**Agent.cs**

Overview:

* Controls the functions of a singular agent in the scene

Variables:

* For pathfinding calculations
  + cameFrom: dictionary of Cell, Cell to keep track of where the parent of each Cell in the pathToTake list
  + costSoFar: dictionary of Cell, float to keep track of how much the path being calculated costs
  + tcameFrom: same as cameFrom, but for a dictionary of Threshold, Threshold
  + tcostSoFar: same as costSoFar, but for a dictionary of Threshold, float
  + pathToTake: list of Cells that create a path from the agent’s starting position to its goal position
  + tpathToTake: same as pathToTake, but as a list of Thresholds
* For updating pathfinding
  + isCalculatingPath: is the agent calculating a path?
  + hasFoundPath: has the agent calculated its path?
  + hasHaltedMovement: has the agent been told to stop moving?
  + hasHaltedCalculating: has the agent been told to stop calculating its path?
  + updateRate: the rate in which the agent will re-calculate its path, in terms of steps
* For movement
  + moveSpeed: the speed at which the agent will move
  + stepsTaken: the number of steps the agent has taken so far
  + Timer: a Timer in Unity to help move the agent at a particular speed
  + nextCell: the next Cell that the agent must move to in its pathToTake

References:

* goal
  + Reference to the agent’s goal in Unity
* map
  + Reference to the Map class

Functions:

* Start
  + Unity’s built-in function
  + Sets up the agent by getting a reference to the Map class, and assigning temporary assignments
  + Lastly calls FindHPAPath to find a path for the agent
* LateUpdate
  + Unity’s built-in function
  + Used for moving the agent from its origin to its goal if it has found a path and is allowed to move
  + 1. if hasFoundPath && !hasHaltedMovement
    - Move the agent through its calculated path
    - 3. nextCell = pathToTake[stepsTaken].worldposition
    - 4. If agent’s current position != nextCell
      * Move the agent to the nextCell
    - 5. Else, check if the agent is at the end of its path
      * 6. If stepsTaken < pathToTake.Count - 1
        + 7. stepsTaken++
        + 8. CheckCell()
        + Check if the agent should recalculate its path
        + 9. if updateRate != 0

10. If stepsTaken % updateRate == 0 && stepsTaken != 0

Recalculate the path

11. pathToTake = FindHPAPath(current agent’s position, goal position)

