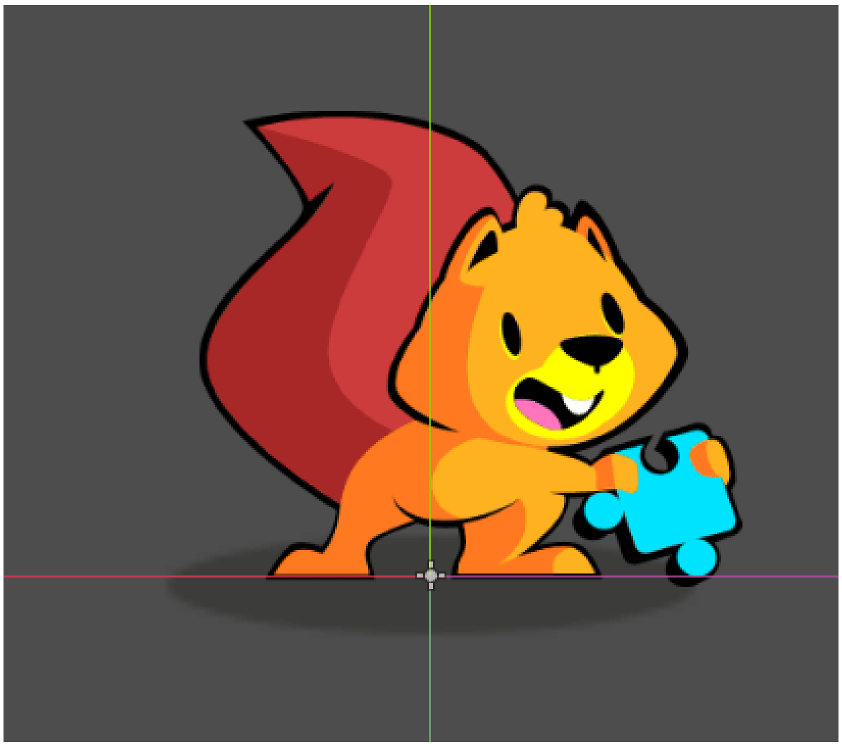
CREATING THE UNIT

By: Nathan Lovato - January 30, 2021

MENU

In this lesson, we'll create the unit, another core building block of the movement system.



The unit will only handle the visuals and the movement of a pawn on the game board.

This will allow us to test it even though we don't have the rest of the systems implemented yet. Coding independent nodes like that make it more efficient to reuse or modify them within and across projects.

You can think of the unit we will make as a lifeless pawn on a game board. It's like a little figurine that you can move around.

THE UNIT'S SCENE

THE UNIT'S SCENE

Let us start by designing the unit's scene.

We will use the *Path2D* and the *PathFollow2D* nodes to make our sprites move along a path.

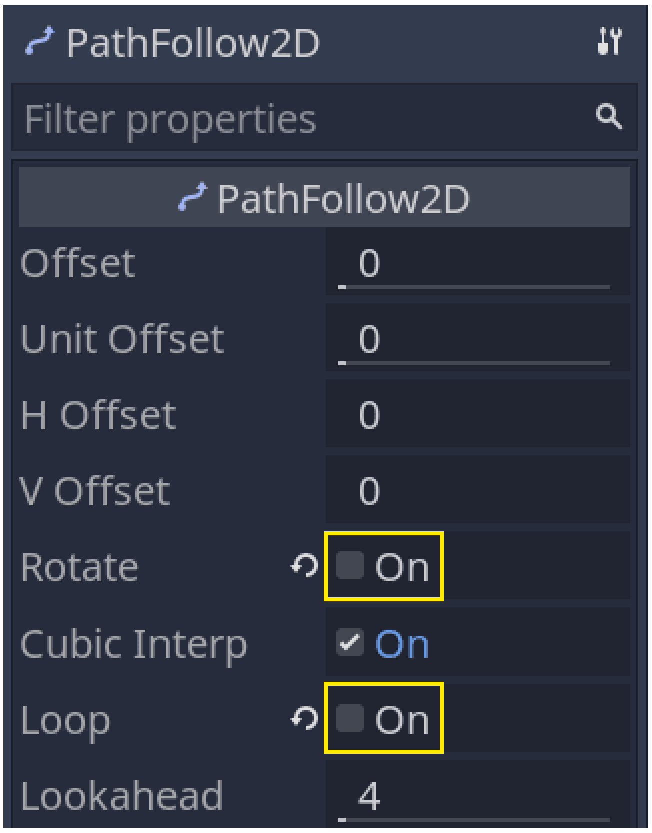
Path2D gives you tools to define a curve using a bezier spline, but you can also use it for straight lines.

