Path Finding Extended

Overview

Path Finding Extended works off a rectangular grid with impassable cells set to 1. If you wish to change this value many scripts will need changed so I recommend creating a constant to containing the value 1 to use. In the case of the weighted scripts A* and Dijkstra's require an additional grid representing the movement costs of each of the tiles, a third grid can be used with A* and Dijkstra's which denotes the costs for moving in a particular direction.

You do not need to create this cost map manually as `path_finder_grid_declare_weights` allows the grid to automatically be created from a tile map and a map of the terrain to costs.

The search scripts all return a list of paths that the search tried before finding the goal or failing to find the goal as a `ds_map` to find the actual path from the start to the goal you must use `path_finder_path_to_goal` which transforms this map into a `ds_list` of all cells that must be travelled through in order. You can convert this to a GameMaker Path using `path_finder_list_to_path` to allow instance to travel along the path without additional scripts.

Path Finding Extended can also work off a node based approach. All the functions that use this node based approach begin with pathfinder_node instead of path_finder to avoid confliction over which utility scripts are to be used with which.

In this system you must create each node using `pathfinder_node_create` and then set each node's connections and if using an algorithm that takes movement costs into account, the cost to move to that node.

Limitations

Version 1 of Path Finding Extended only supports Rectangular Grids and 8 directional movement, future versions will expand the functionality to include more movement and grid options.

Version 2 of Path Finding Extended now comes with a Node based approach that allows any number of directions to be traversed upon and the nodes can be arranged in any way that you wish.

Support

Support for Path Finding Extended is available through customersupport@crystalfortress.com, in certain cases Skype support will be offered. Additional function request or queries should go through this channel.