* CheckCell
  + Helper function used to move the agent
  + 1. nextCell = pathToTake[stepsTaken].position
* HaltMovement
  + Halts the agent from moving from its path simply by changing the hasHaltedMovement
* HaltCalculating
  + Halts the agent from calculating its path by changing the hasHaltedCalculating
* ChangeUpdateRate
  + Change the at which the agent recalculated the path
* WaitUntilCanCalculate
  + Helper function using coroutines to allow the agent to halt path calculation
* FindHPAPath
  + Find a path from the agent’s starting location to the desired goal location. This uses HPA\* algorithm
  + Input: starting location of the agent, goal location
  + Output: the calculated path as a list of cells the agent must visit to get to the goal
  + 1. hasFoundPath = false
  + 2. isCalculatingPath = true
  + 3. finalPath = new List of Cells, startCell = Cell at the agent’s starting position, goalCell = Cell at the goal position
  + 4. if the start is already in the same zone as the goal
    - Simply use the findCellPath to get to the goal
    - 5. tmp = findCellPath(starting Cell, goal Cell)
    - 6. finalPath.Add(tmp)
    - 7. pathToTake = finalOath
    - 8. hasFoundPath = true
    - 9. isCalculatingPath = false
    - 10. Return finalPath
  + 11. Add the start position and the goal position to the Threshold graph by finding which Threshold is close to the two positions
    - thresholdStart = FindNearestThreshold(start, goal)
    - thresholdGoal = FindNearestThreshold(goal, start)
  + 12. Find a path of thresholds starting from thresholdStart and goes to thresholdGoal
    - thresholdPath = FindThresholdPath(start, goal)
    - tPathToTake = thresholdPath
  + 13. Using the threshold path, find a path to the goal
    - nextGoal = map.CellFromThreshold(thresholdStart)
    - temp = FindCellPath(startCell, nextGoal)
    - finalPath.Add(temp)
  + Find the path between the thresholds
  + 14. For (int i = 1, i < thresholdPath.count, i++)
    - Get the last position from the finalPath, which will be the starting position of the next part of the path needed to find. The same is for the goal
    - 15. newStart = map.CellFromThreshold(thresholdPath[i-1])
    - 16. newGoal = map.CellFromThreshold(thresholdPath[i])
    - 17. temp = FindCellPath(newStart, newGoal)
    - 18. finalPath.add(temp)
  + Find the final part of the path
  + 19. nextGoal = map.CellFromThreshold(thresholdPath[thresholdPath.Count - 1])
  + 20. temp = findCellPath(nextGoal, goalCell)
  + 21. finalPath.add(temp)
  + 22. Done!
    - pathToTake = finalPath
    - hasFoundPath = true
    - isCalculatingPath = false
    - Return finalPath
* FindNearestThreshold
  + Given a position, find the Threshold that is near the both the beginning position and the goal position
  + Input: position in question, goal position
  + Output: threshold hat is nearest to those position
  + Find the zone that the starting position belongs to
  + 1. currentCell = map.CellFromWorldPos(starting position)
  + 2. goalCell = mmap.CellFromWorldPos(goal position)
  + 3. zone = map.GetZone(currentCell.zoneID)
  + Find the Threshold that is closest to the goal
  + 4. cost = infinity, i = 0
  + 5. Foreach (Threshold t in zone.Thresholds)
    - 6. Temp = getCellCost(goalCell, t) + getCellCost(currentCell, t)
    - 7. If (temp <= cost)
      * 8. Cost = temp
      * 9. Threshold threshold = t
      * 10. I++
  + 11. Return threshold
* FindCellPath
  + Find a path from the agent’s starting location to the desired goal location. This uses the A\* algorithm
  + Input: start Cell, goal Cell
  + Output: the calculated path as a list of Cells the agent must visit to get to the goal
  + 1. frontier = new priority queue of Cells
  + 2. path = new list of Cells
  + 3. frontier.enqueue(startCell, 0)
  + 4. cameFrom[startCell] = startCell
  + 5. costSoFar[startCell] = 0
  + 6. temp = 0
  + 7. while (frontier.count > 0)
    - 8. Temp++
    - 9. If (hasHaltedCalculating)
      * Create a coroutine to pause this calculation
      * startCoroutine(WaitUntilCanCalculate())
    - 10. current = frontier.Dequeue
    - 11. if the current Cell is actually the goal Cell
      * We have found the goal! Return the path found
      * path = retraceCellPath(startCell, goalCell)
      * Return path
    - Else, find the shortest path to the goal by checking the neighbors of the current Cell
    - 12. Foreach (Edge e in current.edgeszToNeighbors)
      * 13. neighbor = e.incident
      * 14. newCost = costSoFar[current] + GetCellCost(current, neighbor)
      * 15. If (!costSoFar.contains(neighbor) || newCost < costSoFar[neighbor])
        + This neighbor leads to a shorter path to the goal. Update out costs and path in order to consider this neighbor as part of the path
        + 16. costSoFar[neighbor] = newCost
        + 17. priority = newCost + GetCellHeuristic(neighbor, goalCell)
        + 18. frontier.enqueue(neighbor, priority)
        + 19. cameFrom[neighbor] = current
  + 20. Done! Return path
* RetraceCellPath
  + Given the start cell and the goal cell of the agent, give calculated path using the cells that are in cameFrom.
  + Input: start Cell, goal Cell.
  + Output: the calculated path as a list of cells the agent must visit to get to the goal.
  + Loop through the cameFrom dictionary
  + 1. path = new list of Cells
  + 2. currentCell = goal Cell
  + 3. While (currentCell != startCell)
    - 4. path.add(currentCell)
    - 5. currentCell = cameFrom[currentCell]
  + 6. path.add(startCell)
  + 7. Path.reverse()
  + 8. Done! Return path
* FindThresholdPath
  + Same as FindCellPath, the algorithm has not changed. But now it works with Thresholds rather than Cells.
  + cameFrom, costSoFar, and pathToTake will use their Threshold data type versions, tcameFrom, tcostSoFar, etc
* RetraceThresholdPath
  + Same as RetraceCellPath, the algorithm has not changed. But now it works with Thresholds rather than Cells.
  + cameFrom, costSoFar, and pathToTake will use their Threshold data type versions, tcameFrom, tcostSoFar, etc
* GetCellCost
  + Calculate the cost from one Cell to the next
  + This currently is the distance from the source Cell to the sink Cell
  + Input: source Cell, sink Cell
  + Output: cost
* GetCellHeuristic
  + Calculate the heuristic cost from one Cell to the next
  + This currently is the distance from the source Cell to the sink Cell
  + Input: source Cell, sink Cell
  + Output: cost
* GetThresholdCost
  + Same as GetCellCost, but for source Threshold and sink Threshold
* GetThresholdHeuristic
  + Same as GetCellHeuristic, but for source Threshold and sink Threshold
* OnDrawGizmos
  + Used for debugging to draw the Agent’s calculated path in the Scene view tab of Unity