PATHFINDING AND PATH DRAWING

By: Nathan Lovato - January 30, 2021

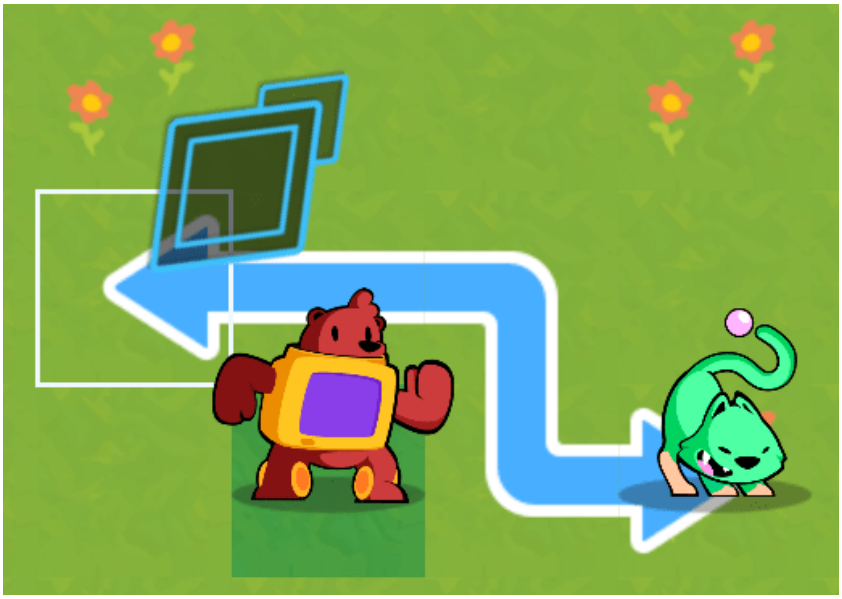
MENU

TRPG Unit movement

Handling grid interactions text
The Grid text
Creating the Unit text
The player's cursor text
Pathfinding and path drawing text
Drawing the path text
The flood fill algorithm text
Unit selection and cursor interaction text

In this lesson, we will create a `PathFinder` class that extends Godot's built-in `AStar2D`, an implementation of the AStar pathfinding algorithm.

We will use it to move units to a cell picked by the player, but also to display a preview of the path the unit will walk, inspired by Fire Emblem.



Godot comes with the algorithm implemented and optimized for you. If you want to learn to implement the AStar pathfinding algorithm itself, check out the Introduction to the A* Algorithm on Red Blob Games.

While it's often presented in games with grids, the AStar algorithm is flexible. It can find paths on any node graph. You could use it for a road or train network in an open-world game, for example.

To use an `AStar2D` object in Godot, you:

1. Add a list of points to the object, giving each a unique index.
2. Connect all the points that should form a walkable path, using the points' indices.

In the case of a grid, you should connect each cell to its unoccupied neighbors.

Once the graph is set, you call the `find_path()` function with the indices corresponding to two points: the start

and the end. The algorithm outputs a list of coordinates through which you need to navigate to reach the end.

You can already see the *catch*: the AStar object works with unique IDs for each point in the graph.

This is where our grid class comes in handy with its `as_index()` method. With this method, we can calculate a unique index for each cell coordinates in our grid.