Function List

```
path_finder_grid_declare(width,height) ;

path_finder_grid_cell_set_wall(grid,x,y);

path_finder_grid_instance_set_as_wall(grid,CellSize,XOffset,YOffset,inst);

path_finder_grid_cell_get_weight(Weighted Graph,id);

path_finder_grid_declare_weights(Graph,Terrain_Mappings,MinCost) ;

path_finder_grid_cell_weight(weighted graph,cell);

path_finder_grid_in_bounds(Grid,x,y);

path_finder_grid_neighbours(grid,x,y);

path_finder_grid_cell_passable(grid,x,y);

path_finder_grid_cell_get_y(grid,id);

path_finder_grid_cell_get_x(grid,id);

path_finder_grid_cell_get_id(grid,x,y) ;

path_finder_instance_get_cell(Inst,Graph,CellSize,XOffset,YOffset);

path_finder_dijkstra_search(graph,weighted graph, start, goal,return mode);

path_finder_a_star_search(Graph,weighted graph, start, goal,returnmode);

path_finder_breadth_first_search(graph, start,goal);

path_finder_greedy_first_search(Graph,start,goal);

path_finder_list_to_path(Graph,Path List,CellSize,XOffset,YOffset);

path_finder_path_to_goal(Paths,Start,Goal);

path_finder_can_reach_goal(Paths,Start,Goal) ‘

path_finder_directional_weight_grid_declare(Top Left , Top Center , Top Right ,
Center Left , Center Right,Bottom Left ,Bottom Center, Bottom Right) ;

path_finder_grid_8_neighbours(grid,x,y);

path_finder_greedy_first_search_8d(Graph,start,goal);

path_finder_breadth_first_search_8d(graph, start,goal);
```

```
path_finder_a_star_search_8d(Graph,weighted graph, start, goal,return mode);

path_finder_dijkstra_search_8d(graph,weighted graph, start, goal,return mode)

path_finder_dijkstra_search_8d_weighted(graph,weighted graph, start, goal,return
mode,MovementWeights) ;

path_finder_a_star_search_8d_weighted(Graph,weighted graph, start,
goal,returnmode,MovementWeights) ;

path_finder_cell_directional_constraints_create() ;

path_finder_cell_directional_constraints_set(Constraint
Map,Cell,Up,Down,Right,Left);

path_finder_grid_neighbours_directional_constraints(grid,x,y,constraints);

path_finder_breadth_first_search_tile_constraint(graph, start,goal,Constraints);

path_finder_greedy_first_search_constraint(Graph,start,goal,Constraints);

path_finder_a_star_search_tile_constraint(Graph,weighted graph, start,
goal,Constraints,returnmode);

path_finder_dijkstra_search_constraint(graph,weighted graph, start,
goal,constraints,return mode);

path_finder_breadth_first_search_general(graph, start,goal,Dirs,Constraints);

path_finder_greedy_first_search_8d(Graph,start,goal,directions,constraints);

path_finder_dijkstra_search_general(graph,weighted graph, start,
goal,MovementWeights,Constraints,directions,returnmode);

path_finder_a_star_search_8d_weighted(Graph,weighted graph, start,
goal,MovementWeights,Constraints,Directions,ReturnMode);

path_finder_cell_directional_constraints_set_ext(Constraint
Map,Cell,Right,TopRight,Top,TopLeft,Left,BottomLeft,Bottom,BottomRight)



ds_list_reverse(list)

ds_list_print_to_console("Header" , "Footer" , List)

maze_generate(width,height,complexity,density)



pathfinder_node_get_costs(Node) ;

pathfinder_node_get_type_cost(Node,Type Map) ;

pathfinder_node_get_connections(Node) ;

pathfinder_node_add_connections(Node,Connected Node List) ;
```

```
pathfinder_node_add_connected_costs(Node,Connected Node Costs) ;

pathfinder_node_create_type_map(Default Cost) ;

pathfinder_node_add_type_cost(Type Map , Type , Cost) ;

pathfinder_node_create(x,y,type) ;

pathfinder_node_debug_draw_connections(Node,One Way Colour , Two Way Colour) ;

pathfinder_node_debug_draw_connection_costs(Node,Colour) ;

pathfinder_node_debug_draw_node_numbers(Graph,Colour) ;

pathfinder_node_debug_draw_nodes(Graph,Colour,Radius) ;

pathfinder_node_highlight_path(Path,Colour) ;

pathfinder_node_breadth_first_search(Graph,Start,Goal) ;

pathfinder_node_greedy_first_search(Graph,Start,Goal) ;

pathfinder_node_a_star_search(Graph,Type Map,Start,Goal,Return Mode) ;

pathfinder_node_dijkstra_search(Graph,Type Map,Start,Goal,Return Mode) ;

pathfinder_node_closest_to_point(Graph,x,y,nth)

pathfinder_node_set_node_size(Size) ;
```

Usage

Using The Grid Based Approach

Set-Up

To begin create a `path_finder` grid using `path_finder_grid_declare(width,height)` with the correct dimensions for your problem and store the returned `ds_grid` id in a variable you will make extensive use of this variable.

```
MyGraph = path_finder_grid_declare(width,height) ;
```

Next you need to set the impassable cells, this can be done manually or if your system is based off instances you can use `path_finder_grid_instance_set_as_wall` which requires you to have decided the offsets and cell size of the physical grid.

```
path_finder_grid_instance_set_as_wall(MyGraph,32,32,32,obj_wall);
```

This grid should contain references to any different types of tile you wish to have, I'll call them tile ids from now on but they are merely numbers representing a different type of terrain. Version 1.0.1

and below of “Path Finding Extended” reserves 1 for impassable cells, any other tile id will be treated as passable.

