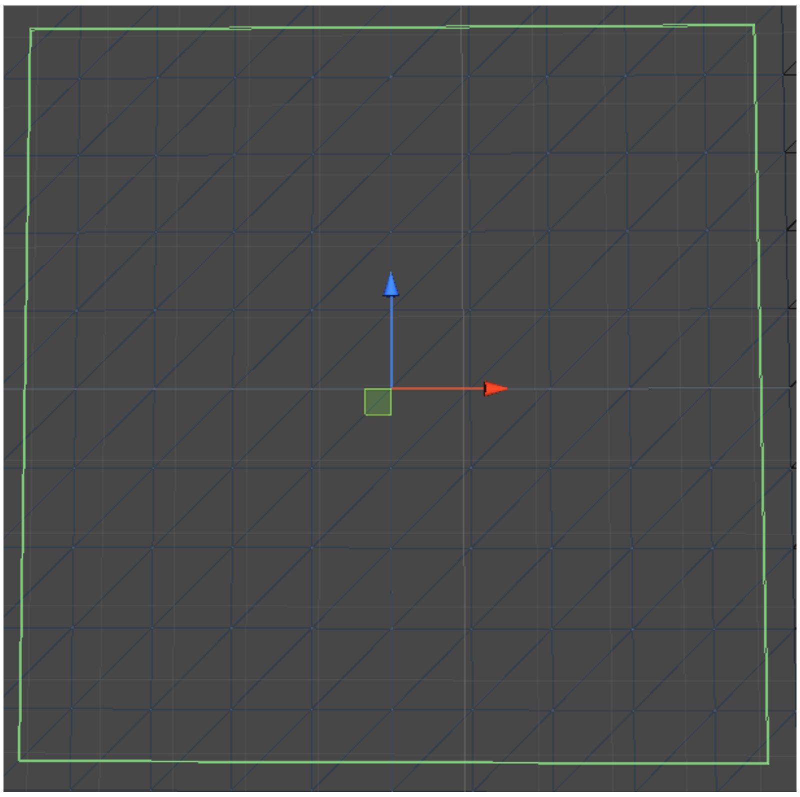
- **Foods:** Each fragment map presents a list of foods that can spawn on the map



Scene Object_foods 11

Element 0	apple 1
Element 1	banana 1
Element 2	carrot 1
Element 3	cheese-slice 1
Element 4	cookie 1
Element 5	corn 1
Element 6	cup-small-water 1
Element 7	egg-open 1
Element 8	melon-slice 1
Element 9	pineapple 1
Element 10	sushi-shrimp 1

- **Colliders**: Each fragment map has a collider used to identify when an NPC enters that section of the map

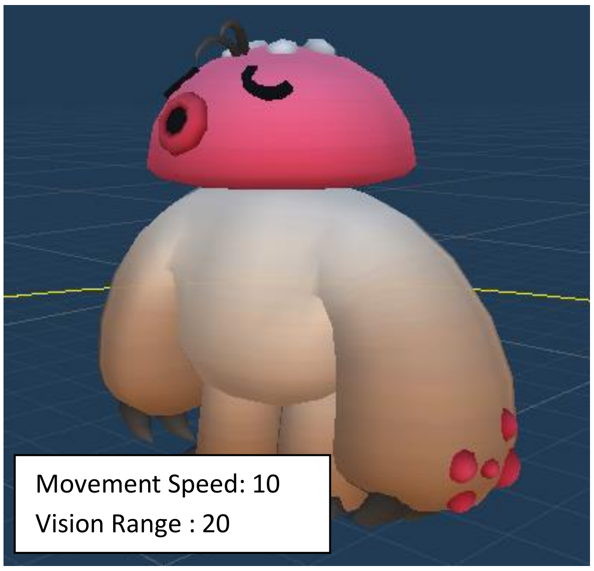


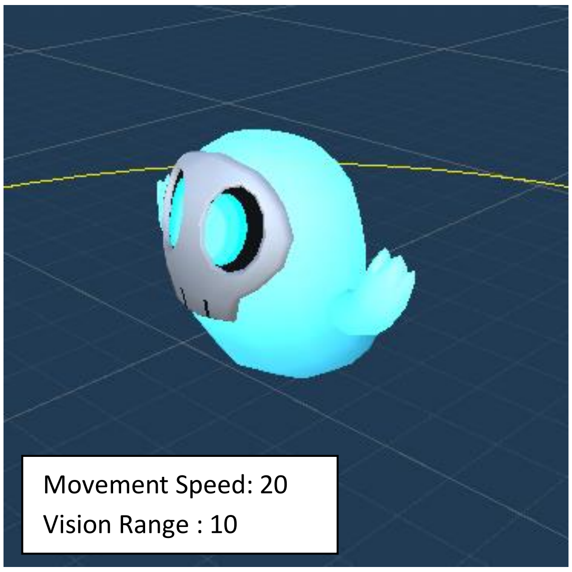
2.2 NON PLAYER CHARACTER PROPERTIES

Each NPC has physical characteristics:

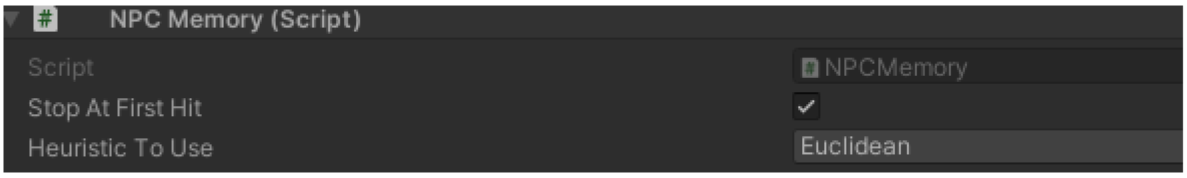
- Reaction Time: How often the NPC do an action.
- Energy: Initial energy of the NPC
- Energy gain: How much energy he takes from food
- Movement Speed
- Rotation Speed
- Vision Range: How far can he find food

These characteristics change depending on the physical structure (model) of the NPC. Like in example below:





Each NPC is provided with a script to calculate the route with different heuristics.



