Teaching myself Godot

start from the start

https://docs.godotengine.org/en/3.5/getting_started/introduction/introduction_to_godot.html

- In Godot, a game is a **tree** of **nodes** that you group together into **scenes**. You can then wire these nodes so they can communicate using **signals**.
- A scene can be a character, a weapon, a menu in the user interface, a single house, an entire level, or anything you can think of. Godot's scenes are flexible; they fill the role of both prefabs and scenes in some other game engines.
    - you can nest a scene in another scene (as a result, I guess)
- each scene is made of Nodes
- nodes are things that are actually like, programmed to interface with the engine

- Each scene has its own scene tree. The scene tree will show you which scenes and nodes are connected to that scene.
    - So, biggest thing to remember: scenes are both Unity-style scenes and Unity-style game objects.

- Nodes emit signals when some event occurs. This feature allows you to make nodes communicate without hard-wiring them in code.
    - model-view-controller design pattern, explained (https://gameprogrammingpatterns.com/observer.html)

        > Say we’re adding an achievements system to our game. It will feature dozens of different badges players can earn for completing specific milestones like “Kill 100 Monkey Demons”, “Fall off a Bridge”, or “Complete a Level Wielding Only a Dead Weasel”.
        >
        > This is tricky to implement cleanly since we have such a wide range of achievements that are unlocked by all sorts of different behaviors. If we aren’t careful, tendrils of our achievement system will twine their way through every dark corner of our codebase. Sure, “Fall off a Bridge” is somehow tied to the physics engine, but do we really want to see a call to unlockFallOffBridge() right in the middle of the linear algebra in our collision resolution algorithm?
        >
        > What we’d like, as always, is to have all the code concerned with one facet of the game nicely lumped in one place. The challenge is that achievements are triggered by

a bunch of different aspects of gameplay. How can that work without coupling the achievement code to all of them?

That's what the observer pattern is for. It lets one piece of code announce that something interesting happened without actually caring who receives the notification.

For example, we've got some physics code that handles gravity and tracks which bodies are relaxing on nice flat surfaces and which are plummeting toward sure demise. To implement the "Fall off a Bridge" badge, we could just jam the achievement code right in there, but that's a mess. Instead, we can just do:

```
void Physics::updateEntity(Entity& entity)
{
  bool wasOnSurface = entity.isOnSurface();
  entity.accelerate(GRAVITY);
  entity.update();
  if (wasOnSurface && !entity.isOnSurface())
  {
    notify(entity, EVENT_START_FALL);
  }
}
```

All it does is say, "Uh, I don't know if anyone cares, but this thing just fell. Do with that as you will."

The achievement system registers itself so that whenever the physics code sends a notification, the achievement system receives it. It can then check to see if the falling body is our less-than-graceful hero, and if his perch prior to this new, unpleasant encounter with classical mechanics was a bridge. If so, it unlocks the proper achievement with associated fireworks and fanfare, and it does all of this with no involvement from the physics code.

In fact, we can change the set of achievements or tear out the entire achievement system without touching a line of the physics engine. It will still send out its notifications, oblivious to the fact that nothing is receiving them anymore.

- (more implementation continues from there. We don't need it.)

- "For example, buttons emit a signal when pressed. You can connect to this signal to run code in reaction to this event, like starting the game or opening a menu."

https://docs.godotengine.org/en/3.5/getting_started/introduction/first_look_at_the_editor.html

- top of the screen lets you switch what you're looking at. if you're "stuck" looking at a script, go click 2d or 3d or whatever you're working with.

- Press F1 and type shit in to look at reference material. No need to go to the site, it's built-in.

https://docs.godotengine.org/en/3.5/tutorials/scripting/c_sharp/c_sharp_differences.html

C# generally uses PascalCase instead of the snake_case used in GDScript and C++.

Math functions¶

Math global functions, like abs, acos, asin, atan and atan2, are located under Mathf as Abs, Acos, Asin, Atan and Atan2. The PI constant can be found as Mathf.Pi.

Random functions¶

Random global functions, like rand_range and rand_seed, are located under GD. Example: GD.RandRange and GD.RandSeed.

Other functions¶

Many other global functions like print and var2str are located under GD. Example: GD.Print and GD.Var2Str.

Example of implications

I have been trying to use the “print” function.

- “print” on its own returns “this doesn’t exist in this context”.
- “Print” is what the function is in the C# implementation.
- “GD.Print” works because we are already “using Godot;”, so it goes into Godot.GD and finds the Print function.
- Alternatively (and more messily), I can specifically add “using Godot.GD;”, and then I can simply call “Print” without the “GD” part.

  ```
  public override void _Ready()
  {
      GD.Print("Goodbye, world");
  }
  ```

This works without adding “using Godot.GD;”.

We’ll add more to this later. Now onto the hex map tutorial.

Trying to make the line-drawer work

https://docs.godotengine.org/en/3.5/tutorials/best_practices/what_are_godot_classes.html?

The engine provides built-in classes like Node. You can extend those to create derived types using a script.

These scripts are not technically classes. Instead, they are resources that tell the engine a sequence of initializations to perform on one of the engine's built-in classes.