You can specify these ids based on instances in the same way as setting walls like in the example below, this can be a very important step if you want to use increased movement costs for different terrain or if you have tiles you can only move through in limited directions.

```
path_finder_grid_instance_set_as_index(MyGraph,32,32,32,obj_water,-1);
```

These IDs a vital if you want to use the tile constraint features for gameplay features search as one-directional doors or jump-through platforms. The example below shows how to set-up constraints for a one-directional tile going where you can only move toward the top of the screen through it. It is this Constraints map that you will want to feed into the search functions with a constraint argument.

```
Constraints = path_finder_cell_directional_constraints_create() ;

path_finder_cell_directional_constraints_set(Constraints,2,true,false,false,false)
;
```

In order to find the path through our grid we first must know where we are and where we are going. For a rectangular grid the cell ids are laid out in the fashion below. If you know the x and y position in the grid you can use the function `path_finder_grid_cell_get_id(Grid,x,y)` which will return the id you need but more typically you’ll want to know the cell an instance is in. This can be achieved if you know a few extra parameters about your grid, namely the offsets and cell size. For use in path_finder scripts it’s generally more useful to get the cell id using `path_finder_instance_get_`cell but you can get the X and Y coordinate with a few simple operations below.

```
path_finder_instance_get_cell(Inst,Graph,CellSize,XOffset,YOffset);
```

```
GridX = floor((x-XOffset)/CellSize) ;

GridY = floor((y-YOffset)/CellSize) ;
```

Cell ID Layout

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19

20	21	22	23	24

Once this is done you need to decide which algorithm to use to find your path. In simple mazes Greedy-First will check less paths but in more complicated mazes will end up taking a longer path. A* and Dijkstra take movement costs into account so if you have different costs you will need to use one of them.

If you need to set movement costs the first thing we need to do is create a terrain mapping

```
TerrainMap = ds_map_create() ;

ds_map_add(TerrainMap,TileID,MovementCost) ;
```

This allows `path_finder_grid_declare_weights` to automatically generate a grid of the movement costs without manually creating the grid. To add more terrain types we merely add more tile ids into the map with their associated cost. This function also allows for a minimum cost which will be used for any tile ids not in the Terrain Map.

```
MyWeights = path_finder_grid_declare_weights(MyGraph,TerrainMap,1) ;
```

If you've decided to use the directional weighted algorithm you'll also need to set-up the directional weight grid using `path_finder_directional_weight_grid_declare(Top Left , Top Center , Top Right , Center Left , Center Right,Bottom Left ,Bottom Center, Bottom Right).`

Next we actually run the chosen algorithm, in the provided demo I use all 4 in order to demonstrate the differences. You'll only want to use a single one. If you are using an algorithm that needs movement costs they take a `ReturnMode` argument which designates what map to return. Typically you'll only want the paths which is a `ReturnMode` of 0 but if you need the costs a `ReturnMode` of 1 will return a map of all the costs.

Version 1.0.2 of "Path Finding Extended" includes the constants `PF_Return_Path` and `PF_Return_Cost` to represent these different Return Modes.

Once we've chosen the algorithm we want to use it, we'll call it and store the result in a variable.

```
MyPath = path_finder_a_star_search(Graph,weighted graph, start, goal,returnmode):
```

You may not have to recalculate your path as often as every step even if your target position changes as this map will contain all paths the algorithm attempted to travel in order to find the original goal, this can be a rather extensive list but can equally be rather barren.

I recommend checking if this map knows of an existing path through use of `path_finder_can_reach_goal` using your current cell id as the Start argument and your new target as the goal argument before recalculating your path.

Once you've got the paths map from the algorithm to get the actual path from the start to the goal you must use `path_finder_path_to_goal` which returns a `ds_list` of the cells to pass through to get from the start cell to the goal cell.

To convert this to a GameMaker Path you can use `path_finder_list_to_path` which will allow simpler movement around your grid.

An Optimisation Opportunity

```
var MyCell = path_finder_instance_get_cell(id,MyGraph,32,32,32) ;

var TargetCell = path_finder_instance_get_cell(target,MyGraph,32,32,32);

//If we know of a Path to the goal already, we can use it

if (path_finder_can_reach_goal(MyPaths,MyCell,TargetCell))

        {

        PathList = path_finder_path_to_goal(MyPaths,MyCell,TargetCell) ;

        GameMakerPath = path_finder_list_to_path(MyGraph,PathList,32,32,32)

        }

else //Otherwise we need to find another

        {

        MyPaths =
        path_finder_a_star_search(MyGraph,MyWeights,MyCell,TargetCell,PF_Return_Path);

        }
```

General Search Scripts

The General search scripts are versions of the search scripts that allow the use of all features that particular search script can use. However due to additional internal checks these scripts will be slightly slower.