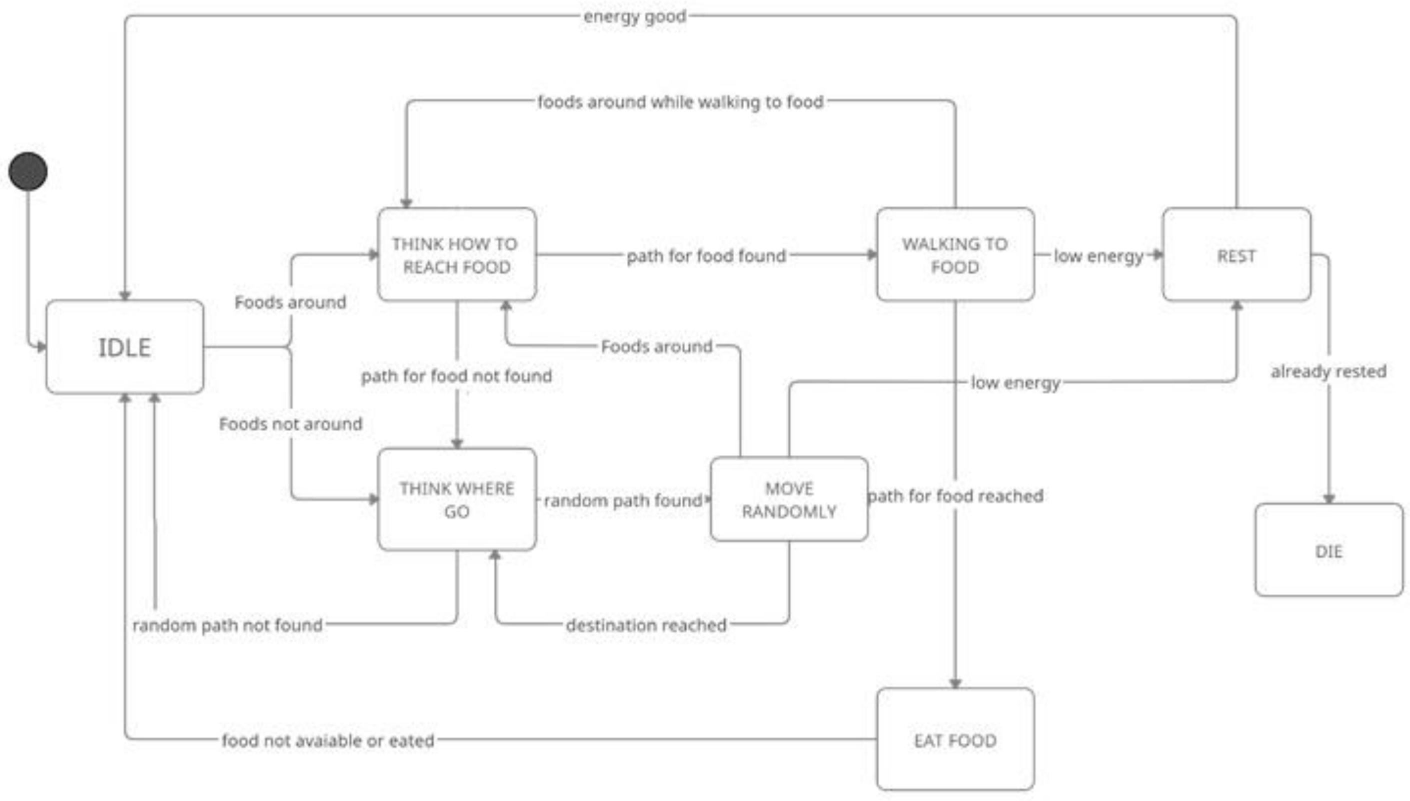
NPC Memory (Script)

Script	NPCMemory
Stop At First Hit	✓
Heuristic To Use	Euclidean

3 ARTIFICIAL INTELLINGENCE

3.1 NON-PLAYER-CHARACTER – FSM

The behavior of the NPC is managed through an FSM with the following structure:



3.1.1 STATE

- **IDLE**: The NPC does nothing.
- **THINK HOW TO REACH FOOD**: Calculate the path to reach the food.
- **THINK WHERE GO**: Calculate random path to follow.
- **WALKING TO FOOD**: Move the NPC along the calculated path until reaching the food.
- **MOVE RANDOMLY**: Move the NPC along the random calculated path.
- **EAT FOOD**: The NPC eats the food and increases its energy.
- **REST**: The NPC rests to regenerate some energy.
- **DIE**: The NPC die.

3.1.2 TRANSITION

- **Foods around**: Check if there is any food in his range of vision.
- **Foods not around**: Check if there isn't any food in his range of vision.
- **Path for food found**: Check if the calculated path for food is available.
- **Path for food not found**: Check if the calculated path for food isn't available

- **Foods around while walking to food**: Check if along the food path there is a near food to eat.
- **Path for food reached**: Check if the NPC reached the food.
- **Food not available or eated**: check if the food has been eaten
- **Random path found**: Check if the random calculated path is available.
- **Random path not found**: Check if the random calculated path isn't available.
- **Destination reached**: Check if the random calculated path has been followed.
- **Low energy**: Check if the energy is equal to 0.
- **Good energy**: Check if the energy is above 0.
- **Already rested**: Check if the NPC already rested.

3.1.3 STATE – TECHNICAL DESCRIPTION

Below is a technical description of the more complex states:

- **WALKING TO FOOD**: This state, as already said, allows the NPC to move from its node position to the node position of the food following the calculated path. To do this, two nested coroutines are called:
  The first is ***MovingToFood* ()**:
  it allows you to check every time we move from one node to another until the destination node is reached.
  The second ***NodeToNodeMovementFood*()**:
  It allows you to move the NPC from node a to node b with a smooth transition (using the MoveTowards method) and with a smooth rotation (interpolating the current direction and the new direction)
- **MOVE RANDOMLY**: Same as walking to food with a difference that it's work with a random destination node.

