User’s manual for “PathFinder 3D”

Version 0.4.0

Сведения об обновлении v0.4.0

Это обновление примечательно тем, что мы перевели ассет на 2018 версию Unity, что позволило использовать все преимущества самой актуальной версии .NET. Теперь в качестве хранилища для пространственного графа выступает встроенный в C# неблокирующий словарь (ConcurrentDictionary), который намного быстрее всего, что мы использовали ранее. Это позволило значительно ускорить обработку препятствий. Теперь, поскольку Unity использует другой, более агрессивный компилятор, многопоточные коды стали намного эффективней. Благодаря всему вышесказанному, работа ассета значительно улучшилась. Поведение преследователей стало быстрее и плавнее. Так же мы поправили некоторые незначительные ошибки.

Общие сведения о работе ассета PathFinder 3D (v0.4.0).

“PathFinder 3D” предназначен для поиска пути (траектории) в пространстве между двумя точками. Найденный путь будет близок к минимально возможному и в большинстве случаев не будет пересекать препятствия. Для поиска пути используются два алгоритма:

1. Модификация алгоритма A* со взвешенной эвристикой;
2. Волновой алгоритм (алгоритм Ли).

Здесь и далее игровые объекты, использующие возможности поиска пути и следования по нему, будем называть *преследователями*. Пространство поиска дискретно и состоит из клеток, образующих граф поиска. Каждая клетка может быть проходимой, либо непроходимой, именно эта характеристика определяет возможность нахождения пути через нее.

Для того что бы стало возможным искать путь, необходимо исследовать игровое пространство на наличие препятствий т.е. построить граф поиска. Процесс обработки игрового пространства должен быть выполнен один раз, до того как кто-либо из преследователей захочет искать путь (рекомендуем делать это при старте сцены). Если игровая сцена меняется во время игры (например препятствия двигаются или меняют размеры и поворот),то можно пересчитывать граф поиска в окрестности тех препятствий, которые изменились. Правда, не стоит делать это слишком часто, так как процесс обработки препятствий достаточно трудоемок.

Весь основной функционал ассета содержится в двух MonoBehaviour классах - *SpaceManager* и *Pursuer*.

Класс Pursuer можно найти в папке /PathFinder 3D/Scripts/Controllers. Pursuer предназначен для поиска пути и следования по нему. Все игровые объекты, для которых подразумевается возможность поиска пути и следования по нему (преследователи), должны иметь скрипт Pursuer.cs среди своих компонентов.

Возможен сценарий, при котором на сцене будет множество преследователей. В таком случае процедура поиска пути будет вызываться достаточно часто, что может повлечь падение производительности. Для этого в классе SpaceManager организована очередь преследователей. Преследователям, находящимся в очереди, выдается разрешение на поиск

пути, таким образом, что бы количество преследователей, для которых вычисляется путь одновременно не превышало числа логических ядер процессора.

Класс SpaceManager можно найти в папке /PathFinder 3D/Scripts/SpaceProcessing. SpaceManager предназначен для обработки сцены и контроля за препятствиями, так-же он управляет очередью преследователей. Скрипт SpaceManager должен быть инициализирован единожды и присутствовать на игровой сцене в единственном экземпляре.

Препятствиями считаются игровые объекты, удовлетворяющие следующим условиям:

- На объекте присутствует активный MeshCollider с назначенным мешем;
- На объекте присутствует активный MeshFilter с назначенным мешем, причем игровой объект имеет запрещающий тэг;
- На объекте присутствует активный Terrain.

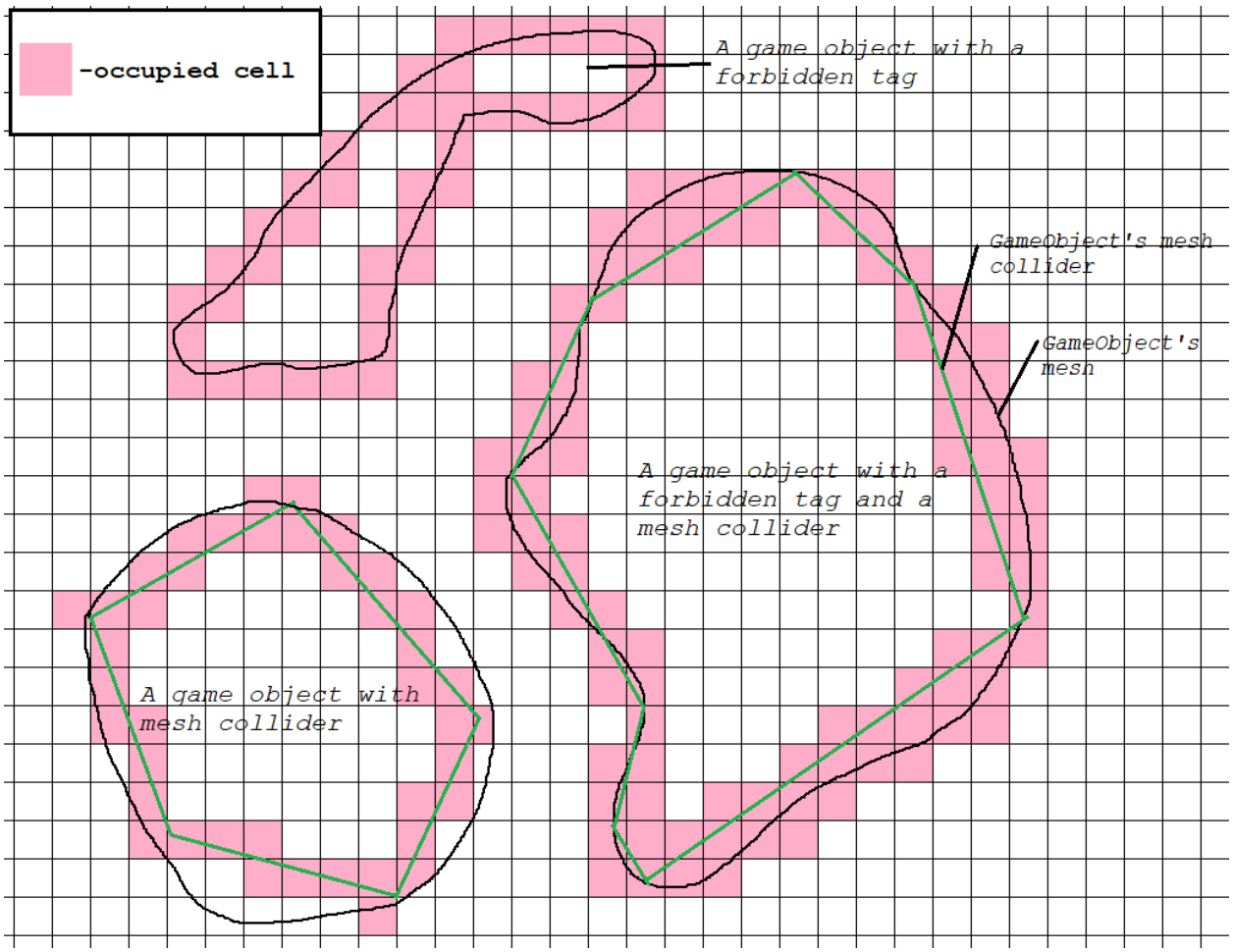
Все объекты, имеющие компонент Pursuer не будут рассматриваться как препятствия.

