# Adding new entities (super streamlined)

This project makes it easy to add new things to the world without touching `map.lua`. Pick the workflow that matches your need:

- Easiest: Type-based spawn (no rules) — define a factory, set Tiled object `type`, done.
- Named variants: Keep multiple presets (box1, box2) and spawn by name or `variant` property.
- One-off/singletons: Use a tiny factory that returns a shared module (e.g., `statue`).

## 1) The minimal path: use object.type

Best when you just want to drop objects in Tiled and have them spawn.

Steps:

1. Create a factory at `spawning/factories/<type>.lua` that implements:
   - `create(world, x, y, obj, cfg, ctx) -> instance`
2. Register the new type in `data/spawn_registry.lua` under `types`:

```
types = {
  mytype = { factory = 'spawning.factories.mytype', defaults = { /* optional */ } },
}
```

3. In Tiled (entity object layer), set your object's Type to `mytype`.
4. Optional: add a custom property `variant = "big"` to pick a preset (see next section).

That's it. No map.lua changes, no rules required.

Spawner behavior with this flow:

- If no matching rule is found, it will spawn by `object.type` if that type exists in the registry.
- Merges `defaults <- variant preset <- object.properties` into the `cfg` passed to your factory.

## 2) Variants and presets

If your type has different flavors (sizes, behavior), define them under `variants` in the registry.

Example registry excerpt:

```
return {
  types = {
    box = { factory = 'spawning.factories.box', defaults = { type='dynamic', restitution=0.2 } },
  },
  variants = {
    box = {
      box1 = { w=16, h=16 },
      box2 = { w=28, h=28 },
      big  = { w=48, h=48 },
    },
  },
}
```

Pick a preset in one of two ways:

- By name (strict): add a rule with `fromName` (name must exactly match an existing variant key)

  ```
  rules = {
    { when = { namePrefix = 'box' }, type = 'box', variant = { fromName = true } },
  }
  ```

  Usage in Tiled: set `name = box1` or `box2`. If the name isn't a known variant, it won't spawn.

- By property (flexible): set `variant = "big"` on the object properties; no rule required.
    - This works only when you are spawning by `object.type` or a rule that doesn't derive the variant from the name.

Merge order recap:

- The final `cfg` your factory receives is: `types.defaults <- variants[preset] <- object.properties` .

## 3) Writing a factory

Factories are tiny adapters. They take Tiled objects and build your entity instance.

Contract:

- `create(world, x, y, obj, cfg, ctx)`
    - `world` : love.physics World
    - `x, y` : object center in pixels
    - `obj` : the full Tiled object (name, type, width, height, properties)
    - `cfg` : merged config from registry + object
    - `ctx` : extra context ( `{ registry, level, map }` )

Example (rectangle body placeholder):

```lua
---@diagnostic disable: undefined-global
local M = {}
function M.create(world, x, y, obj, cfg, ctx)
  local w = cfg.w or obj.width or 16
  local h = cfg.h or obj.height or 16
  local body = love.physics.newBody(world, x, y, cfg.bodyType or 'dynamic')
  local shape = love.physics.newRectangleShape(w, h)
  local fixture = love.physics.newFixture(body, shape)
  fixture:setUserData({ kind = obj.type or 'mytype', name = obj.name })
```

```
    return {
      body = body, shape = shape, fixture = fixture,
      kind = obj.type or 'mytype', name = obj.name,
      update = function(self, dt) end,
      draw = function(self)
        love.graphics.setColor(0.9, 0.8, 0.3, 1)
        love.graphics.rectangle('line', x - w/2, y - h/2, w, h)
        love.graphics.setColor(1,1,1,1)
      end,
    }
  end
  return M
```

For real entities, just require your module and call its constructor (see `spawning/factories/box.lua` , `ball.lua` , `bell.lua` , `statue.lua` ).

Shortcut: copy the template at `spawning/factories/_template_generic.lua` and adapt it.

## 4) Project structure: where files go

- Factories: `spawning/factories/<type>.lua`
- Registry: `data/spawn_registry.lua`
- Entity modules: place them where you want (e.g., project root or `entities/` ). Your factory can `require` them.
- Docs: `docs/*`

Map integration is already done:

- `map.lua` calls the Spawner, collects results into `state.entities` and draws/updates them.
- New types don’t require changes to map code.

## 5) Singletons and shared modules

Some entities are singletons (like the `statue`) that don't have physics. Your factory can just return a shared module:

```lua
-- spawning/factories/statue.lua
local M = {}
function M.create(world, x, y, obj, cfg, ctx)
  local statue = require('statue')
  statue.load(x, y)
  return statue
end
return M
```

In the registry:

```lua
types = { statue = { factory = 'spawning.factories.statue', defaults = {} } }
variants = { statue = { statue = {} } }
rules = {
  { when = { namePrefix = 'statue' }, type = 'statue', variant = { fromName = true } },
}
```

## 6) Troubleshooting

- It didn't spawn:
  - If using name-based rules, make sure `name` exactly matches an existing variant under `variants[type]`.
  - If using `object.type` flow, confirm you registered the type under `types`.
  - Check the console: the Spawner prints require() failures for factories.
- Wrong size/behavior:
  - Verify the preset under `variants` and any custom object properties.

    - Remember merge order: object properties override presets and defaults.
- Need per-type post-setup?
    - Add a method `postSpawn(ctx)` to your instance and call it from your factory; or extend your module constructor to accept `ctx`.

## 7) Quick checklist when adding a brand-new thing

- [ ] Create `spawning/factories/<type>.lua` with `create(...)`
- [ ] Add `types.<type>` entry in `data/spawn_registry.lua`
- [ ] Optional: add `variants.<type>` presets and either:
    - add a rule for strict name matching, or
    - use `object.properties.variant = "preset"`
- [ ] In Tiled, set `type = <type>` (and name/variant if applicable)
- [ ] Run the game: the Map already spawns, updates, and draws `state.entities` automatically

## 8) Object module templates in this repo

When you need a new game object module (not just a factory), start from these:

- `foo.lua` — full-featured, well-documented template showing physics, update/draw, and batch helpers
- `bar.lua` — minimal, bare-bones rectangle object with update/draw and cleanup hooks

Tips:

- Keep your module responsible for its own physics and visuals. The factory should only translate Tiled data into a call like `MyThing.new(world, x, y, opts)`.
- Prefer pixel units everywhere. This project sets `love.physics.setMeter(1)`, so 1 pixel = 1 meter.