WRITE UP

1. Botones para seleccionar tipo de búsqueda

Se añadió botones adicionales en la parte inferior de la pantalla, para que el usuario seleccione el tipo de búsqueda que desee realizar una vez el pacman está en ejecución, o también para volver al menú principal.

Figura 1. *Botones que permiten seleccionar el tipo de búsqueda.*

2. Armando el Árbol de búsqueda

Para que sea posible armar el árbol de búsqueda, se implementó la clase 'Node', la cual contiene procedimientos que permiten esta tarea, añadiendo información a cada nodo, los cuales vendrían a ser las galletas del laberinto.

```
3 public class Node{
4     public var father : Node = null;
5     public var children = new Array();
6     public var name : String;
7     public var content : GameObject;
8     public var visited : int = 0;
9
10
11    function getVisited(){
12        return visited;
13    }
14
15    function setVisited(i : int){
16        visited = i;
17    }
18
19    function Node(g : GameObject){
20        content = g;
21    }
22
23
24    function setContent(g : GameObject){
25        content = g;
26    }
```

Figura 2. *Extracto de código de la clase 'Node'.*

A continuación, en el script 'Search', se tienen procedimientos importantes para que se arme el árbol iniciando como raíz con el PACMAN, para ello se implementaron varias funciones como 'getNeighbors', la cual permite encontrar las galletas vecinas a una galleta en específico, también está la función 'nodeExists', la cual comprueba si un nodo ya fue añadido por otro, finalmente tenemos la función 'checkExpanded' la cual comprueba si un nodo ya fue expandido o no.

```
42 //DEVUELVE ARRAY DE TODOAS LAS CARNES VECINAS
43 function  getNeighbors(center : Vector3){
44     var nodesNeighbors= new Array();
45     var hitColliders = Physics.OverlapSphere(center, 10);
46     for (var i = 0; i < hitColliders.Length; i++) {
47         if(hitColliders[i].tag=="Meat" && hitColliders[i].gameObject.transform.position != center){
48             nodesNeighbors.Push(hitColliders[i].gameObject);
49         }
50     }
51     for(i = 0; i < nodesNeighbors.length ; i++){
52         var a  : GameObject = nodesNeighbors[i] as GameObject;
53     }
54     return nodesNeighbors;
55 }
```

Figura 3. *Código de la función 'getNeighbors'.*

```
108 function nodeExists(name : String){
109     var node : Node;
110     for(var i =0 ; i < childrenAdded.length ; i++){
111         node = childrenAdded[i] as Node;
112         if(node.getName()==name)
113         return true;
114     }
115     return false;
116 }
```

Figura 4. *Código de la función 'nodeExists'.*

```
120 function checkExpanded(name : String){
121     var node : Node;
122     for(var i =0 ; i < tree.length ; i++){
123         node = tree[i] as Node;
124         if(node.getName()==name)
125         return true;
126     }
127     return false;
128 }
```

Figura 5. *Código de la función 'checkExpanded'.*

Una vez que se selecciona un tipo de búsqueda, en pantalla se puede ver como se va armando el árbol a medida que las galletas van cambiando de color.