При обработке сцены, игровые объекты с запрещающими тэгами будут приравниваться к препятствиям. Расчет клеток, занимаемых такими объектами, будет производиться на основе Mesh, назначенного соответствующему полю компонента MeshFilter.

При обработке препятствий, выбор занятых клеток производится на основании отдельных компонентов игрового объекта. Поэтому если игровой объект содержит несколько компонентов, присущих препятствию (например, и Terrain и MeshCollider), то выбор занятых клеток будет осуществлен для каждого такого компонента.

К сожалению, в настоящее время не реализована возможность обработки препятствий, оснащенных следующими компонентами: BoxCollider, SphereCollider, CapsuleCollider. Но такие препятствия могут быть обработаны на основании Mesh выбранного в MeshFilter компоненте, если игровой объект препятствия имеет запрещающий тэг.

Схематично, результат обработки нескольких препятствий будет выглядеть следующим образом (двумерная проекция):

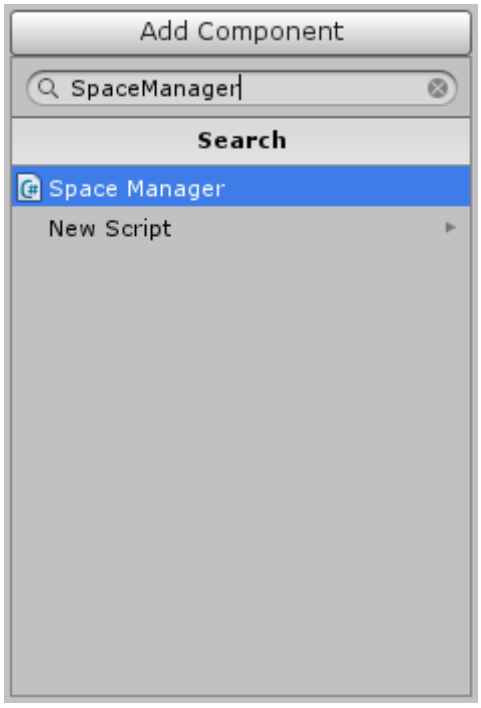

-occupied cell
A game object with a forbidden tag
GameObject's mesh collider
GameObject's mesh
A game object with a forbidden tag and a mesh collider
A game object with mesh collider

Использование ассета

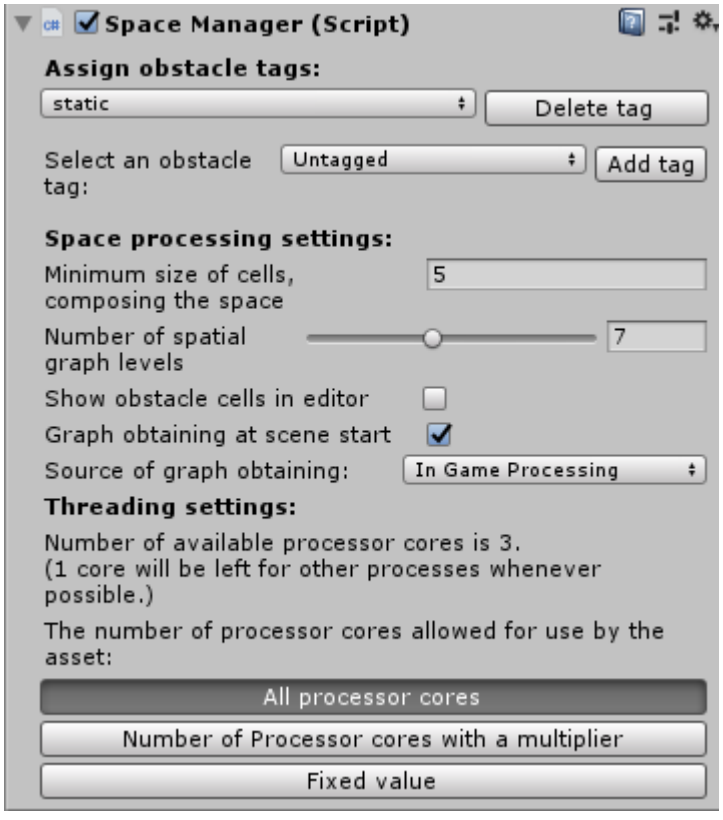
1. На игровой сцене, в которой предполагается использование данного ассета, добавьте MeshCollider ко всем имеющимся объектам, движение сквозь которые предполагается невозможным (кроме Terrain). Прочим непроходимым объектам назначьте запрещающие тэги.

2. Создайте пустой игровой объект и поместите на него скрипт “SpaceManager”.

Скрипт “SpaceManager” должен присутствовать на сцене в единственном экземпляре.



3. Настройте добавленный скрипт.

