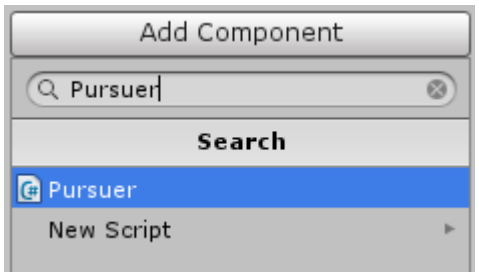
User's manual for "PathFinder3D"

After completing the import of our game asset there are only few steps left for using its all possibilities:

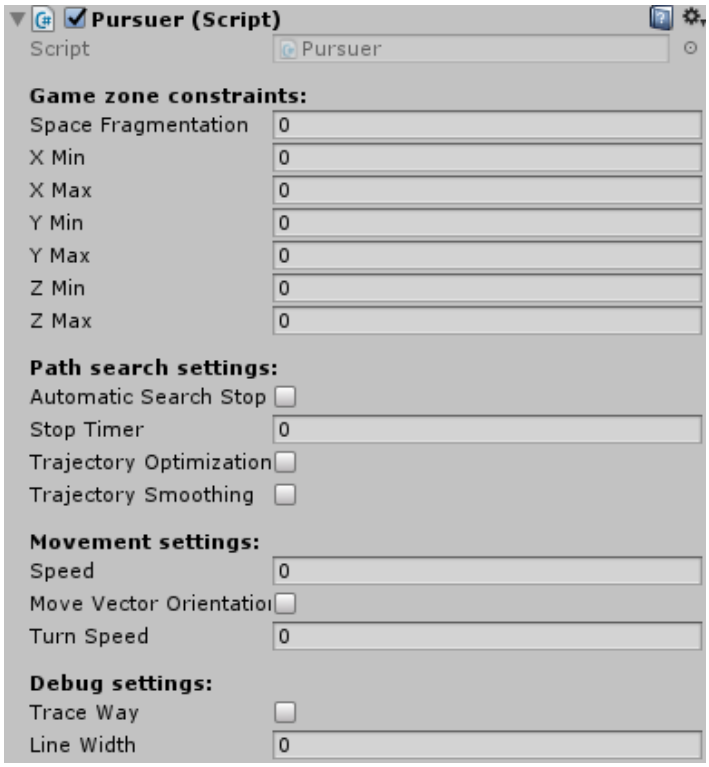
1.Add colliders to all objects that cannot be walked through in the required game scene

2.For every game object, which is to find the path (hereinafter – pursuer) add the script "Pursuer.cs" in the Inspector Menu.



3. Adjust the added script in the Inspector menu. It is quite easy to do.

You will see a following list of parameters:



Let's describe these parameters in detail.

1st section – "Game zone constraints" defines the position and size of the zone **(parallelepiped in the world coordinates), where searching for the path takes place.**

XMin,XMax – X axis constraint, etc.

"Space Fragmentation" defines the size of cells which divide the space inside of the zone. The degree of path's accuracy depends on its value. The cell should be a little bigger than the pursuer. It is necessary for the last one to avoid obstacles along the path. For example, if the game object's size is approx 4.5 units (Unity

coordinate system) then "Space fragmentation" should be 5. Be careful while setting this parameter as for the large zone too small cell size causes great decrease in speed of searching. Generally speaking, for better performance, always strive to ensure that the cell size is as large as possible and the size of the search zone is as small as possible.

2nd section – "Path Search Settings" defines characteristics of searching for the path and the final trajectory shape.

"Automatic Search Stop" is a Boolean parameter that enables an automatic cancel of **searching for the path** at the expiration of allotted time. It can be rather useful when searching for the path takes inappropriately long time or the path cannot be found at all. The last one situation is difficult to be recognized in advance and makes you wait till algorithm ends.