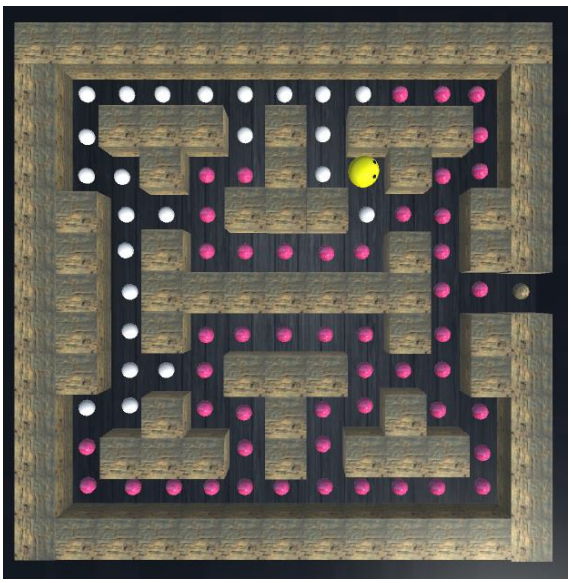

Figura 6. *Ejemplo del cambio de color de galletas mientras se arma el árbol.*

3. Búsqueda por DFS, BFS o Heurística A*

Según la búsqueda seleccionada por el usuario, se armará el árbol ya sea por DFS o BFS, para ello existen dos funciones que permiten dicha tarea, 'makeTreeBFS', para armar el árbol BFS , 'makeTreeDFS' para armar el árbol DFS y la función 'makeTreeAStart' para armar el árbol con heurística.

```
59 function makeTreeBFS(){
60
61     var tmp : Node;
62     var neighbor : GameObject;
63     var child : Node;
64     var meats : int = GameObject.FindGameObjectsWithTag("Meat").Length;;
65
66     children.Push(nodeS);
67     while(meats > 0){
68         while( children.length>0 ){
69             child = children.Pop() as Node;
70
71             if(checkExpanded(child.getName())){
72                 continue;
73             }
74
75             actualPosition = child.getContent().transform.position;
76             neighbors = getNeighbors(actualPosition);
77
78             while(neighbors.length > 0) {
79                     neighbor = neighbors.Pop() as GameObject;
80                     if(!nodeExists(neighbor.transform.name)){
81                         tmp = new Node(neighbor);
82                         tmp.setName(neighbor.transform.name);
83                         tmp.setVisited(1);
84                         tmp.setFather(child);
85                         child.addChild(tmp);
86                         childrenAdded.Push(tmp);
87                         neighbor.GetComponent.<Renderer>().material.mainTexture = null;
88                         yield WaitForSeconds (0.1);
89                         toExpand.Push(tmp);
90                 }
```

Figura 7. *Extracto de código de la función 'makeTreeBFS'.*

```
184 function makeTreeDFS(){
185     var tmp : Node;
186     var neighbor : GameObject;
187     var child : Node;
188     var meats : int = GameObject.FindGameObjectsWithTag("Meat").Length;
189 
190 
191 
192     children.Unshift(nodeS);
193 
194     while(meats > 0){
195         while( children.length>0 ){
196             child = children.Shift() as Node;
197 
198             if(checkExpanded(child.getName())){
199                 continue;
200             }
201 
202             actualPosition = child.getContent().transform.position;
203             neighbors = getNeighbors(actualPosition);
204 
205             while(neighbors.length > 0) {
206                     neighbor = neighbors.Shift() as GameObject;
207                     if(!nodeExists(neighbor.transform.name)){
208                         tmp = new Node(neighbor);
209                         tmp.setName(neighbor.transform.name);
210                         tmp.setVisited(1);
211                         tmp.setFather(child);
212                         child.addChild(tmp);
213                         childrenAdded.Unshift(tmp);
214                         neighbor.GetComponent.<Renderer>().material.mainTexture = null;
```

Figura 8. *Extracto de código de la función 'makeTreeDFS'.*

Para armar el árbol con heurística, primero fue necesario crear la función 'sortByHeuristic', la cual recibe como parámetro un array de nodos vecinos a cada nodo a medida que el árbol se va armando.

```
8 // Sorts array by heuristcs
9 function sortByHeuristic(n: Array){
0     var flag = false;
1     var l = new Array();
2     var tmp;
3     var count=0;
4 
5     while(!flag){
6         for(var i = 0 ; i<n.length-1 ; i++){
7             var go1 = n[i] as GameObject;
8             var go2 = n[i+1] as GameObject;
9             var d1 = Vector3.Distance(go1.transform.position, nodeG.getContent().transform.position);
0             var d2 = Vector3.Distance(go2.transform.position, nodeG.getContent().transform.position);
1             if(d1 < d2){
2                 tmp = n[i];
3                 n[i] = n[i+1];
4                 n[i+1] = tmp;
5                 count++;
6             }
7         }
8         if(count==0)
9             flag = true;
0         count = 0;
1     }
2 }
```

Figura 9. *Código de la función 'sortByHeuristic'.*

Finalmente, está función se usará dentro de la función ‘makeTreeAStart’, que es la que se encargará de armar el árbol completo con heurística.

```
//Create a tree taking heuristics to make decisions about who node is expanded first
function makeTreeAStart(){
    var tmp : Node;
    var neighbor : GameObject;
    var child : Node;
    var meats : int = GameObject.FindGameObjectsWithTag("Meat").Length;

    children.Unshift(nodeS);

    while(meats > 0){
        while( children.length>0 ){
            child = children.Shift() as Node;
            if(checkExpanded(child.getName())){
                continue;
            }
            actualPosition = child.getContent().transform.position;
            neighbors = getNeighbors(actualPosition);
            sortByHeuristic(neighbors);

            while(neighbors.length > 0) {
                    neighbor = neighbors.Shift() as GameObject;
                    if(!nodeExists(neighbor.transform.name)){
                        tmp = new Node(neighbor);
                        tmp.setName(neighbor.transform.name);
                        tmp.setVisited(1);
                        tmp.setFather(child);
                        child.addChild(tmp);
                        childrenAdded.Unshift(tmp);
                        neighbor.GetComponent.<Renderer>().material.mainTexture = null;
                        Input.ResetInputAxes();
                        if(steps){
                            while(!Input.GetKeyDown("space"))
                                yield;
                        }else{
                            yield WaitForSeconds (0.1);
                        }
                        children.Unshift(tmp);
                }
```

