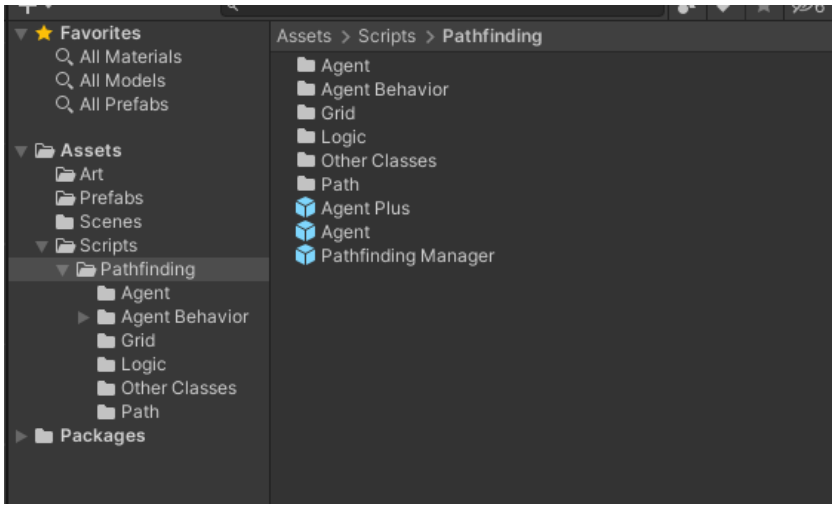
TOP-DOWN AI DOCUMENTATION

By Atomic Potato Games

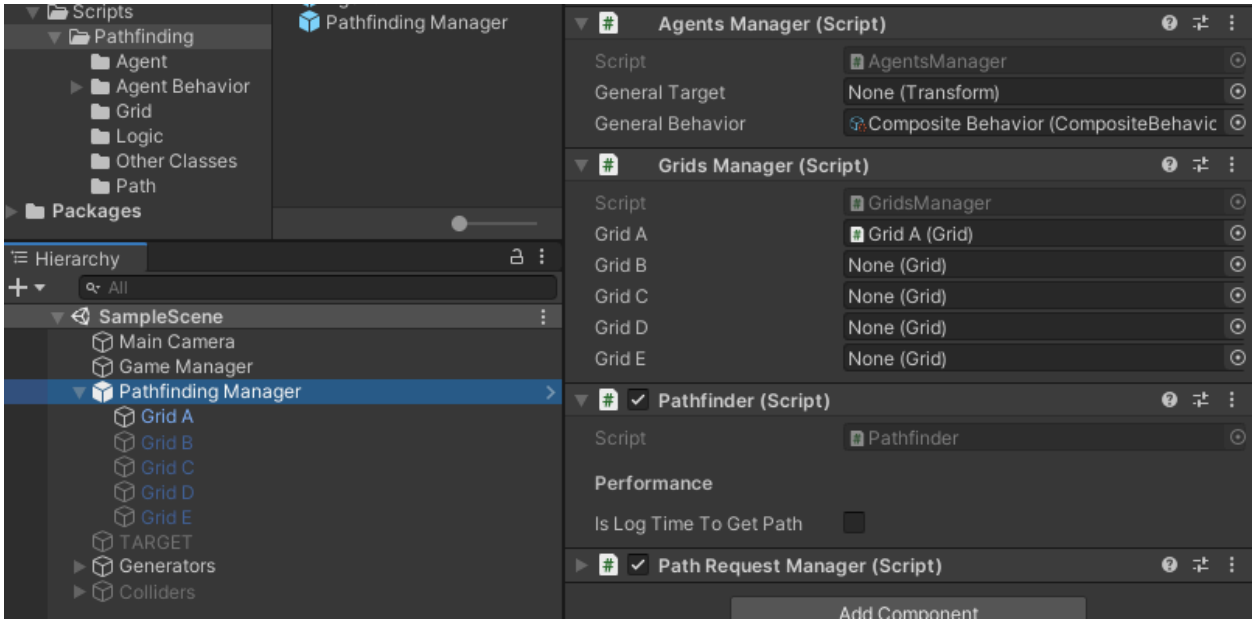
This project was based on Sebastian Lague implementation of A. Further info:*
https://www.youtube.com/@SebastianLague
https://github.com/SebLague/Pathfinding