3.1.4 TRANSITION – TECHNICAL DESCRIPTION

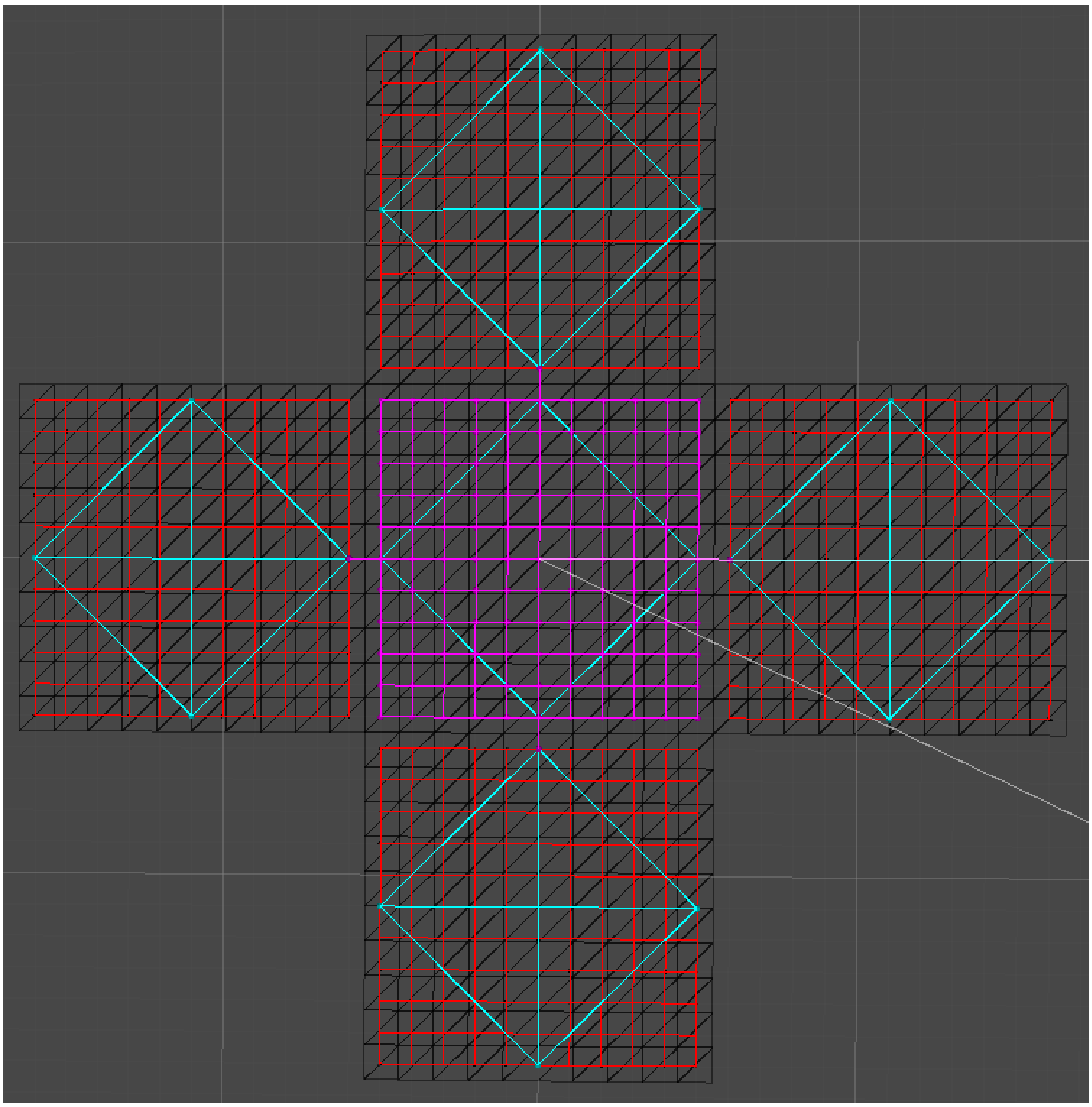
Below is a technical description of the more complex transition:

- **Foods around**: Check's every game object with the specific food tag if there is at least one in the vision range of the NPC. From the list of food in the range it's takes the food with the minimum distance from the NPC, then stop the coroutine, assign the new target food node, and then return true to start the next state.
- **Foods around while walking to food**: This transition can occur when the NPC is following a path for a found food. This transition makes it possible to verify whether along this path there are other foods that are closer to be reached

3.2 MAP AND FRAGMENT MAP

3.2.1 STRUCTURES

- **Fragment Map:** Portion of the map represented by a plane of dimension (X, 1, Z).
- **Fragment Graph:** Each fragment map is associated with a graph generated starting from the matrix given by the dimensions of the plane (Matrix [X, Z])
- **Map:** Set of all fragment maps
- **Main Graph:** Graph given by the union of all fragment graphs
- **Abstract Graph**: Hierarchical graph generated starting from the main Graph and the fragment Graphs.