The *PathFollow2D* node allows you to move along that path conveniently by animating an offset value.

Create a new scene with a *Path2D* node as its root and ad a *PathFollow2D* as a child.

Select the *Path2D* node and in the *Inspector*, clear its *Curv* property. With the *PathFollow2D* node, the curve would lock the node in place. We'll set the *Curve* in the script.

By default, when using *PathFollow2D*, the node will rotate its children along the path and cycle back to the start. Select the node and turn off the *Rotate* and *Loop* properties.


PathFollow2D

Filter properties

PathFollow2D

Offset	0
Unit Offset	0
H Offset	0
V Offset	0
Rotate	☐ On
Cubic Interp	✓ On
Loop	☐ On
Lookahead	4

Then, add two sprites as children of *PathFollow2D*. The first is there to represent the shadow while the other there is going to be the unit. I named them respectively *Shadow* and *Sprite*. Assign the texture `unit_shadow.svg` to the *Shadow* in the *Inspector* and set its *Modulate* color to a dark tone.

You can lower the scale of the *Sprite* node to `0.25`, as the assets we included in the project are large for this game.

We will use the *Path2D* and the *PathFollow2D* nodes to make our sprites move along a path.

Path2D gives you tools to define a curve using a bezier spline, but you can also use it for straight lines. Finally, add an *AnimationPlayer* as a child of the *Unit* node. We wil use it to make the *Sprite* blink when the player selects it.

BLINKING ANIMATION FOR SELECTED UNITS

We will use an animation to make the character blink or flash when it is selected. To do that, we'll use the *Modulate* property and its RAW color mode.

We'll also animate the *Sprite* node directly so that the *Shadow* doesn't change color with it.

First, though, let's define an animation named `idle`. It's an empty animation that resets the sprite's *Modulate* property. As animations in Godot override the nodes' properties, I recommend always setting up an animation to reset your node to it's default state.

Our `idle` animation should only have one key for the *Modulate* property. I typically give this kind of animation a duration of 0 seconds.

Now, we can work on the `selected` animation.

Now, we can work on the `selected` animation.

Let's temporarily assign a texture to the *Sprite* to preview the animation. Drag the `squirrel.png` to the node's *Texture*.

Select the *AnimationPlayer* and create a new animation named `selected`. It should loop and last a bit more than one second. I set its duration to `1.4` seconds.

The animation needs two keys that make the *Sprite -> Modulate* property cycle:

1. The first one, at the start of the timeline, should use the current modulate color, an opaque white.
2. The second one should use the RAW color mode with values higher than `1` in the R, G, and B channels. I used `1.5` for each of them. Doing so makes the sprite become brighter.

Now, you may clear the *Sprite* node's *Texture*. We'll use an exported variable in the script to assign a different sprite to each unit instance.

CODING THE UNIT

Let's move to the code. Attach a script to the *Unit* node with the following content.