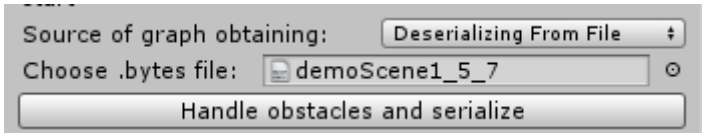


Рассмотрим блоки параметров.

Блок ***"Assign obstacle tags"*** позволяет добавлять и удалять тэги в список тэгов препятствий(запрещающих тэгов).

Блок ***"Space processing settings"*** предоставляет следующие настройки:

- Возможноть задавать минимальный размер клеток, на которые будет поделено пространство;
- Установка количества слоев графа поиска (каждый вышестоящий уровень графа поиска состоит из клеток, размер которых равен размеру клеток предыдущего уровня + 0.33 минимального размера клеток, т.е. клетки двух соседних уровней различаются между собой по размеру с шагом 0.33f*cellMinSize);
- Включение/выключение отображения занятых клеток в окне сцены редактора
- Включение/выключение получения графа при старте сцены
- Переключение способа получения графа. Это может быть расчет при старте сцены ("In Game Processing"), или загрузка графа из бинарного файла, куда он был предварительно сериализован ("Deserializing From File") (Сериализация и запись в файл может быть произведена при помощи кнопки с изображения ниже).



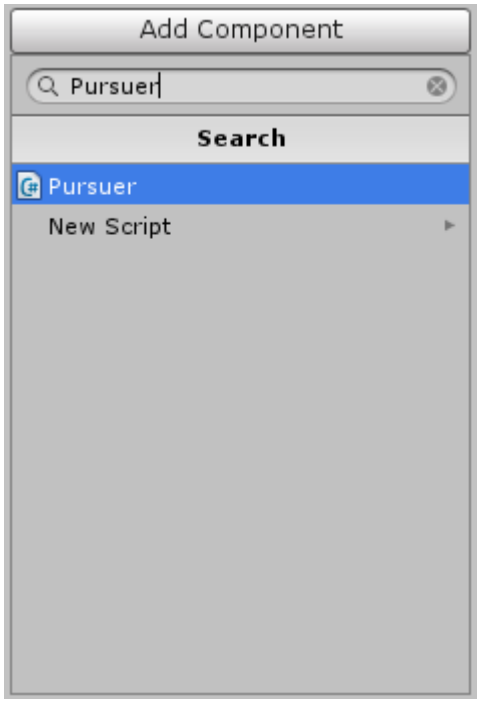
Количество уровней детализации стоит подбирать экспериментально. Как правило оптимальное количество равняется 5. Чем больше уровней детализации вы поставите, тем дольше будет длиться процесс обработки пространства или десериализации графа.

Размер ячеек не следует делать чрезмерно малым. Уменьшение размера ячеек влечет увеличение их количества, а как следствие, замедление всех расчетов и увеличение нагрузки на CPU. Размер должен быть строго необходимым.

Блок ***"Threading settings"*** предоставляет настройки использования многопоточности, которые используются при обработке препятствий, а так же при поиске пути. Здесь можно выбрать способ, с помощью которого будет задаваться ограничение на максимальное число рабочих потоков. Ниже описан результат выбора одно из трех доступных способа ограничения числа потоков.

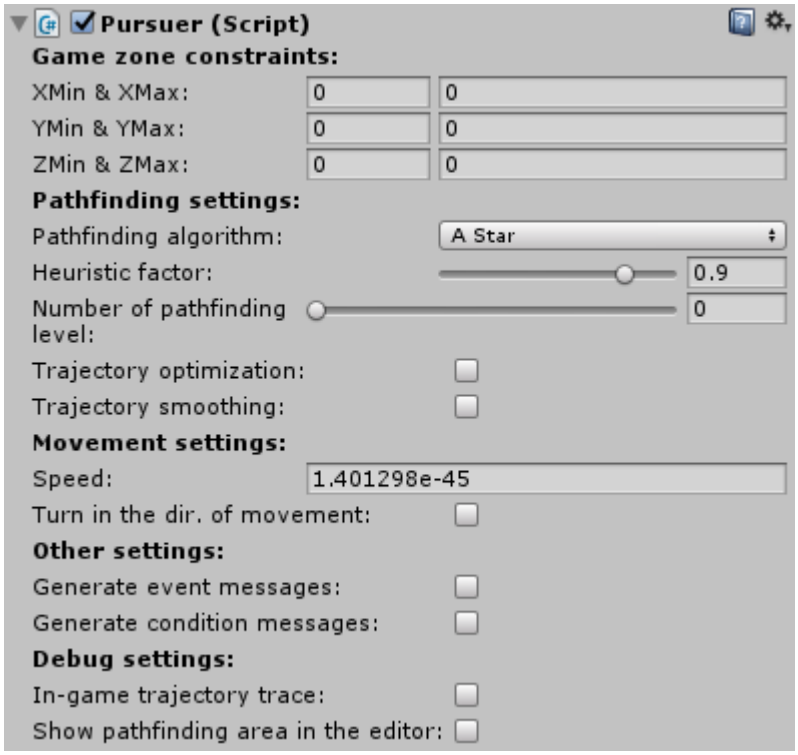
by the number of cores	Максимально число потоков будет равно числу программных ядер платформы.
by the num. of cores with a multiplier	Максимальное число потоков будет равно числу программных ядер платформы, умноженному на коэффициент, который вы можете задать сами. Это полезно, если вы хотите использовать например половину всех доступных ядер.
by a specific number	Максимальное число потоков будет равно числу, которое вы зададите.

4. На все игровые объекты-преследователи поместите скрипт "Pursuer".



5. Настройте добавленный скрипт.

Вы увидите следующий список параметров:



Поясним их значение.

Первый блок параметров ***"Game zone constraints"*** определяет положение и размеры зоны(параллелепипеда в мировых координатах), в которой возможен поиск пути.

XMin & XMax – ограничения по оси X, для других осей аналогично.

Второй блок параметров ***"Pathfinding Settings"*** определяет алгоритм поиска пути и итоговый вид траектории.