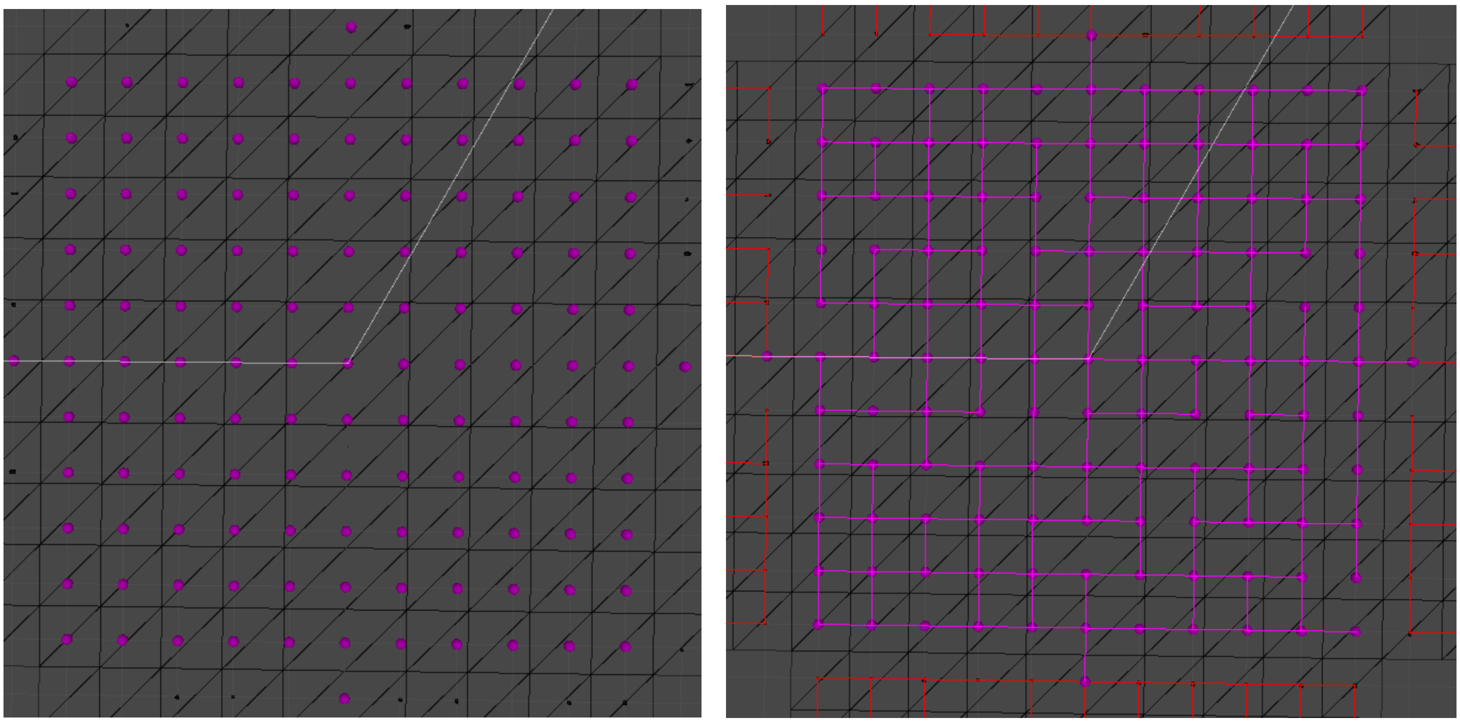


** Main Graph **Abstract Graph **Fragment graph

Properties:

- **graph topology**: mesh
- **Edge**: bidirectional
- **Edge probability**: the probability has a range that goes from 50 to 100%, this is to try to avoid having too many unreachable partitioned graphs.

Each fragment graph is composed of XxZ nodes (where X, Z is the X and Z axis dimension of the plane) plus 1 node for each path to each adjacent graph. It is possible to have only one path for each adjacent graph.



XxZ nodes + 4 node for each path to each adjacent graph

3.2.2 GRAPH GENERATOR

Starting from a single plane, the first fragment graph is generated. To do this a matrix is generated starting from the x, z dimensions of the plane. For each coordinate of the matrix, a node of the graph is assigned. Each node is associated with a game object with the corresponding position on the plane.

Instantiated game objects can be of two types:

- **Pin**: Empty Game Object that used to identify the location of the associated node.
- **Food**: Game Object that identifies the food that NPCs can eat. There is a probability to be instantiated.

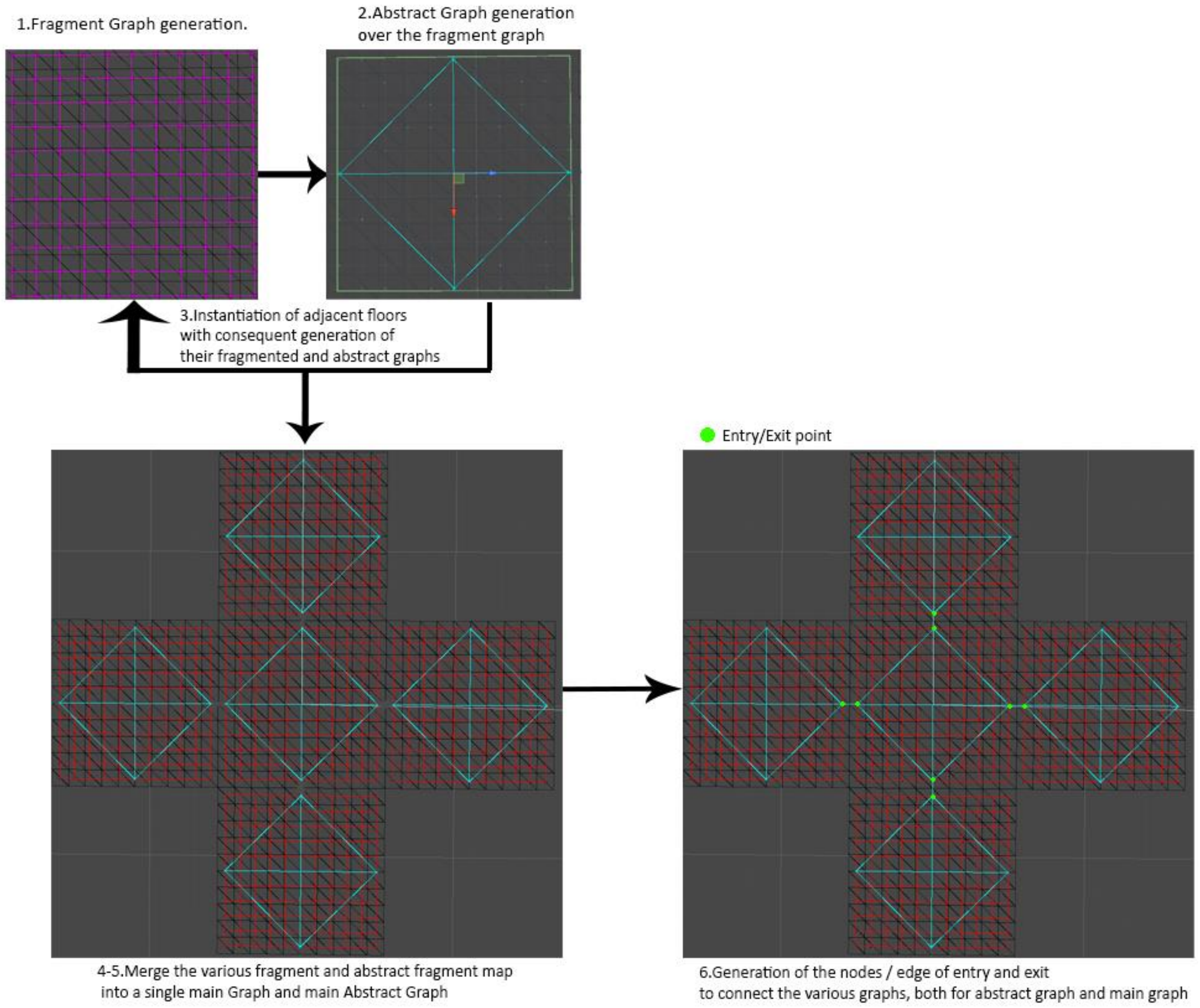
Then a first abstract graph is generated by identifying the 4 exit / entry points of the plan and analyzing if there is at least one path from each exit / entry point to any other exit / entry point, if so, an edge is associated with this pair. At this point we will have a fragmented graph and the associated abstract graph. Then the adjacent floors will be generated, and the same logic will be reapplied.