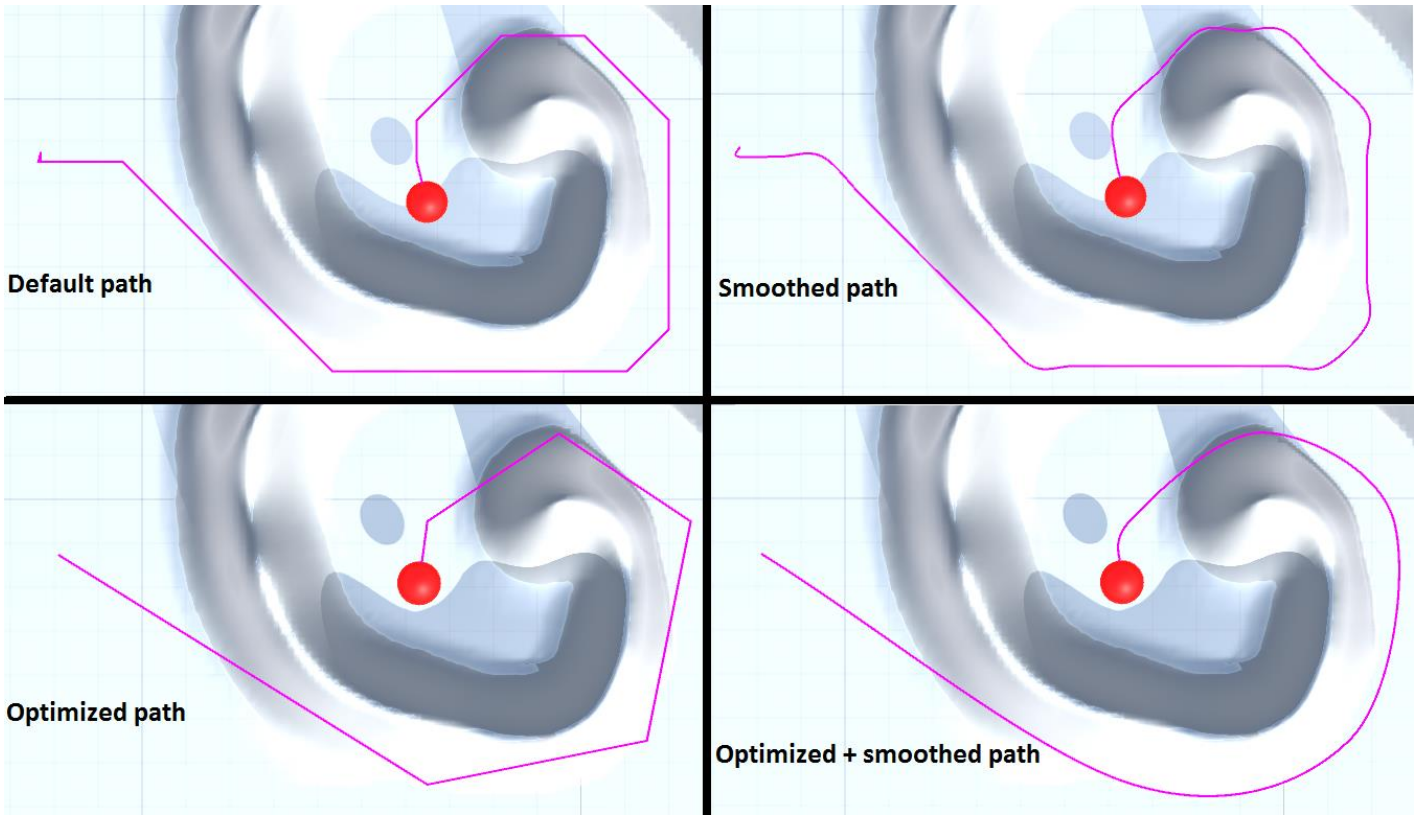
"Stop Timer" – defines the allotted time when the "Automatic Search Stop" value is true.

"Trajectory Optimization" is a Boolean parameter that enables to maximally decrease the found path due to the optimization when its value is true.

"Trajectory Smoothing" parameter enables local smoothing for the found trajectory. The points of smoothed trajectory are always close enough to the origin curve ones.

Be careful! Smoothed trajectory will inevitably deviate from the origin one due to the interpolation. That's why smoothed path can run into the obstacles.

You can see the result of applying to these parameters below:



3rd section – "Movement settings" defines how the pursuer moves along the path

"Speed" – velocity of object along the path, defines the distance in Unity coordinate system covered by the pursuer per one event FixedUpdate().

"MoveVectorOrientation" – defines if object's local vector z is co-directional with the motion vector

"TurnSpeed" – defines maximum degree the pursuer turns per one event FixedUpdate().

"Debug Settings" section, supposedly, isn't to be explained thoroughly ☺ Its purpose is the found path visualization.