Так же, есть возможность выбора уровня пространственного графа, на котором будет осуществляться поиск(Number of pathfinding level - slider). Выбор уровня по сути определяет размер клеток, на которых будет искаться путь. Номер уровня может быть выставлен в диапазоне от 0 до числа уровней, выбранных в SpaceManager.

Алгоритм A* следует выбирать, если сцена содержит много открытых пространств, и препятствия на ней занимают объем, меньший, чем объем свободного пространства.

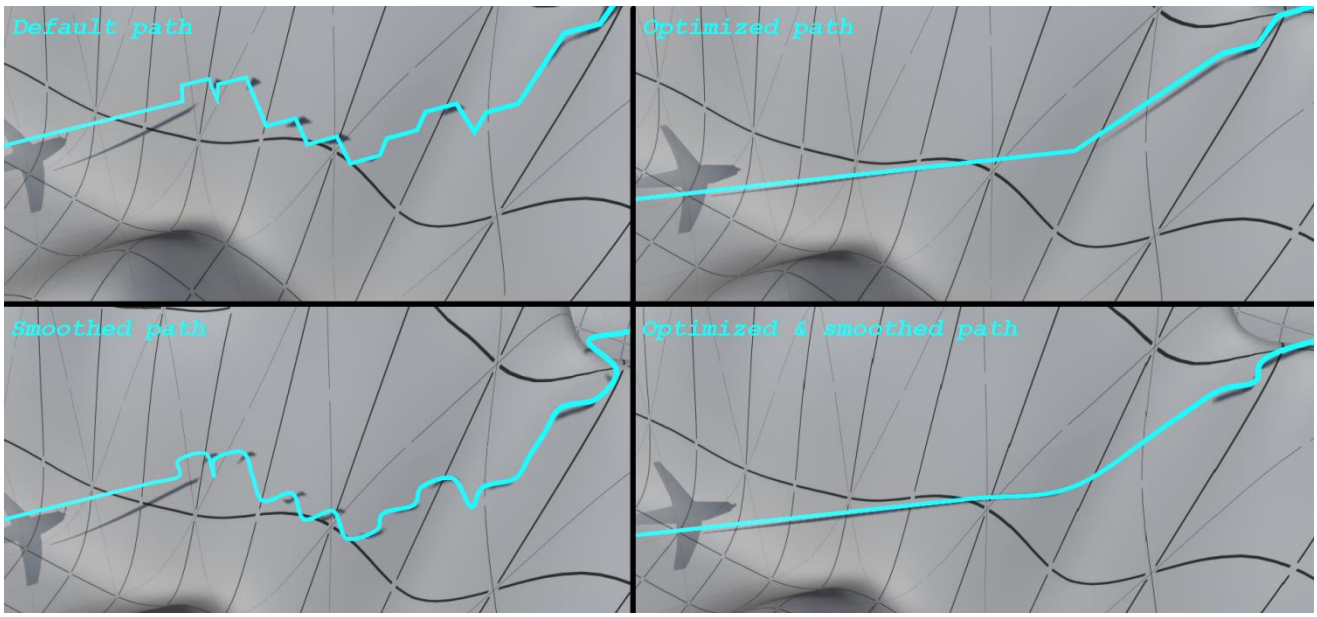
Волновой алгоритм стоит выбирать, если ваша сцена достаточно “тесна”. Например, она является замкнутым лабиринтом, или системой тоннелей. Лучше всего выбрать алгоритм, проведя тесты на сцене.

При выборе алгоритма A* станет доступен параметр “**Heuristic Factor**”. Он определяет вес эвристики поиска пути, проще говоря, позволяет выбирать между минимальностью пути и временем поиска. Допустимый диапазон (0.5f,1f). 0.5f - самый короткий путь из возможных, 1f - самое быстрое время поиска. Рекомендуемое значение -0.9f.

Параметр “**Trajectory Optimization**” отвечает за оптимизацию траектории. Если он имеет истинное значение, то найденная траектория будет избавлена от излишков и максимально упрощена.

Параметр “**Trajectory Smoothing**” позволяет включить сглаживание найденной траектории. Точки сглаженной траектории всегда будут находиться в некоторой, достаточно малой окрестности точек исходной траектории. Будьте осторожны при использовании сглаживания. Небольшое отклонение от исходной траектории неизбежно при интерполяции исходного пути. Из-за этого сглаженная траектория может задевать окружающие препятствия в некоторых местах.

Смысл этих параметров отражен на картинке ниже:



*Третий блок параметров “**Movement settings**”* определяет характеристики движения объекта по построенному пути.

Он позволяет настраивать скорость движения (в координатах Unity) по найденной траектории, а так же включать поворот преследователя в сторону вектора движения (будет ли локальный вектор направления (z) объекта сонаправлен с вектором движения).

Четвертый блок ***"Other settings"*** позволяет включить или отключить отправку сообщений игровому объекту. Подробнее о сообщениях читайте в разделе "Состояния и события класса Pursuer".

Блок ***"Debug Settings"***, позволяет рисовать найденную траекторию, а так же включать или выключать отображение зоны поиска в редакторе.

6. После обработки сцены вызовите функцию MoveTo() из класса Pursuer.

Если при настройке SpaceManager вы оставили активным параметр "Process At Startup", то обработка сцены начнется в момент старта сцены. По завершении обработки сцены всем игровым объектам, содержащим Pursuer среди своих компонентов, будет отправлено сообщение "**TheGraphIsReady**". Получение этого сообщения следует расценивать как сигнал о готовности графа поиска. *Вызывайте MoveTo() только после завершения обработки сцены. В противном случае будет возбуждено исключение* "**GraphNotReadyException**".

Функция MoveTo() принимает на вход один параметр типа Vector3 или Transform, означающий цель, к которой необходимо двигаться. После вызова функции начнется поиск пути до цели. По окончании поиска объект начнет движение до цели по найденной траектории.