Each time a fragment map is generated, the adjacent fragment maps are associated with it by saving them in a game object structure present in the component of the created fragment map. So, each fragmented graph will be merged into a single main graph, and each abstract graph will be merged into a single main abstract graph.

At this point, the various exit points must be connected to the entry points of the adjacent graphs. Since each fragment map saves the game object of the various adjacent fragment maps, by analyzing them it is possible to associate the correct exit nodes with the entry nodes and update the graph. At this point the graph will be complete and ready to be used.

3.2.3 CREATION STEPS

1. Generation of the fragmented graph
2. Generation of the abstract graph for that specific fragmented graph
3. Instantiation of adjacent floors with consequent generation of their fragmented and abstract graphs
4. Merge of the various fragmented graphs into a single main graph
5. Merge of various abstract fragmented graphs into a single main graph
6. Generation of the nodes / edge of entry and exit to connect the various graphs, both for abstract graph and main graph


1.Fragment Graph generation.
2.Abstract Graph generation over the fragment graph
3.Instantiation of adjacent floors with consequent generation of their fragmented and abstract graphs
Entry/Exit point
4-5.Merge the various fragment and abstract fragment map into a single main Graph and main Abstract Graph
6.Generation of the nodes / edge of entry and exit to connect the various graphs, both for abstract graph and main graph

4 PATHFINDING

4.1 THE PROCESS

The pathfinding process is used to identify a random path or path to food. The function for both situations is equivalent. The only thing that changes is:

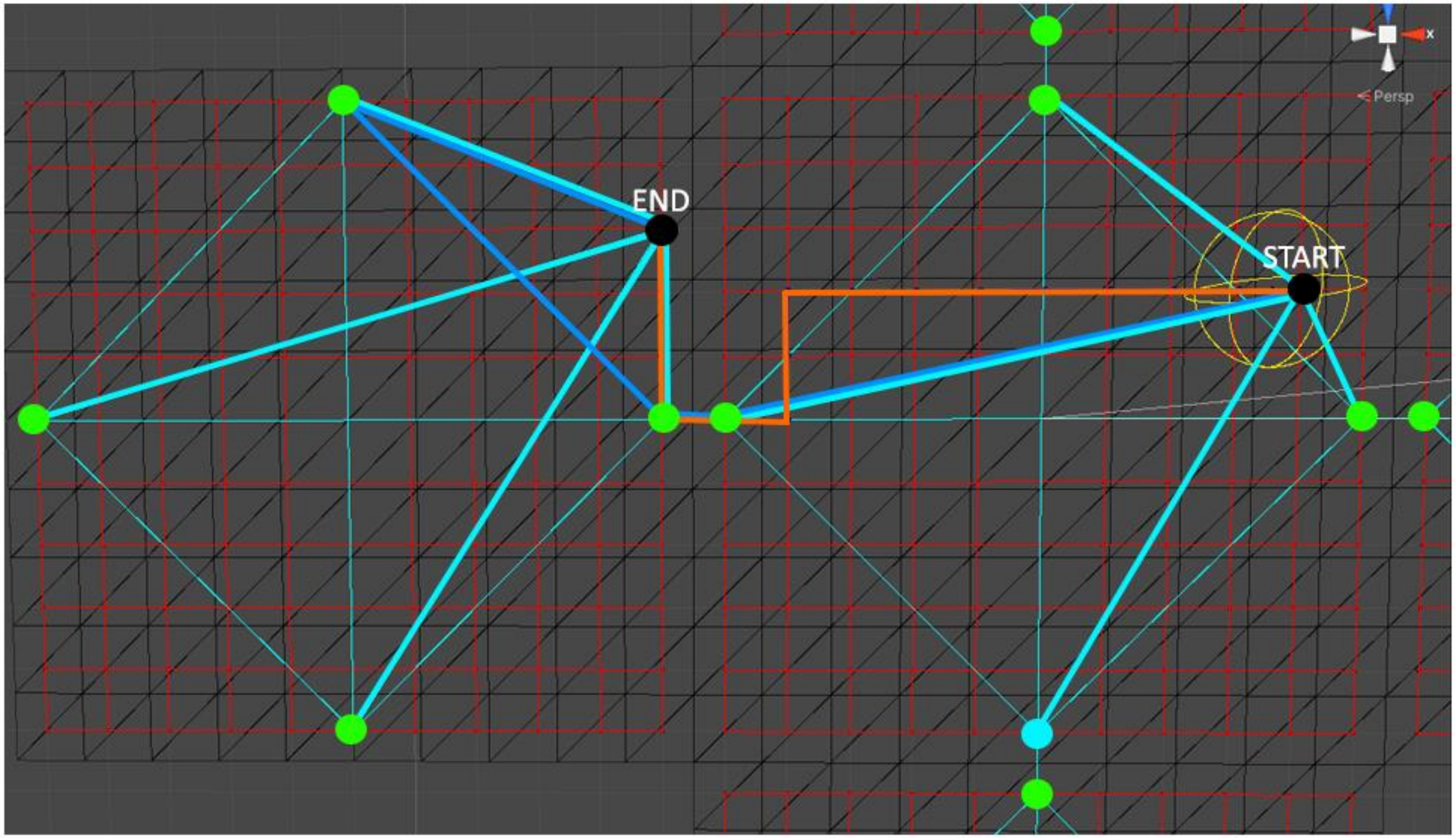
- for the random path the destination node is randomly extrapolated from the list of nodes of the main graph.
- for the path for food the destination node has already been discovered and is equivalent to the node to which a certain food is assigned.