Usage

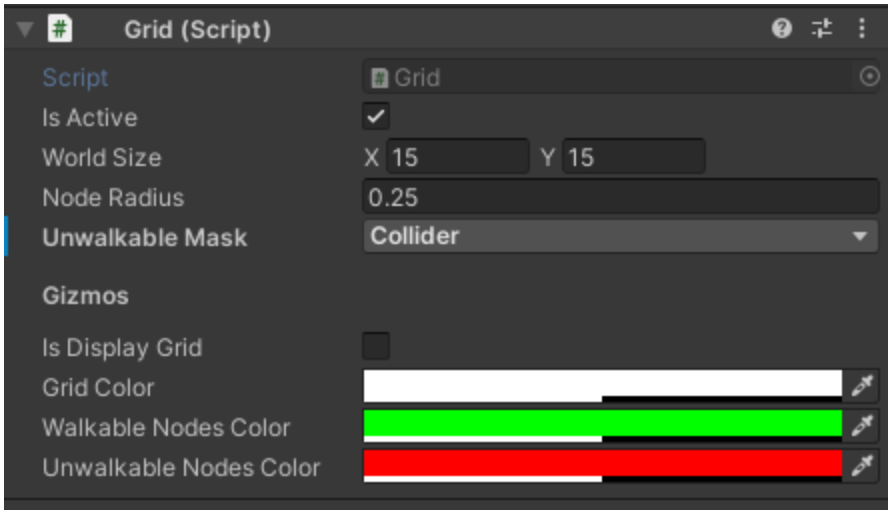
1. Head to the Pathfinding folder and add the Pathfinding Manager Object to your scene.



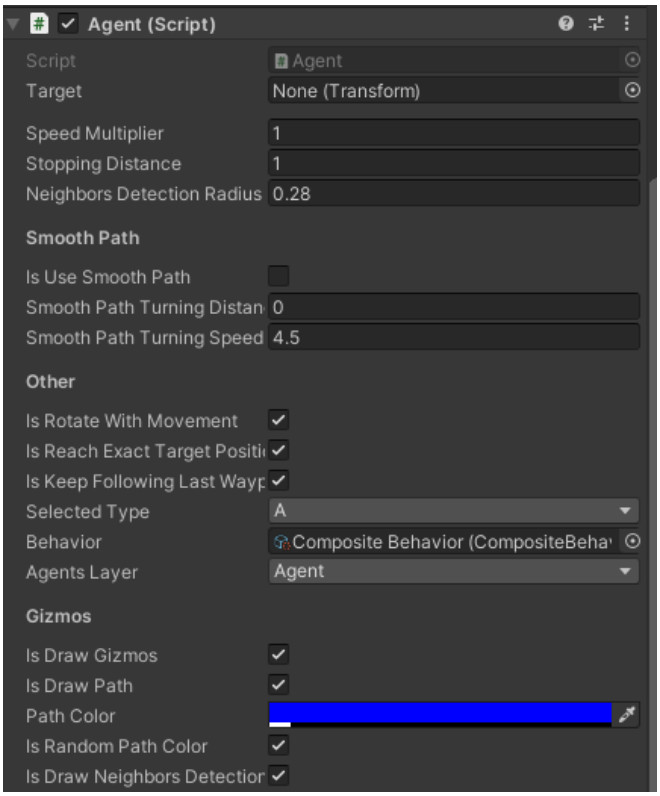
2. By default Grid A (child of the Pathfinding Manager) is set up in the Grids Manager (attached to the Pathfinding Manager object).
Feel free to add extra grids to the grids manager by dragging one of the child grid objects to the corresponding parameter in the grids manager.



3. For each grid added
    a. Set the **World Size** to be scanned. In other words the area which the AI agent will be able to navigate (Tip: enable **Is Display Grid** to visualize the area)
    b. For each grid set the **Unwalkable Mask** layer, which is the colliders layer that the grid will mark areas on as unwalkable by the AI agents.



4. Create an agent:
    a. **OPTION A:** Create an empty game object and add to it a 2D collider, Agent, Rigidbody2D *(optional)* components
    b. **OPTION B:** Drag either Agent or Agent Plus prefabs onto the scene