Пример использования MoveTo() после обработки сцены:

Пусть на игровой сцене есть настроенный SpaceManager с параметром "Process At Startup" установленным в истинное значение, а так же преследователь, на котором в качестве компонентов имеются скрипты Pursuer и PursuerController. Тогда следование до цели может быть начато следующим образом:

PursuerController.cs

```
using UnityEngine;

[RequireComponent(typeof(Pursuer))]
public class PursuerController : MonoBehaviour {
    public Transform target;
    //TheGrapIsReady() will be executed when the gameObject receives the
"TheGraphIsReady" message
    public void TheGraphIsReady()
    {
        gameObject.GetComponent<Pursuer>().MoveTo(target);
    }
}
```

Что бы прервать следование до цели и перевести Pursuer в исходное состояние, используйте метод ResetCondition(). Вызов этого метода возможен в любой момент времени и при любом состоянии преследователя. Вызов переведет преследователя в первоначальное состояния, остановив при этом все активные потоки.

И так, вот перечень правил для успешного использования ассета:

-и преследователь и цель должны находиться внутри зоны поиска, настроенной в Pursuer скрипте преследователя.

-на момент начала поиска пути пространство вокруг преследователя должно быть свободно от препятствий в радиусе не менее размера клетки уровня поиска, выбранного в настройках Pursuer. То же касается и цели преследования.

-минимальный размер клеток пространства, настроенный в SpaceManager должен быть строго необходимым. (не надо делать его меньше, чем это необходимо)

-на момент начала поиска пути сцена должна пройти первичную обработку препятствий или загрузку графа из бинарного файла. (SpaceManager.isPrimaryProcessingCompleted == true)

-преследователь не должен находиться внутри непроходимых объектов, это же требование справедливо и для цели преследования.

Это все, что необходимо знать новичку для использования нашего ассета. Но у нас есть в запасе еще куча возможностей, и если есть необходимость, вы можете прочесть про них ниже.

Мы будем очень благодарны, если вы оставите детальный обзор нашего ассета на странице ассет стора, основываясь на опыте использования, разумеется.

Дополнительные возможности

Преследование движущейся цели

В практике построения игровых сценариев нередка ситуация, когда преследуемый объект постоянно находится в движении. Поскольку нахождение пути не является моментальной операцией, частый пересчет пути с остановкой преследователя выглядит внешне как постоянное дергание преследователя. Мы в нашем ассете внедрили возможность корректирования пути без остановки преследователя.

Положим, преследователь уже находится в движении и следует по пути до местоположения цели. Но во время движения преследователя цель сдвинулась на значительное расстояние от того места, куда был найден путь. В таком случае следует вызвать функцию RefinePath(), подавая в качестве аргумента новое местоположение цели. Эта функция имеет один входной параметр (аналогично функции MoveTo()) типа Vector3, либо Transform.

Пример применения данной возможности вы можете найти в первой демо-сцене(PathFinder 3D/Scene1/Scripts/MissleController.cs). Данная функция используется для корректирования пути самонаводящихся ракет.

В типичном случае вызов корректирования пути может выглядеть вот так:

```
private void Update()
{
        //if the target has moved from the previous coordinate(targetOldPos) to more than
"targetPathUpdateOffset", update the path to the target
        if (Vector3.Distance(targetOldPos, target.position) > targetPathUpdateOffset)
        {
            targetOldPos = target.position;
            if (thisPursuerInstance.GetCurCondition() == "Movement")
                thisPursuerInstance.RefinePath(target);
        }
}
```

Нахождение пути вне “Pursuer”

Вполне допустима ситуация, что вам понадобилось найти путь для собственных целей между двумя точками в пространстве (не для использования этого пути внутри Pursuer). Сделать это можно с помощью класса Pursuer, путем вызова процедуры FindWay().

Полное определение функции выглядит следующим образом:

```
void FindWay(Vector3 startPos, Vector3 targetPos, int pathfindingLvl, List<Vector3> foundPath, PathfindingAlgorithm usingAlg, Action failurePathfindingActions = null, Action succesPathfindingActions = null, bool inThreadOptimization = false, bool inThreadSmoothing = false)
```

Аргументы имеют следующее назначение:

- Vector3 startPos - координата точки начала пути
- Vector3 targetPos - координата точки до которой надо найти путь
- int pathfindingLvl - уровень пространственного графа, на котором будет осуществляться поиск
- List<Vector3> foundPath - экземпляр List<Vector3>, куда будут записаны точки найденного пути
- PathfindingAlgorithm usingAlg – алгоритм, с помощью которого осуществляется поиск, может принимать следующие значения: PathfindingAlgorithm.AStar, PathfindingAlgorithm.waveTrace
- Action failurePathfindingActions - делегат, содержащий инструкции на случай, если не удалось найти путь между двумя заданными точками
- Action succesPathfindingActions - делегат, содержащий инструкции на случай успешного нахождения пути
- bool inThreadOptimization - необходимо ли оптимизировать найденную траекторию?
- bool inThreadSmoothing - необходимо ли сгладить найденную траекторию?

Параметры failurePathfindingActions и succesPathfindingActions нужны для того что бы мы могли узнать, когда путь нашелся, либо не нашелся. Поскольку поиск пути происходит в другом потоке, завершение функции поиска будет для нас незаметным, если мы не будем использовать эти параметры.

Пример применения этой процедуры:

Пусть у вас имеется игровой объект, который содержит настроенный Pursuer скрипт, а так же другой скрипт, который будет использовать Pursuer исключительно для поиска пути.

```
using System;
using System.Collections.Generic;
using UnityEngine;

[RequireComponent(typeof(Pursuer))]
public class PathfindingScript : MonoBehaviour
{

    Pursuer thisPursuerInstance;
    List<Vector3> foundPath;
    private void Start()
    {
        thisPursuerInstance = gameObject.GetComponent<Pursuer>();
        foundPath = new List<Vector3>();
    }
    public void FindWay(Vector3 from, Vector3 to)
```