We'll start with most of the node's properties and corresponding setter functions.

```
# Represents a unit on the game board.
# The board manages the Unit's position inside the
game grid.
# The unit itself is only a visual representation
that moves smoothly in the game world.
# We use the tool mode so the `skin` and
`skin_offset` below update in the editor.
tool
class_name Unit
extends Path2D

# Preload the `Grid.tres` resource you created in
the previous part.
export var grid: Resource =
preload("res://Grid.tres")
# Distance to which the unit can walk in cells.
# We'll use this to limit the cells the unit can
move to.
export var move_range := 6
# Texture representing the unit.
# With the `tool` mode, assigning a new texture to
this property in the inspector will update the
# unit's sprite instantly. See `set_skin()` below.
export var skin: Texture setget set_skin
# Our unit's skin is just a sprite in this demo
and depending on its size, we need to offset it so
# the sprite aligns with the shadow.
export var skin_offset := Vector2.ZERO setget
set_skin_offset
# The unit's move speed in pixels, when it's
moving along a path.
export var move_speed := 600.0

# Coordinates of the grid's cell the unit is on.
var cell := Vector2.ZERO setget set_cell
# Toggles the "selected" animation on the unit.
var is_selected := false setget set_is_selected

# Through its setter function, the `_is_walking`
property toggles processing for this unit.
# See `_set_is_walking()` at the bottom of this
```

```
code snippet.
var _is_walking := false setget _set_is_walking

onready var _sprite: Sprite =
$PathFollow2D/Sprite
onready var _anim_player: AnimationPlayer =
$AnimationPlayer
onready var _path_follow: PathFollow2D =
$PathFollow2D


# When changing the `cell`'s value, we don't want
to allow coordinates outside the grid, so we clamp
# them.

func set_cell(value: Vector2) -> void:
    cell = grid.clamp(value)


# The `is_selected` property toggles playback of
the "selected" animation.
func set_is_selected(value: bool) -> void:
    is_selected = value
    if is_selected:
        _anim_player.play("selected")
    else:
        _anim_player.play("idle")


# Both setters below manipulate the unit's Sprite
node.
# Here, we update the sprite's texture.
func set_skin(value: Texture) -> void:
    skin = value
    # Setter functions are called during the
node's `_init()` callback, before they entered the
    # tree. At that point in time, the `_sprite`
variable is `null`. If so, we have to wait to
    # update the sprite's properties.
    if not _sprite:
        # The yield keyword allows us to wait
until the unit node's `_ready()` callback ended.
        yield(self, "ready")
    _sprite.texture = value


func set_skin_offset(value: Vector2) -> void:
    skin_offset = value
    if not _sprite:
        yield(self, "ready")
    _sprite.position = value


func _set_is_walking(value: bool) -> void:
    _is_walking = value
    set_process(_is_walking)
```

Now, here’s the smooth movement logic.

```
# Emitted when the unit reached the end of a path
along which it was walking.
# We'll use this to notify the game board that a
unit reached its destination and we can let the
# player select another unit.
signal walk_finished
```

```
func _ready() -> void:
    # We'll use the `_process()` callback to move
the unit along a path. Unless it has a path to
    # walk, we don't want it to update every
frame. See `walk_along()` below.
    set_process(false)

    # The following lines initialize the `cell`
property and snap the unit to the cell's center on
the map.
    self.cell =
grid.calculate_grid_coordinates(position)
    position = grid.calculate_map_position(cell)


    if not Engine.editor_hint:
        # We create the curve resource here
because creating it in the editor prevents us from
        # moving the unit.
        curve = Curve2D.new()


# When active, moves the unit along its `curve`
with the help of the PathFollow2D node.
func _process(delta: float) -> void:
    # Every frame, the `PathFollow2D.offset`
property moves the sprites along the `curve`.
    # The great thing about this is it moves an
exact number of pixels taking turns into account.
    _path_follow.offset += move_speed * delta

    # When we increase the `offset` above, the
`unit_offset` also updates. It represents how far
you
    # are along the `curve` in percent, where a
value of `1.0` means you reached the end.
    # When that is the case, the unit is done
moving.
    if _path_follow.unit_offset >= 1.0:
        # Setting `_is_walking` to `false` also
turns off processing.
        self._is_walking = false
        # Below, we reset the offset to `0.0`,
which snaps the sprites back to the Unit node's
        # position, we position the node to the
center of the target grid cell, and we clear the
        # curve.
        # In the process loop, we only moved the
sprite, and not the unit itself. The following
        # lines move the unit in a way that's
transparent to the player.
        _path_follow.offset = 0.0
        position =
grid.calculate_map_position(cell)
        curve.clear_points()
        # Finally, we emit a signal. We'll use
this one with the game board.
        emit_signal("walk_finished")


# Starts walking along the `path`.
# `path` is an array of grid coordinates that the
function converts to map coordinates.
func walk_along(path: PoolVector2Array) -> void:
    if path.empty():
        return

    # This code converts the `path` to points on
the `curve`. That property comes from the `Path2D`
    # class the Unit extends.
    curve.add_point(Vector2.ZERO)
    for point in path:

curve.add_point(grid.calculate_map_position(point)
- position)
    # We instantly change the unit's cell to the
target position. You could also do that when it
    # reaches the end of the path, using
`grid.calculate_grid_coordinates()`, instead.
    # I did it here because we have the
coordinates provided by the `path` argument.
    # The cell itself represents the grid
coordinates the unit will stand on.
    cell = path[-1]

    # The `_is_walking` property triggers the move
animation and turns on `_process()`. See
    # `_set_is_walking()` below.
    self._is_walking = true
```