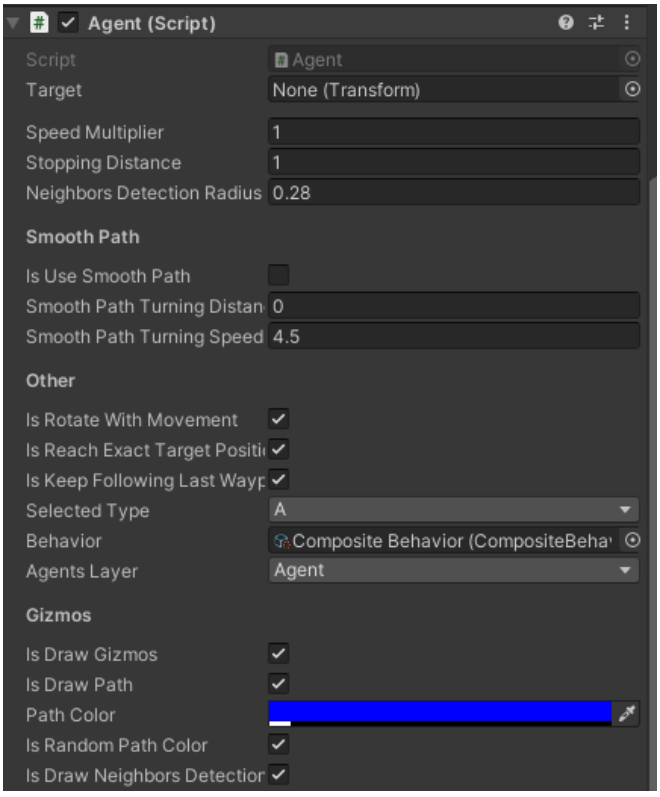
5. Set a target for the agent from either the Agents Manager, agent Target Property, or via script.
6. (OPTIONAL) Set a behavior for the Agent either from the Agents Manager or from the Agent. Composite behavior is recommended which includes the follow path and avoidance behaviors.

Features

Tip: Hover over variable names to get a description of what they do

Agent

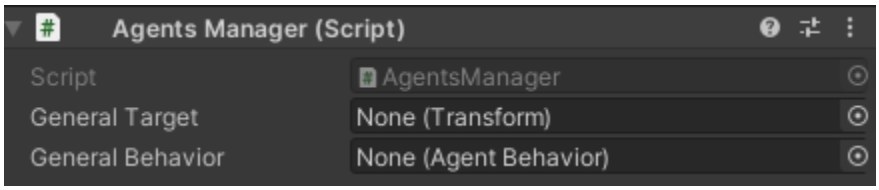
The agent class to be attached to a game object for it to be able to follow a target on grid path



- **Smooth Path:** By default the agent will follow a straight path to the target and make harsh turns. Smooth path will curve turns.
- **Selected Type:** Currently only picks which grid to use when traversing the world. Having different grids can be useful if you have agents with different sizes, so you can make a grid with a larger node size for that agent.
- **Behavior:** By default, it will be a composite behavior. Behaviors specify how the agent will move, will it follow a path, will it be affected by other agents and avoid them.

Agents Manager

Contains a list of all the agents, and acts as a container for common values between all agents. From here you can set a **General Target** and **General Behavior** which will set the target and behavior values for all the agents in the scene at the start of the game.



Agent Behaviors

An agent behavior affects how the agent will move. It calculates a velocity and returns it to the agent class to be added to the current velocity of the agent.
By default, agents are all assigned the Composite Behavior

Follow Path Behavior

Calculates the velocity to follow a path, either smooth or straight path.

Avoidance Behavior

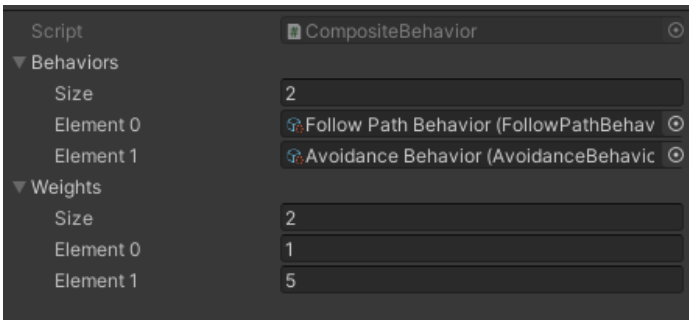
A basic behavior to avoid other agents in the path preventing any overlap with other agents. It is based on BOIDs (flock simulation)

For the agent to avoid other agents, you must set a **Neighbors Detection Radius** value, once an agent is inside this radius they will be avoided. As well as the **Agents Layer**.
Note: Neighbors detection radius must not be smaller than the agent's collider

If set in a Composite Behavior, the recommended value for a weight is of **5 or above** for strong avoidance, a **value of 1** for a small overlap.

Composite Behavior

Holds multiple behaviors with a weight assigned to each behavior, then returns the average velocity from the combination of these behaviors multiplied by the weight.