4. After launching the scene it is necessary to call the function MoveTo() from Pursuer class.

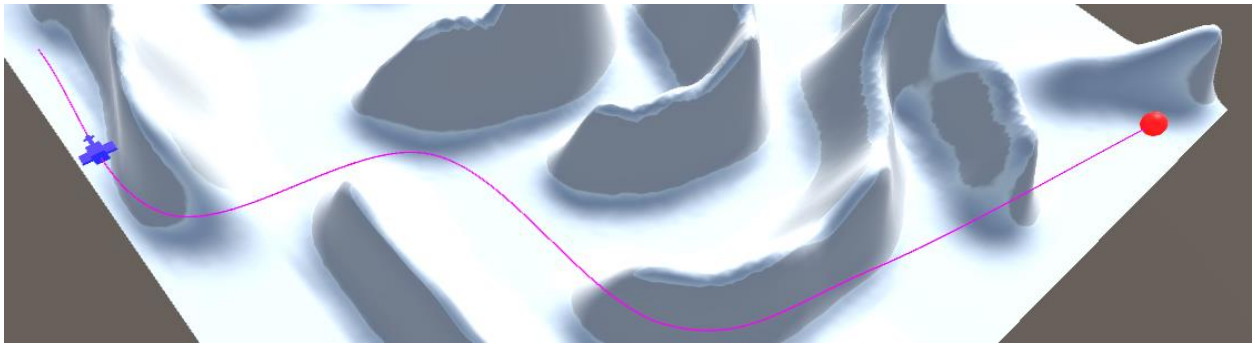
The argument of this function can be a coordinate (Vector3) or an object of Transform type. Warning! If the target is an object of Transform type be sure to deactivate colliders on both the pursuer and the target. Otherwise, the path cannot be found a priori.

Calling the method is followed by search for a path to the target. At the end of the search the pursuer starts moving along the found trajectory. Repeated calling MoveTo() is useless before completion the way. To stop the movement and enable a new search to another target use CancelMovement() method. After calling it you can use MoveTo() again.

So, there are some tips for using this asset successfully:

-correct settings for the search zone
-accurate choice of cell size
-target and pursuer should be disposed at a distance greater than Space Fragmentaion from the edges of the search zone.
-colliders on the pursuer and the target are off while searching the path
-use Automatic Search Stop if available
-leave a spare space free from colliders around the pursuer and the target. The nearest colliders should be further than Space Fragmentation value.

Here is an example of proper "Pursuer" class performance considering all tips above:

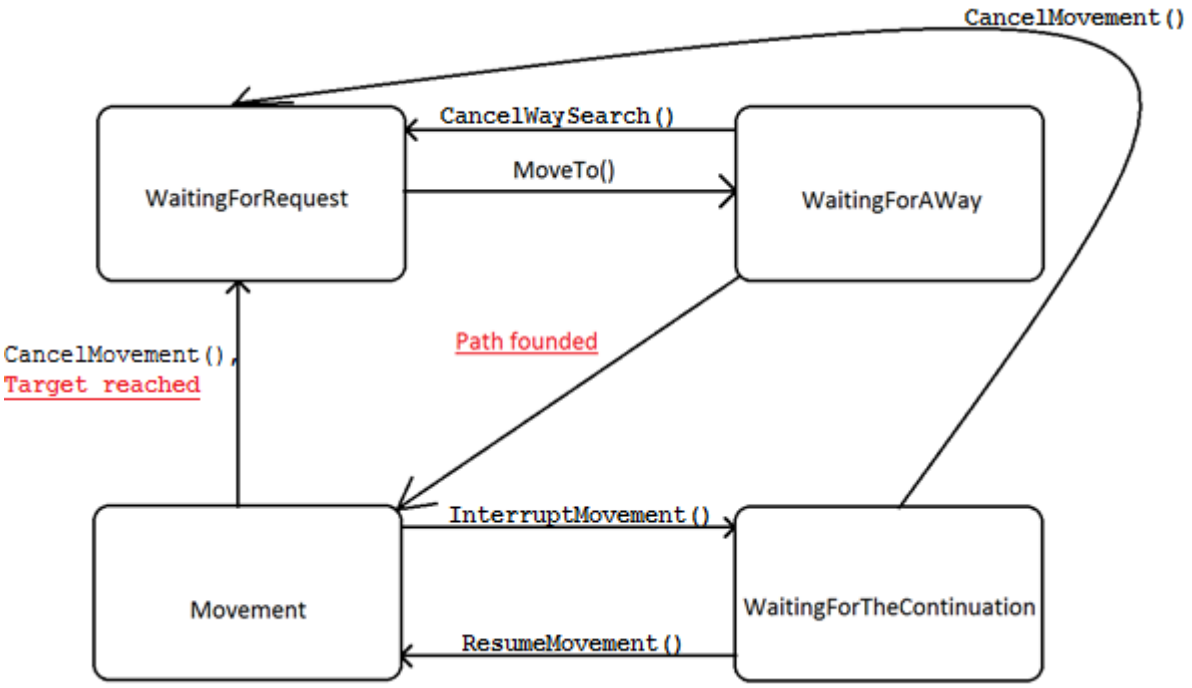


It was all, what a beginner has to know for using our asset. However, we got various opportunities in reserve and if you need more, read below.

Additional opportunities

State machine

The finite state machine (FSM) is realized for comfortable pursuer management within the "Pursuer.cs" class. Below you can see the diagram of its functioning.



FSM is realized as a delegate,

```
delegate void conditions();
conditions curCondition;
```

that takes one of the following procedures (matching the machine states) as the value:

```
void Movement()
void WaitingForRequest()
void WaitingForAWay()
void WaitingForTheContinuation()
```

This delegate proceeds every time an event FixedUpdate() occurs.

When the pursuer is motionless, the state is "WaitingForRequest". That's the initial pursuer's state after the scene started.

After receiving a request for moving toward the target, pursuer goes into state "WaitingForAway". This state lasts till the path is recieved as an argument for some procedure (in Pursuer class it is - `void ProcessWay(List<Vector3> InputWay))`, that is called by core after the path is found. To interrupt this state on purpose user can call the CancelWaySearch() method. Then the pursuer goes into the initial state.

After the path is found the core calls the procedure ProcessWay(List<Vector3> InputWay). During this procedure the state changes into "Movement". This state can be forced into "WaitingForTheContinuation" state, what is useful if you need to suspend pursuer's movement for some reason. At the moment of reaching the destination the state goes back to the initial one ("WaitingForRequest"). Forced transition to the initial state is reached through CancelMovement() method. Also, this function will erase the current path and lead all variables to default ones through calling ClearResources() method.

The transition from the state "WaitingForTheContinuation" to the state "Movement" is possible with ResumeMovement() method and to the initial state ("WaitingForRequest") with CancelMovement() method.

Core calls.

All classes and interfaces in Assets/PathFinder3D/Core/ consist the core of our asset and perform the algorithm of the search for the path in space considering obstacles (game object colliders). We strongly recommend you not to change this folder content.

Here is an example of how to call the core for searching for the path from one point to another:
Firstly you need to allow using the namespace " Assets.PathFinder3D ":

```
using Assets.PathFinder3D;
```

Calling the core must take place within the child class of "MonoBehaviour" class:

```
public class YourClass : MonoBehaviour
{
    // procedure initiates the search for the path
    void YourProc()
    {
        float xMin = -100;
        float xMax = 100;
        float yMin = -100;
        float yMax = 100;
        float zMin = -100;
        float zMax = 100;
        float spaceFragmentation = 5;
        double heuristicFactor = 0.7d;
        Vector3 target = Vector3.one;
        Coroutine AStarCoroutine;

        /initiation of SpaceConstraints instance - defines the search zone
        constraints /
```

```
        SpaceConstraints constraints new SpaceConstraints(xMin, xMax, yMin,
        yMax, zMin, zMax);

        // initiation of SpaceGraph  instance  describes a graph which is the
        foundation for the search. "spaceFragmentation" argument defines
        space granularity (size of the cells which are the graph nodes).
        heuristicFactor  argument sets the heuristics value that defines
        algorithm's tendency of choosing the nodes that leads faster to the
        target. Allowable range for this value: [0.5d,1]. 0.5d  - the
        shortest path but the longest search. 1 – the fastest search but
        hardly optimized path. this argument is the current  MonoBehaviour
        instance

        SpaceGraph spaceGraph new SpaceGraph(constraints, spaceFragmentation,
        heuristicFactor, this);

        // set the target of the path
        spaceGraph.SetGoal(target);
        //set the starting point coordinates
        spaceGraph.SetStart(transform.position);


        // request of the search start.  returns the Coroutine function
        (AStarCoroutine) realizing the search for the path. It is useful when
        you need to stop the search. On this purpose stop the realized
        function with StopCoroutine(). This function's argument is a
        procedure with one argument List<Vector3> which is called after the
        search completion
        AStarCoroutine = spaceGraph.GetWay(PathReciever);

    }

    //procedure-reciever of the found path is called by co-routine
    "AStarCoroutine" after the search completion

    void PathReciever(List<Vector3> Input_path)

    {

    // "Input_path" argument is a list of the found path nodes

    //Further in the procedure body you can do anything to the found path}

}
```