Figura 10. *Extracto de código de la función ‘makeTreeAStart’.*

4. Encontrando la galleta de salida

Se colocó una galleta meta en la salida del laberinto, esto es para que el pacman arme el camino para llegar desde su posición inicial hacia la salida. Para ello utilizamos la función ‘getPath’, a la cual le mandamos como parámetro el nodo meta, es decir, la galleta de salida, la función construirá el camino desde la salida hacia el pacman.

```
function getPath(goal : Node) {
    var father : Node = goal;
    var path = new Array();

    while(true){
        path.Push(father);
        if(!father.getFather()){
            return path.Reverse();
        }
        father = father.getFather();
    }
}
```

Figura 11. *Código de la función ‘getPath’.*

5. Movimiento del pacman

Una vez que se encontró el camino para llegar a la salida, se creó la función ‘exit’, la cual recibe como parámetro el camino obtenido del paso anterior. La función se encargará de realizar el movimiento del pacman desde su origen hacia la meta.

```
function exit(path : Array) {

    var node : Node;
    var go : GameObject;

    node = path[CurrentNodeIndex] as Node;
    go = node.getContent() as GameObject;


    if (!Mathf.Approximately(
            Vector3.Distance(nodeS.getContent().transform.position, go.transform.position),
            0
    )) {
        getDirection(nodeS, node);
        nodeS.getContent().transform.position = Vector3.MoveTowards(nodeS.getContent().transform.position, go.transform.position, 1);

    } else {
        CurrentNodeIndex++;
        Debug.Log("Starting to move towards next node: " + CurrentNodeIndex);

        if (CurrentNodeIndex >= path.length) {

            makePathBFS = false;
            makePathDFS = false;
            return;
        }
    }
}
```

Figura 12. *Código de la función ‘exit’.*

Figura 13. *Ejemplo del pacman llegando a la salida mediante BFS.*

6. Coloreando camino de salida

Adicionalmente, se implementó una función para que cambie el color de las galletas que el pacman deberá seguir para llegar a la salida.

```
146 //Paints the path from pacman to goal
147 function paintPath(path : Array){
148     var node : Node;
149     var go : GameObject;
150     for(var i=1 ; i < path.length ; i++ ){
151         node = path[i] as Node;
152         go = node.getContent() as GameObject;
153         go.GetComponent.<Renderer>().material.mainTexture = waterTexture;
154     }
155 }
```

Figura 14. *Extracto de código de la función 'paintPath'.*

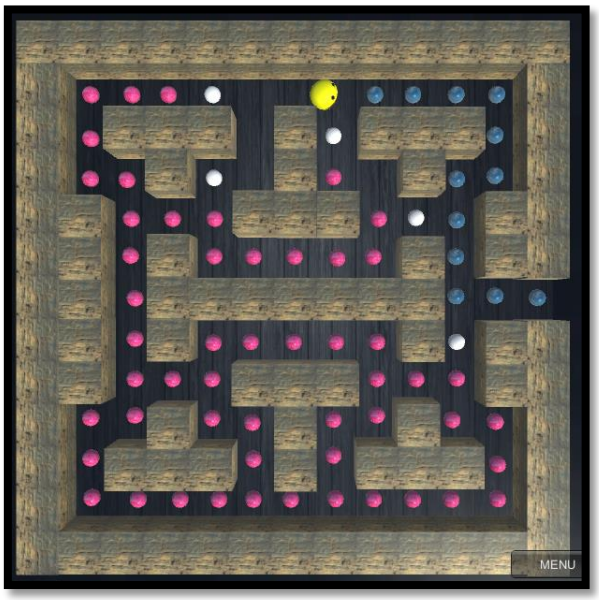

Figura 15. *Galletas del camino de salida con color azul.*

6. Botones de pausa y continuar

Con el fin de permitir que el usuario pueda pausar el juego mientras se arma el árbol, se añadieron los botones de 'steps' y 'continue', para que cada vez que presione la barra espaciadora, se vea en pantalla la siguiente galleta que se va añadiendo al árbol.

```
if(steps){
    while(!Input.GetKeyDown("space"))
        yield;
}else{
    yield WaitForSeconds (0.1);
}
children.Unshift(tmp);
```

Figura 16. *Extracto de código que permite la funcionalidad de los botones 'steps' y 'continue'.*