Here is the code for the `PathFinder` class. Create a new script and save it there.

```
# Finds the path between two points among
walkable cells using the AStar pathfinding
algorithm.
class_name PathFinder
extends Reference

# We will use that constant in "for" loops later.
It defines the directions in which we allow a
unit
# to move in the game: up, left, right, down.
const DIRECTIONS = [Vector2.LEFT, Vector2.RIGHT,
Vector2.UP, Vector2.DOWN]

var _grid: Resource
# This variable holds an AStar2D instance that
will do the actual pathfinding. Our script is
mostly
# here to initialize that object.
var _astar := AStar2D.new()


# Initializes the Astar2D object upon creation.
func _init(grid: Grid, walkable_cells: Array) ->
void:
    # Because we will instantiate the
`PathFinder` from our UnitPath's script, we pass
it the data it
    # needs to initialize itself via its
constructor function, _init().
    _grid = grid
    # To create our AStar graph, we will need the
index value corresponding to each grid cell
```

```
index value corresponding to each grid cell.
Here,
    # we cache a mapping between cell coordinates
and their unique index. Doing so here slightly
    # simplifies the code and improves
performance a bit.
    var cell_mappings := {}
    for cell in walkable_cells:
        # For each cell, we define a key-value
pair of cell coordinates: index.
        cell_mappings[cell] =
_grid.as_index(cell)
    # We then add all the cells to our AStar2D
instance and connect them to create our
pathfinding
    # graph.
    _add_and_connect_points(cell_mappings)


# Returns the path found between `start` and
`end` as an array of Vector2 coordinates.
func calculate_point_path(start: Vector2, end:
Vector2) -> PoolVector2Array:
    # With the AStar algorithm, we have to use
the points' indices to get a path. This is why we
    # need a reliable way to calculate an index
given some input coordinates.
    # Our Grid.as_index() method does just that.
    var start_index: int = _grid.as_index(start)
    var end_index: int = _grid.as_index(end)
    # We just ensure that the AStar graph has
both points defined. If not, we return an empty
    # PoolVector2Array() to avoid errors.
    if _astar.has_point(start_index) and
_astar.has_point(end_index):
        # The AStar2D object then finds the best
path between the two indices.
        return _astar.get_point_path(start_index,
end_index)
    else:
        return PoolVector2Array()


# Adds and connects the walkable cells to the
Astar2D object.
func _add_and_connect_points(cell_mappings:
Dictionary) -> void:
    # This function works with two loops. First,
we register all our points in the AStar graph.
    # We pass each cell's unique index and the
corresponding Vector2 coordinates to the
    # AStar2D.add_point() function.
    for point in cell_mappings:
        _astar.add_point(cell_mappings[point],
point)

    # Then, we loop over the points again, and we
connect them with all their neighbors. We use
    # another function to find the neighbors
given a cell's coordinates.
    for point in cell_mappings:
        for neighbor_index in
_find_neighbor_indices(point, cell_mappings):
            # The AStar2D.connect_points()
function connects two points on the graph by
index, *not*
            # by coordinates.

_astar.connect_points(cell_mappings[point],
neighbor_index)
```

```
# Returns an array of the `cell`'s connectable
neighbors.
func _find_neighbor_indices(cell: Vector2,
cell_mappings: Dictionary) -> Array:
    var out := []
    # To find the neighbors, we try to move one
cell in every possible direction and is ensure
that
    # this cell is walkable and not already
connected.
    for direction in DIRECTIONS:
        var neighbor: Vector2 = cell + direction
        # This line ensures that the neighboring
cell is part of our walkable cells.
        if not cell_mappings.has(neighbor):
            continue

        # Because we call the function for every
cell, we will get neighbors that are already
        # connected. If you don't don't check for
existing connections, you'll get many errors.
        if not
_astar.are_points_connected(cell_mappings[cell],
cell_mappings[neighbor]):

out.push_back(cell_mappings[neighbor])
    return out
```

In the next lesson, we'll use the `PathFinder` to find and draw a path between two cells.

← PREVIOUS

NEXT →

MADE BY

Nathan Lovato

GDQuest founder. Courteous designer with a taste for Free Software. I promote sharing and collaboration.

RELATED COURSES

GODOT NODE ESSENTIALS 80$

Learn to create professional 2D games with the Godot game engine.

ULTIMATE GODOT COURSE BUNDLE

365$

This ultimate bundle gives you access to ALL our current and future Godot courses, at a discount. It's like a lifetime membership.

VIEW ALL

1 COMMENT

JI

JIM
November 1, 2021

Sorry, but I am confused as to where the script is supposed to be saved. The tutorial states: "Here is the code for the PathFinder class. Create a new script and save it there" But where "there" is isn't clear. Is it supposed to be saved as part of the Grid scene? Is it supposed to be saved as a new Resource or new Script?

REPLY TO JIM

Type here...

Supports GitHub-flavored Markdown.

Name

Your name

Website (optional)

https://example.com

Email (optional)

johndoe@example.com

Enter your email to get notified when someone replies to your comment.
We encrypt your addres with a strong 256-bit AES encryption.
We'll only use your address for notifications. You can unsubscribe anytime.

SEND COMMENT

This page was last modified on September 5, 2021

IMPROVE THIS PAGE