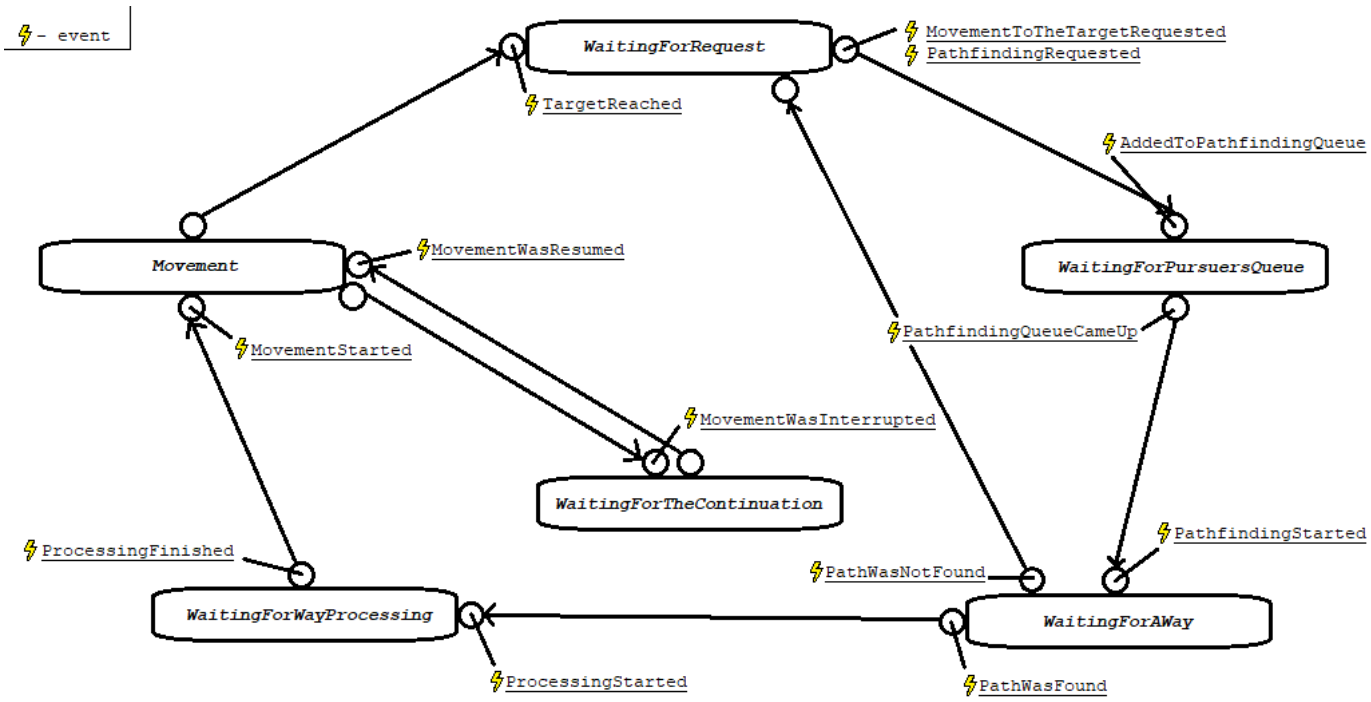
```
    {
        Action toDoAfterWayFound = new Action(() => {
gameObject.SendMessage("PursuerHasFoundAPath"); });
        Action toDoIfWayNotFound = new Action(() => {
gameObject.SendMessage("PursuerHasNotFoundAPath"); });
        thisPursuerInstance.FindWay(from, to, 0, foundPath,
PathfindingAlgorithm.AStar , toDoIfWayNotFound, toDoAfterWayFound, true, true);
    }
    public void PursuerHasFoundAPath()
    {
        Debug.Log("The way was found! The path contains the number of points = " +
foundPath.Count);
    }
    public void PursuerHasNotFoundAPath()
    {
        Debug.Log("There is no way between points");
    }
}
```

На самом деле, случай, когда между двумя точками не существует пути довольно сложно обнаружить в силу того, что алгоритм будет пытаться найти путь, пока не исчерпает все пространство поиска. По этой причине поиск может длиться очень продолжительное время. Для того что бы избежать такой ситуации, можно вручную остановить поток поиска пути (например по таймеру). Сделать это можно с помощью вызова процедуры StopAllThreadingTasks().

Состояния и события класса Pursuer

Для упрощения управления и взаимодействия с классом Pursuer,в него внедрена машина состояний и система событий, реализованных с помощью сообщений, отправляемых игровому объекту содержащему Pursuer в числе своих компонентов. Подробнее о механизме сообщений вы можете прочесть в соответствующем разделе документации - https://docs.unity3d.com/ru/current/ScriptReference/GameObject.SendMessage.html. Т.е. в ключевые моменты работы класса Pursuer происходит отправка соответствующего сообщения игровому объекту, на котором работает данный экземпляр Pursuer (gameObject.SendMessage(“messageText”)). Перечень всех сообщений приведен в разделе “Pursuer. Пользовательские вызовы и события”.

Ниже представлена машина состояний, отражающая штатный режим работы класса Pursuer после вызова MoveTo(). Переходы между состояниями сопровождаются срабатыванием некоторых событий. Назначение всех событий будет описано в таблице ниже.



Машина реализована с помощью делегата:

```
delegate void condition();

volatile condition curCond;
```

,принимающего в качестве значения одну из следующих процедур, соответствующих состояниям машины:

- void WaitingForRequest()
- void WaitingForPursuersQueue()
- void WaitingForAWay()
- void WaitingForWayProcessing()
- void WaitingForTheContinuation()
- void Movement()

Данный делегат выполняется каждый раз, когда наступает событие FixedUpdate(). Каждый раз при выполнении делегата игровой объект преследователя получает одно из соответствующих сообщений:

- CondWaitingForRequest
- CondWaitingForPursuersQueue
- CondWaitingForAWay
- CondWaitingForWayProcessing
- CondWaitingForTheContinuation
- CondMovement

Если преследователь находится в состоянии покоя и ничем не занят, то этому порядку вещей соответствует состояние “WaitingForRequest”. Это первоначальное состояние преследователя после старта сцены.

После вызова MoveTo() преследователь переходит в состояние “WaitingForPursuersQueue”. В этом состоянии он будет пребывать до тех пор, пока SpaceManager не выдаст ему разрешение на начало поиска пути.

После получения разрешения преследователь начнет поиск пути и перейдет в состояние “WaitingForAWay”. В этом состоянии он будет находиться, пока путь не будет найден.