Grids Manager

Contains the grids that can be used for agents pathfinding.
It is useful to have multiple grids with different node sizes to account for agents with different collider sizes.

Grid

The grid which the agent will traverse.

Pathfinder

Handles the calculation of the A* algorithm which finds the shortest path between 2 nodes on a grid
(Further info: https://youtu.be/-L-WgKMFuhE?si=3rsrBzKm9MGdusFL)

Path Request Manager

Handles path requests on the pathfinder.

Helpful Methods & Functions

- **Updating the grid (Grid):** To make a dynamic world grid when a certain path is blocked or opened by a collider that is generated during runtime and not the start of the game. The following function updates a portion of the grid to save on performance.

```
/// <summary>
/// Updates the nodes IsWalkable property in a provided rectangular area
/// </summary>
/// <param name="bottomLeftCornerPosition">The bottom left edge world position of the rectangular area to be updated</param>
/// <param name="topRightCornerPosition">The top left edge world position of the rectangular area to be updated</param>
0 references
public void UpdateGridSection(Vector2 bottomLeftCornerPosition, Vector2 topRightCornerPosition)
{
    Node nodeBottomLeft = GetNodeFromWorldPosition(bottomLeftCornerPosition);
    Node nodeTopRight = GetNodeFromWorldPosition(topRightCornerPosition);

    for (int i = nodeBottomLeft.GridPositionX; i <= nodeTopRight.GridPositionX; i++)
    {
        for (int j = nodeBottomLeft.GridPositionY; j <= nodeTopRight.GridPositionY; j++)
        {
            Nodes[i,j].IsWalkable = !Physics2D.OverlapBox(Nodes[i,j].WorldPosition, new Vector2(_nodeDiameter, _nodeDiameter), 0f, _unwalkableMask)
        }
    }
}
```

Another option for updating the entire grid also exists:

```
/// <summary>
/// Updates the nodes IsWalkable property for the entire grid
/// </summary>
0 references
public void UpdateGrid()
{

    for (int i = 0; i <= NodesCountX - 1; i++)
    {
        for (int j = 0; j <= NodesCountY - 1; j++)
        {
            Nodes[i,j].IsWalkable = !Physics2D.OverlapBox(Nodes[i,j].WorldPosition, new Vector2(_nodeDiameter, _nodeDiameter), 0f, _unwalkableMask)
        }
    }
}
```

- **Getting agent neighbors (Agent):** Returns a list of all agents that are within the **Neighbors detection radius.**

```
/// <summary>
/// Populates and returns a list of surrounding agents that are in the agent neighbors detection radius
/// </summary>
1 reference
public List<Agent> GetNeighbors()
{
    RaycastHit2D[] hits = Physics2D.CircleCastAll(transform.position, _neighborsDetectionRadius, Vector2.zero, Mathf.Infinity, _agentsLayer);
    List<Agent> neighbors = new List<Agent>();
    foreach(RaycastHit2D hit in hits)
    {
        if (hit.collider != Collider)
            neighbors.Add(hit.collider.gameObject.GetComponent<Agent>());
    }
    return neighbors;
}
```

- **Force updating the agent path (Agent):** For cases where the grid has changed, i.e. a set of nodes has become walkable or unwalkable, this forces the agent to update the path, since **SendPathRequest** will not update the agent path unless the target node

position has changed since the last path request.

```
3 references
Coroutine _forcePathRequestCoroutine;
/// <summary>
/// Sends a request to update the path regardless of any restrictions
/// (Currently the only restriction is the target end node must be different than the last path request)
/// </summary>
0 references
public void ForceSendPathRequest()
{
    if (_pathRequestCoroutine != null)
    {
        StopCoroutine(_pathRequestCoroutine);
        _pathRequestCoroutine = null;
    }
    if (IsPathRequestSent == true)
        IsPathRequestSent = false;

    if (_forcePathRequestCoroutine == null)
        _forcePathRequestCoroutine = StartCoroutine(SendRequest());

    IEnumerator SendRequest()
    {
        if (Target == null)
            yield break;

        // Delaying the path request at the start of the game
        // since delta time is quite high at the start
        if (Time.timeSinceLevelLoad < .3f)
            yield return new WaitForSecondsRealtime(.3f);

        PathRequestManager.RequestPath(new PathRequest(transform.position, Target.position, Grid, null, UpdatePath));
        _forcePathRequestCoroutine = null;
    }
}
```