Here's the complete `Unit.gd` script without the comments

```
tool
class_name Unit
extends Path2D

signal walk_finished

export var grid: Resource =
preload("res://Grid.tres")
export var skin: Texture setget set_skin
export var move_range := 6
export var skin_offset := Vector2.ZERO setget
set_skin_offset
export var move_speed := 600.0

var cell := Vector2.ZERO setget set_cell
var is_selected := false setget set_is_selected

var _is_walking := false setget _set_is_walking

onready var _sprite: Sprite =
$PathFollow2D/Sprite
onready var _anim_player: AnimationPlayer =
$AnimationPlayer
onready var _path_follow: PathFollow2D =
$PathFollow2D


func _ready() -> void:
    set_process(false)

    self.cell =
grid.calculate_grid_coordinates(position)
    position = grid.calculate_map_position(cell)

    if not Engine.editor_hint:
        curve = Curve2D.new()


func _process(delta: float) -> void:
    _path_follow.offset += move_speed * delta

    if _path_follow.unit_offset >= 1.0:
        self._is_walking = false
        _path_follow.offset = 0.0
        position =
grid.calculate_map_position(cell)
        curve.clear_points()
        emit_signal("walk_finished")


func walk_along(path: PoolVector2Array) -> void:
    if path.empty():
        return

    curve.add_point(Vector2.ZERO)
    for point in path:

curve.add_point(grid.calculate_map_position(point)
- position)

    cell = path[-1]
    self._is_walking = true


func set_cell(value: Vector2) -> void:
    cell = grid.clamp(value)


func set_is_selected(value: bool) -> void:
    is_selected = value
    if is_selected:
        _anim_player.play("selected")
    else:
        _anim_player.play("idle")


func set_skin(value: Texture) -> void:
    skin = value
    if not _sprite:
        yield(self, "ready")
    _sprite.texture = value


func set_skin_offset(value: Vector2) -> void:
    skin_offset = value
    if not _sprite:
        yield(self, "ready")
    _sprite.position = value


func _set_is_walking(value: bool) -> void:
    _is_walking = value
    set_process(_is_walking)
```

TESTING THE UNIT

We'll now test our newly created class so you can see how it works. There's quite a lot of code at a glance, but the unit is fairly easy to use.

We'll create our *Main* scene right away, where we'll add other nodes as we code them.

Create a new *2D Scene* with a node named *Main* at its root

We prepared a map to fill the game's background. Create an instance of the `Map.tscn` file included in the start project and an instance of the *Unit* scene you just created.

In the `Unit.gd` script, at the end of its `_ready()` function, call `walk_along()`. It takes a `PoolVector2Array` of grid coordinates as its argument, and the easiest way to create one is to first create an array with the coordinates that we want to use. Here's an example.

```
func _ready() -> void:
    #...
    var points := [
        Vector2(2, 2),
        Vector2(2, 5),
        Vector2(8, 5),
        Vector2(8, 7),
    ]
    walk_along(PoolVector2Array(points))
```

Before running the game, you can enable *Visible Navigation* in the *Debug* menu. This will allow you to preview the curve generated by our function.

If you play the game you will see that the unit moves without any issue.

To get the result below, I assigned `squirrel.png` to the *Unit*'s *Skin* property.

In the next part, we will work on the cursor. It's the other element we need to implement unit selection and to issue orders through the game board.

← PREVIOUS

NEXT →

MADE BY

Nathan Lovato

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

3 COMMENTS

IM

IMADUMMY
July 6, 2021

After adding the walk_along() function. I got a "Attemp to call function 'add-point' in base 'null instance' on a null instance"

I think I'm lost

REPLY TO IMADUMMY

JI

JIM
October 30, 2021

So I ran into the same problem. For me it was because I was using the code with the comments as the Unit.gd script and that lead me to add the extra

> with the points var and calling walk_along() in the middle rather than at the end of the _ready() function. Once I moved it to the correct place in the _ready() function, it worked fine.
>
> REPLY TO JIM

JO

JOHN
September 28, 2021

When you add multiple units, the animation played is always on the first unit even if you select the other ones.

REPLY TO JOHN

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

IMPROVE THIS PAGE