После нахождения пути преследователь будет обрабатывать найденный путь, т.е. перейдет в состояние “WaitingForWayProcessing”. В этом состоянии он будет находиться, пока путь не будет обработан.

По завершении обработки найденного пути преследователь сможет начать движение и перейдет в состояние “Movement”. Находясь в этом состоянии преследователь будет следовать по найденному пути и выйдет из этого состояния только после достижения конечной точки пути. Выход из этого состояния означает возврат в первоначальное состояние (“WaitingForRequest”).

Движение преследователя может быть приостановлено в любое время с помощью вызова метода InterruptMovement(). Вызов этого метода переведет преследователя в состояние “WaitingForTheContinuation”. Возобновить движение можно с помощью вызова метода ResumeMovement(). Этот вызов переведет преследователя обратно в состояние “Movement”.

Из любого состояния возможен переход в первоначальное состояние (WaitingForRequest). Это возможно с помощью вызова метода ResetCondition(). Вызов этого метода так же остановит все активные потоки преследователя.

SpaceManager. Пользовательские вызовы и события.

Как уже было сказано ранее, SpaceManager должен присутствовать на сцене в единственном экземпляре. Обратиться к этому экземпляру можно из любого MonoBehaviour скрипта. Сделать это удобнее всего с помощью следующего вызова:
`Component.FindObjectOfType<SpaceManager>();`

SpaceManager предоставляет следующий список методов, доступных для вызова извне:

Метод	Описание	Аргументы	Возвращаемое значение
`public bool PrimaryProcessing()`	Производит первичную обработку сцены. Если ProcessAtStartup имеет истинное значение, то метод будет вызван при старте сцены. В противном случае его необходимо вызвать вручную. Завершение первичной обработки ознаменуется отправкой сообщения “PrimaryProcessingFinished” игровому объекту, на котором работает SpaceManager, а так же отправкой сообщения "TheGraphIsReady" всем игровым объектам, на которых есть скрипт Pursuer.	-	bool - вернет истинное значение, если первичная обработка была начата. Вернет ложное значение в случае, если первичная обработка уже была произведена ранее.
`public float GetProcessingProgress()`	Имеет смысл вызывать во время первичной обработки. Возвращает текущий прогресс первичной обработки сцены.	-	float - отражает текущий прогресс первичной обработки. Диапазон значений - [0.0f,1.0f].
`public int GetTotalTrisCountToProcess()`	Имеет смысл вызывать во время первичной обработки. Возвращает общее количество треугольников всех препятствий, которые будут обработаны.		int - общее количество треугольников всех препятствий.
`public int GetCurentProcessedTris`	Имеет смысл вызывать во время первичной обработки. Возвращает	-	int - текущее количество уже

Count()	текущее количество уже обработанных трегольников.		обработанных треугольников.
public void HandleAnObstacle (GameObject obstacleGOToHandle)	Производит перерасчет клеток, занимаемых препятствием. Клетки, занимаемые препятствием ранее, освобождаются от него. Имеет смысл вызывать для препятствий, появившихся после первичной обработки. Либо для препятствий, поменявших положение, размер или поворот во время игры.	GameObject obstacleGOToHandle - игровой объект препятствия, которое необходимо обработать.	-
public void RemoveAnObstacle(GameObject obstacleGOToRemove, bool destroyGOAfter = false)	Производит освобождение клеток занимаемых препятствием. Данный метод может быть полезен в случае, если препятствие разрушилось или стало недействительным.	GameObject obstacleGOToRemove - игровой объект препятствия, клетки занимаемые которым необходимо освободить. bool destroyGOAfter – уничтожать ли игровой объект после изъятия из пространственного графа	-

Pursuer. Пользовательские вызовы и события.

Для вашего удобства мы внесли в класс “Pursuer ” некоторое количество процедур и функций, позволяющих упростить управление преследователем и механизм межклассового взаимодействия.

Ниже представлена таблица, в полной мере раскрывающая назначение и функционал этих процедур и функций.

public float GetTotalPathLength()	Описание	Возвращает длину последнего найденного пути.
	Аргументы	-
	Возвращаемое значение	float - длина пути с учетом оптимизации и сглаживания в координатах unity.
public List<Vector3> GetFoundPath()	Описание	Возвращает список вершин последнего найденного пути без учета оптимизации и сглаживания.
	Возвращаемое значение	List<Vector3> - список вершин траектории.
public List<Vector3> GetFinalPath()	Описание	Возвращает список вершин последнего найденного пути с учетом оптимизации и сглаживания (если данные опции включены).
	Возвращаемое значение	List<Vector3> - список вершин траектории.
public bool SetConstraints(float xMin, float xMax, float yMin, float yMax, float zMin, float zMax)	Описание	Задает положение и размеры области, в которой возможен поиск пути.
	Аргументы	float xMin, float xMax - ширина области по оси x, и т.д.
	Возвращаемое значение	bool - вернет ложное значение, если хоть одна пара значений имеет ошибочные значения (Min>=Max). Вернет истинное