All general search scripts take a DIRS argument that specifies if you want to use 4 or 8 directional movement. Input 0 for 4 directional movement and 1 for 8 directional.

Maze Generate

This is a simple algorithm that allows you to create a maze to test the path finding options on. It only supports mazes of odd lengths, you can modify the script to support even lengths but this will result in odd looking boundaries. This function returns a `ds_grid` that can be used just like any other `path_finder` grid.

Clean Up

While path_finder is good at destroying its internal data structures there is a few you need to tidy up yourself.

- The map of paths must be destroyed using `ds_map_destroy(Paths)`

- The path_finder grid must be destroyed using `ds_grid_destroy(Grid)`
- The path_finder weighted grid must be destroyed using `ds_grid_destroy(Weighted Grid)`
- The Terrain Mappings if used must be destroyed using `ds_map_destroy(Terrain Map)`
- The directional weighted grid must be destroyed using `ds_grid_destroy(Directional Weights)`

Using the Node Based Approach

Set-Up

To use the Node Based Approach you'll need to create all the nodes you need beforehand and give them coordinates and types. The node ids (actually ds_map indices) should be stored in a node array, it is this array you feed into any pathfinder_node script that takes a Graph argument.

```
random_amount = 20;

node_size = 0 ;


for(var i = 0 ; i < random_amount ; i++)

    {

    var Y = irandom_range(96,room_height-96) ;

    var X = 32 + 48*i ;

    Node[i] = pathfinder_node_create(X,Y,0) ;

    node_size = i ;

    }


for(var i = 0 ; i < random_amount ; i++)

    {

    var L = ds_list_create() ;

    var C = ds_list_create() ;

    var Nex = i + irandom_range(1,3) ;
    var Pre = i - irandom_range(1,3) ;

    var Alt = Nex + random_amount/2 ;

    if (Nex > node_size) { Nex -= node_size ; } ;

    if (Pre < 0 ) { Pre += node_size ; } ;

    if (Alt > node_size ) { Alt -= node_size ; } ;


    ds_list_add(C,irandom_range(1,3),irandom_range(2,5),1) ;

    ds_list_add(L,Nex,Pre,Alt) ;

    pathfinder_node_add_connections(Node[i],L) ;

    pathfinder_node_add_connected_costs(Node[i],C) ;

}
```

Once this is done we do a very similar procedure to the grid based approach in order to find the path through the nodes.

```
BFS = pathfinder_node_breadth_first_search(Node,Node[0],Node[3]) ;

Path  = path_finder_path_to_goal(BFS,Node[0],Node[3]) ;
```

This leaves BFS holding a ds_map of paths to all the nodes it checked and Path holding a ds_list of the nodes to travel to from `Node[0]` to `Node[3]` in the correct order.

You can covert this into a GameMaker Path with `pathfinder_node_list_to_path` or use your own script to move between the nodes.

If you wish to use A* or Dijkstra's searches you may wish to create a type map that denotes different costs associated with moving to a node of a given type. The Type Map is optional, if you do not need the additional costs from a Type Map input -1. However if you wish to use it you must first create a map then add the costs for each type to this map.

```
TypeMap = pathfinder_node_create_type_map(1) ;
```

Clean Up

You must destroy any structures returned by the pathfinder_node scripts by yourself. These include the following structures.

All pathfinder_node search scripts return a ds_map of nodes which should be destroyed with ds_map_destroy.

The list generated by `path_finder_path_to_goal` should be destroyed with ds_list_destroy.