Since the search for the route must work on a map that is constantly expanding (See paragraph 5) a very efficient system is needed. Therefore, the methodology of a hierarchical graph is used to generate a higher-level graph (abstract graph) that helps us to calculate the path.

That said, when pathfinding is performed, these are the operations that are performed(See 4.2 Section for graphical interpretation)

1. The abstract paths from the starting node to the exit nodes of the fragment graph in which the NPC is located is calculated.
2. The abstract paths from the various entry point nodes (of the fragment graph where the destination node is present) to the destination node is calculated.
3. The abstract path from the starting node to the destination node is calculated using the abstract graph containing the information calculated in points 1 and 2.
4. The abstract path calculated is used to determine each single path that connects the nodes of the abstract path. Once the individual paths have been determined, they are chained together to determine the entire level 0 path to follow.

4.2 GRAPH OF THE STEPS



Point 1-2   Point 3   Point 4

● Exit/Entry point

5 PREOCEDURAL CONTENT GENERATION

5.1 FRAGMENT MAP GENERATION (SINGLE PLANE)

Each fragment map is associated with a collider that allows you to identify when an NPC enters that portion of the map. When this happens, adjacent fragment maps are generated.

5.2 ENVIRONMENT GENERATION

To generate different environments there is a structure that associates different game objects to different sets of environments.
A graph is associated with each fragment map. When creating the graph, an environment set is determined, and an object of this set is instantiated at each node-to-node connection that is not present. Furthermore, to make the environment more alive, objects are instantiated that do not interfere with the position of the graph nodes.

For food spawning, there is a chance that it will spawn on every single node of the fragmented map.

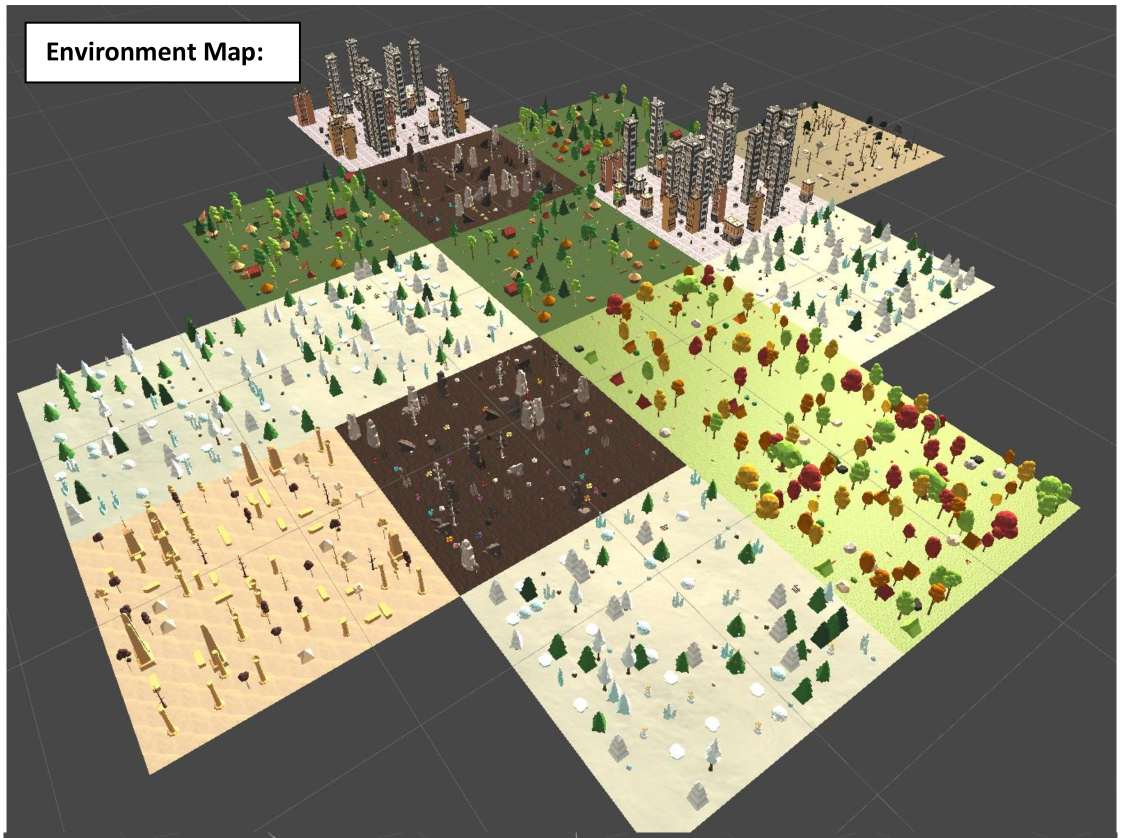
Properties:

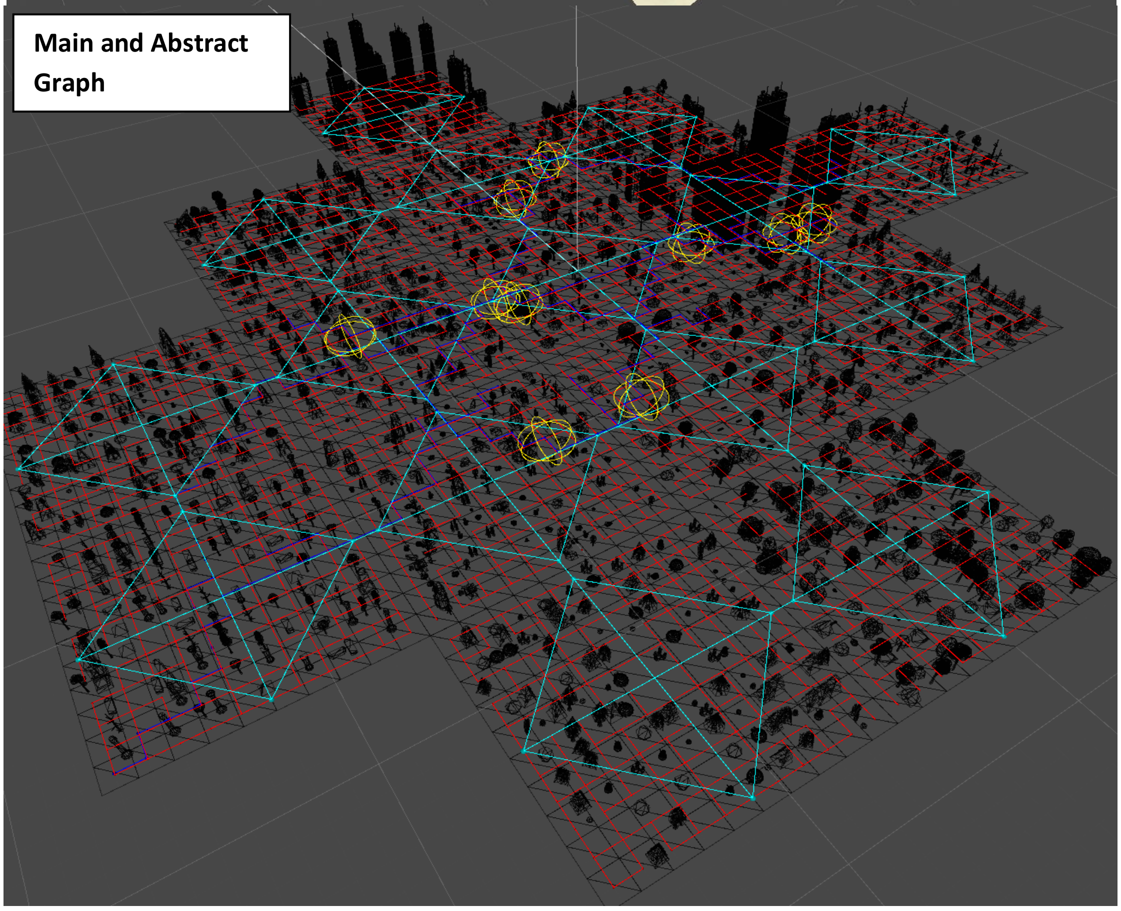
Below is the list of properties of component of the map:

- **Food Probability**: Food spawn probability ranging from 0 to 100%
- **Edge Probability**: Probability that a node is connected, ranging from 50 to 100%

5.3 RESULTS EXAMPLE

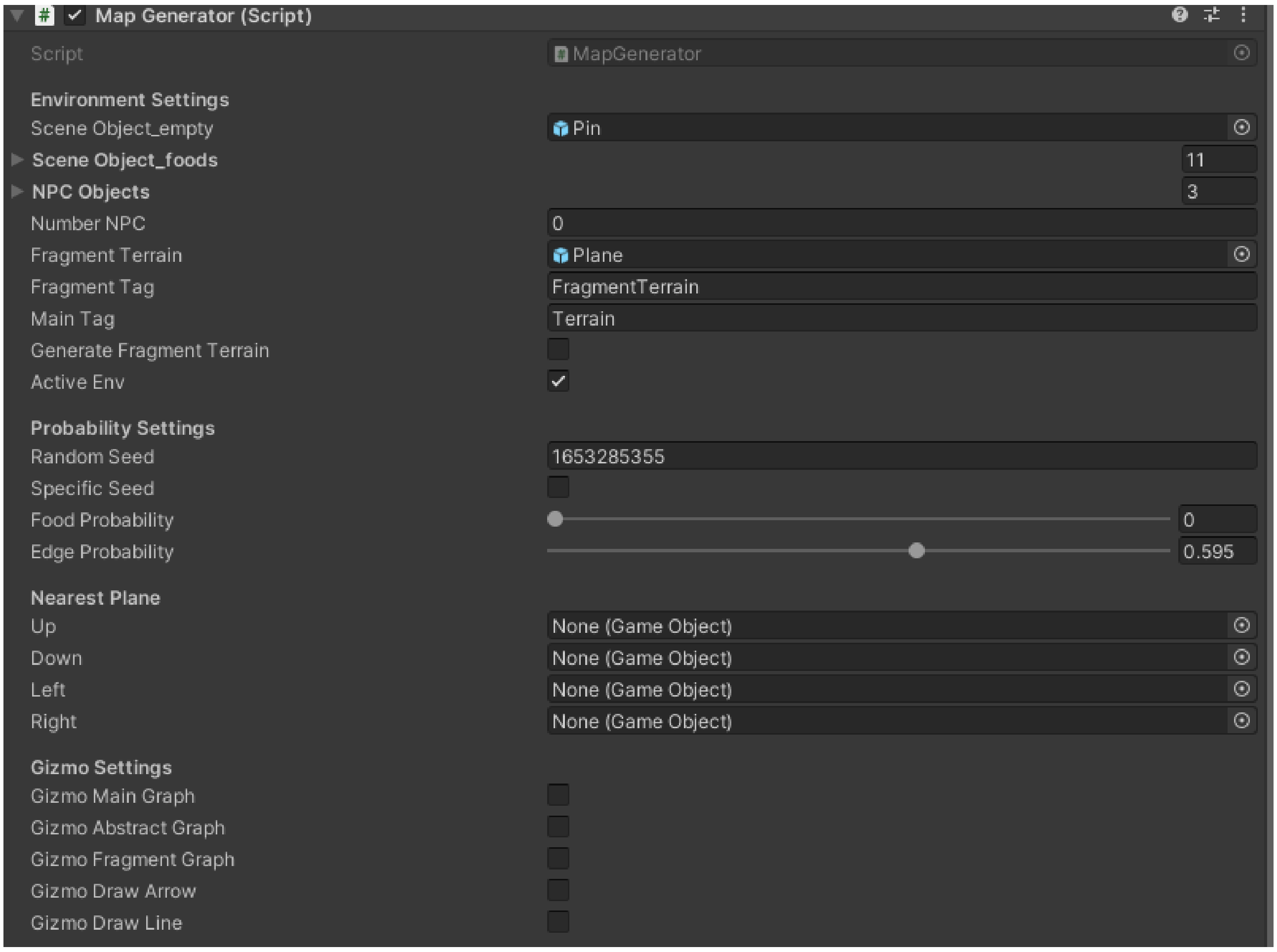
The result after some time from starting the simulation is the following:





6 COMPONENTS PROPERTIES

6.1 TERRAIN



Map Generator (Script)