		значение, если новые значения успешно заданы.
public string GetCurCondition()	Описание / Возвращаемое значение	Возвращает имя метода присвоенного в данный момент делегату состояния. (см. раздел “Состояния и события класса Pursuer”)
float GetCurWayProgress()	Описание / Возвращаемое значение	Возвращает отношение длин уже пройденной части пути ко всему пути.
public bool RefinePath(Transform / Vector3 target)	Описание	Данную функцию следует использовать для корректирования пути до цели в том случае, если цель поменяла свое местоположение во время преследования. После вызова преследователь начнет пересчет еще не пройденной части пути, продолжая двигаться. После завершения пересчета преследователь перестроится на новую траекторию.
	Аргументы	Transform / Vector3 target - новое положение цели.
	Возвращаемое значение	bool - вернет ложное значение, в случае если преследователь не находится в состоянии “Movement”, т.е. не преследует никакую цель. Вернет истинное значение, если корректирование пути началось успешно.
public void MoveTo(Transform / Vector3 target, bool topPriority = false)	Описание	Данный метод следует использовать в случае, если вы хотите начать движение до цели. (см. раздел “Состояния и события класса Pursuer”). При вызове данного метода будут совершены следующие проверки: • Находится ли цель внутри области поиска • Находится ли преследователь внутри области поиска • Не находится ли цель в клетке занятой препятствием • Не находится ли преследователь в клетке занятой препятствием Отрицательный результат хоть одной проверки отменит вызов данного метода и приведет к отправке соответствующего сообщения игровому объекту преследователя.(об этом ниже)
	Аргументы	1. Transform / Vector3 target - цель(координата) до которой необходимо найти путь и начать движение. 2. bool topPriority = false - данный параметр определяет порядок запуска потока поиска пути. Истинное значение приведет к запуску поиска пути в обход очереди преследователей. Ложное значение приведет к постановке преследователя в очередь поиска пути(штатный сценарий). Поиск начнется по достижении своей очереди преследователем. (см. раздел “Состояния и события класса Pursuer”)
public bool InterruptMovement()	Описание	Приостановит движение преследователя, путем перевода в состояние “WaitingForTheContinuation”. (см. раздел “Состояния и события класса Pursuer”)
	Возвращаемое значение	bool - вернет ложное значение, если

		преследователь не находится в состоянии “Movement”. В случае успешной приостановки движения вернет истинное значение.
public void ResetCondition()	Описание	Следует вызывать для перевода преследователя в первоначальное состояние (“WaitingForRequest”). Останавливает все активные потоки, очищает все локальные переменные, убирает преследователя из очереди(если он в ней состоит).
public bool ResumeMovement()	Описание	Возобновит движение преследователя, путем перевода в состояние “Movement”. (см. раздел “Состояния и события класса Pursuer”)
	Возвращаемое значение	bool - вернет ложное значение, если преследователь не находится в состоянии “WaitingForTheContinuation”. В случае успешного возобновления движения вернет истинное значение.
public void StopAllThreadingTasks()	Описание	Останавливает все активные потоки, очищает все локальные переменные, убирает преследователя из очереди(если он в ней состоит). О применении данной процедуры рассказано в разделе Нахождение пути вне “Pursuer”.
public void FindWay(Vector3 startPos, Vector3 targetPos, int pathfindingLvl, List<Vector3> foundPath, PathfindingAlgorithm usingAlg, Action failurePathfindingActions = null, Action succesPathfindingActions = null, bool inThreadOptimization = false, bool inThreadSmoothing = false)	Подробно о применении данного метода рассказано в разделе **Нахождение пути вне “Pursuer”**.	

Так же в класс Pursuer внедрена система событий, основанная на механизме отправки сообщений игровому объекту (объекту, содержащему данный экземпляр класса Pursuer). Подробнее о механизме сообщений вы можете прочесть в соответствующем разделе документации - https://docs.unity3d.com/ru/current/ScriptReference/GameObject.SendMessage.html.

Все сообщения представлены в таблице ниже с разъяснениями.

Текст сообщения	Условия возникновения сообщения
EventAddedToPathfindingQueue	Возникает после того как преследователь встал в очередь на поиск пути.
EventPathfindingQueueCameUp	Возникает после того как преследователь в порядке очереди получил разрешение на поиск пути.
EventPathfindingRequested	Возникает при вызове функции MoveTo() после успешного прохождения всех проверок на предмет возможности поиска пути межу двумя точками.
EventPathfindingFromOutwardPointRequested	Возникает после вызова MoveTo() в

	случае, если преследователь находится вне области поиска, указанной для него.
EventPathfindingToOutwardPointRequested	Возникает после вызова MoveTo() в случае, если цель находится вне области поиска, указанной для преследователя.
EventPathfindingFromStaticOccupiedCellRequested	Возникает после вызова MoveTo() в случае, если преследователь находится в клетке, занятой некоторым препятствием.
EventPathfindingToStaticOccupiedCellRequested	Возникает после вызова MoveTo() в случае, если цель находится в клетке, занятой некоторым препятствием.
EventPathfindingStarted	Возникает при старте потока поиска пути и переводе состояния преследователя в “WaitingForAWay”.
EventProcessingStarted	Возникает после того как путь был найден, в момент запуска потока обработки пути.
EventProcessingFinished	Возникает при завершении работы потока обработки пути.
EventMovementToTheTargetRequested	Возникает при вызове MoveTo().
EventMovementStarted	Возникает после завершения работы потока обработки пути при переходе преследователя в состояние “Movement”.
EventMovementWasInterrupted	Возникает после успешного вызова InterruptMovement() при переходе преследователя в состояние “WaitingForTheContinuation”.
EventMovementWasResumed	Возникает после успешного вызова ResumeMovement() при переходе преследователя в состояние “Movement”.
EventTargetReached	Возникает при достижении преследователем последней точки пути при переходе преследователя в состояние “WaitingForRequest”
EventPathWasFound	Возникает после завершения потока поиска пути при переходе преследователя в состояние “WaitingForWayProcessing”. Означает успешное нахождение пути между двумя точками.
EventPathWasNotFound	Возникает после завершения потока поиска пути при переходе преследователя в состояние “WaitingForRequest”. Означает тот факт, что путь между двумя точками не удалось найти.
EventPathRefiningRequested	Возникает после вызова RefinePath().
EventPathRefiningStarted	Возникает после вызова RefinePath() при запуске потока поиска пути.
EventPathWasRefined	Возникает после успешного нахождения корректировочного участка траектории и записи обновленного пути.
EventPathWasNotRefined	Возникает в случае, если не удалось найти корректировочный участок траектории.

По поводу любых возникающих вопросов, предложений, замечаний вы можете обращаться по электронному адресу:

support@gracefulalgs.com

Страничка ассета на ассетсторе:

https://assetstore.unity.com/packages/tools/ai/pathfinder-3d-100285

Вот соответствующая тема форума:

https://forum.unity.com/threads/pathfinder-3d-pathfinding-in-three-dimensional-space.499144/

Страничка ассета на нашем сайте:

https://gracefulalgs.com/portfolio/pathfinder/

Всегда рады помочь вам :)