User defined procedures and functions

For comfort reasons we included some procedures and functions in the "Pursuer.cs" class. They allow simplifying the interclass interaction mechanism and the pursuer management

You can find full descriptions of their assignments and functionalities in the table below. All of the following members have an access attribute "public", which allows you to access them from outside the class.

Procedure/function name	Assignment	Arguments	Value returned
`void ClearResources()`	cleans local class fields. deletes the recorded path and attendant characteristics	-	-
`bool CancelMovement()`	Stops the moving along the path, calls ClearResources(), puts the FMS into WaitingForRequest state.	-	`bool` – returns `true` value in case of successful performance and `false` value when the FSM cannot be put into "WaitingForRequest" state.
`bool CancelWaySearch()`	Stops the Coroutine function that performs the search for the path, puts the FMS into WaitingForRequest state.	-	
`List<Vector3> GetWay()`	Allows to get the list of the found path nodes.	-	`List<Vector3>` - list of the found trajectory nodes. Returns `null` when the FSM is in a state where the path isn't assumed to be found (WaitingForRequest,WaitingForAWay.
`string GetStatus()`	Allows to know the current FSM state	-	`string` – a string with a full name of the method given to "curCondition" variable-delegate
`float GetLength()`	Allows to know the whole trajectory length	-	`float` – the found trajectory length (in Unity coordinate system). Returns -1 when the FSM is in a state where the path isn't assumed to be found (WaitingForRequest ,WaitingForAWay)
`float GetProgress()`	Allows to know the ratio between the traveled distance and the whole trajectory length	-	`float` – ratio between the traveled distance and the whole trajectory length. Returns -1 when the FSM is in a state where the path isn't assumed to be found (WaitingForRequest ,WaitingForAWay)

float GetSpeed()	Allows to know the current pursuer's speed (in Unity coordinate per "FixedUpdate" event	-	float - the value of "speed" variable
bool InterruptMovement()	Suspends the pursuer moving by putting the FSM into WaitingForTheContinuation state	-	bool – returns true value in case of successful performance and false value when the FSM cannot be put into "WaitingForTheContinuation" state.
bool MoveTo(Vector3 target) bool MoveTo(Transform target)	Calls the ClearResources method. Initiates the search, puts the FSM into "WaitingForAWay" method. Transmits "ProcessWay" procedure as a parameter for couroutine function of the search for the path. Assign launched couroutine function to the AStarCoroutine variable.	Vector3 target – the target point coordinate Transform target – an object of an appropriate type that includes the target point	bool – returns true value in case of successful performance and false value when the FSM cannot be put into "WaitingForAWay" state
bool ResumeMovement	Resumes the suspended movement	-	bool – returns true value in case of successful performance and false value when the FSM cannot be put into "Movement" state
bool RedefineConstraints(float XMin, float XMax, float YMin, float YMax, float ZMin, float ZMax, float SideLength,double heur_fact)	Changes the search zone, space granularity, sets a new heuristics. Renews the search zone by initiaing the performing of constructor for SpaceConstraints constraints instance with new parameters Initiates the performing of constructor for SpaceGraph spaceGraph instance with the new search zone and the search graph characteristics	float XMin, float XMax, float YMin, float YMax, float ZMin, float ZMax – coordinates which constraints the search zone float SideLength space granularity characteristics– (size of the cells the space is divided in) double heur_fact – a heuristics value, defines the algorithm tendency of choosing the edges that leads faster to the target. . Allowable range for this value: [0.5d,1]. 0.5d - the shortest path but the	bool – returns true value in case of successful performance and false value if argument values are out of allowable range

		longest search. 1 – the fastest search but hardly optimized path. this argument is the current MonoBehaviour instance	
bool SetSpeed(float sp)	Allows to set the current pursuer speed (in Unity coordinates per an "FixedUpdate" event)	float sp – speed that is required for the pursuer	bool – returns true value in case of successful performance and false value if speed value isn't allowable.