Script	MapGenerator
Environment Settings	
Scene Object_empty	Pin
Scene Object_foods	11
NPC Objects	3
Number NPC	0
Fragment Terrain	Plane
Fragment Tag	FragmentTerrain
Main Tag	Terrain
Generate Fragment Terrain	☐
Active Env	✓
Probability Settings	
Random Seed	1653285355
Specific Seed	☐
Food Probability	0
Edge Probability	0.595
Nearest Plane	
Up	None (Game Object)
Down	None (Game Object)
Left	None (Game Object)
Right	None (Game Object)
Gizmo Settings	
Gizmo Main Graph	☐
Gizmo Abstract Graph	☐
Gizmo Fragment Graph	☐
Gizmo Draw Arrow	☐
Gizmo Draw Line	☐

6.1.1 ENVIRONMENT SETTINGS

- **Scene Object_Empty**: Identifies the empty object to be associated with each node of the graph.
- **Scene Object_foods**: Identify the list of foods that can be instantiated on the graph nodes.
- **Number NPC**: Number of NPCs that spawn.
- **Fragment Terrain**: Pointer to the plane associated with that component.
- **Fragment Tag**: Name to identify a fragment Terrain.
- **Main Tag**: Name To identify the main Terrain.
- **Active Env**: Used to enable or disable the environment instantiation.

6.1.2 PROBABILITY SETTINGS

- **Specific Seed**: It allows you to work on the same test case.
- **Food Probability**: Probability to spawn food.
- **Edge Probability**: Probability to have a connection between nodes.

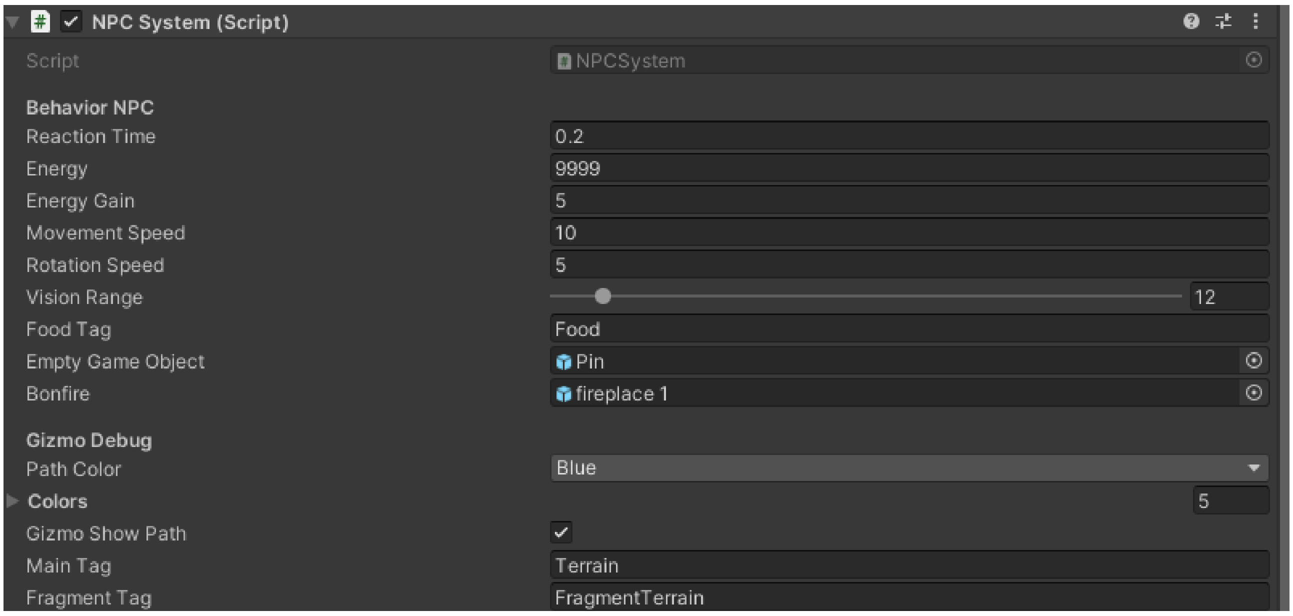
6.1.3 NEARESTPLANE

Structure to identify the floors surrounding the current floor.

6.1.4 GIZMO SETTINGS

- **Gizmo Main Graph**: Show the main Graph nodes of the map.
- **Gizmo Abstract Graph**: Show the main Abstract graph nodes of the map or single fragment Map
- **Gizmo Fragment Map**: Show the fragment Graph nodes of the selected fragment map.
- **Gizmo Draw Arrow**: Show both bidirectional edges with an arrow in the direction of the edge.
- **Gizmo Draw Line**: Show the edges of the selected Graph.

6.2 NPC



NPC System (Script)

Script	NPCSystem
Behavior NPC	
Reaction Time	0.2
Energy	9999
Energy Gain	5
Movement Speed	10
Rotation Speed	5
Vision Range	12
Food Tag	Food
Empty Game Object	Pin
Bonfire	fireplace 1
Gizmo Debug	
Path Color	Blue
Colors	5
Gizmo Show Path	✓
Main Tag	Terrain
Fragment Tag	FragmentTerrain

6.2.1 BEHAVIOR NPC

- **Reaction Time**: How often the NPC perform an action.
- **Energy**: Initial energy of the NPC.
- **Energy gain**: How much energy he takes from food.
- **Movement Speed**.
- **Rotation Speed**.
- **Vision Range**: How far can he find food.

6.2.2 GIZMO DEBUG

- **Path Color**: Different colors for the path to follow
- **Gizmo Show Path**: If checked show the gizmo of the path to follow
- **Main Tag**: Name To identify main Terrain
- **Fragment Tag:** Name to identify the fragment Terrain

6.3 CAMERA

In game it is possible to switch from third person view to top view with the **F button**

In game it is possible to switch from an NPC to another with the **E (next)** and **Q(previous)** button

7 HIEARARCHY

- Planes (fragment Terrain)
    - Environment
        - All list of instantiated objects
    - Foods
        - All list of instantiated food
- Env System: Game object that contains the structure of all group of environments
- NPCs
    - NPC Skeleton
    - Camera

8 SCRIPT

- MapGenerator.cs: It manages all the creation of the map and the environment
- NPCsCamera.cs: It manages the camera control
- NPCSystem.cs: It manages the FSM of the NPC.
- NPCMemory.cs: It manages the calculated path
- HPASolver.cs: It manages the pathfinding