SaveState: Non-destructive runtime state overlay

Keep runtime changes (like `bell1.isSolved`) completely separate from your Tiled maps. This module overlays per-map, per-entity properties on top of the live STI map, so content stays read-only and designers can safely iterate.

Why this instead of editing maps?

- No touching exported Tiled files at runtime
- Works across level transitions
- Easy to inspect and version control (saves as a Lua table)
- Decoupled from game logic; applied via `GameContext`

Data model

```
-- data[mapId][entityName][prop] = value
{
  ["tiled/map/1"] = {
    bell1 = { isSolved = true }
  }
}
```

- mapId: a stable id per level. We use the base path (e.g., `tiled/map/1`). You can swap to a custom Tiled map property later with one line.
- entityName: the name in your `entity` layer (e.g., `bell1`).
- prop: any boolean/number/string your game logic reads from the entity's `properties`.

Session-only by default

- By default, the module runs in ephemeral mode and does NOT write to disk. All changes live only for the current run.
- To enable persistence later, set:

```
SaveState.setPersistent(true)
SaveState.init('save/slot1.lua')
```

File format and location (when persistence is enabled)

- Saves to `save/slot1.lua` by default (a Lua file with `return { ... }`).
- Stored in LÖVE's save directory (based on `t.identity` in `conf.lua`).
  - Windows: `%AppData%/LOVE/2d_platformer_empty`.

API

- `SaveState.init(path)` → Initialize and load existing data if present.
- `SaveState.setCurrentMapId(id)` / `SaveState.getCurrentMapId()` → Set/get the active map.
- `SaveState.setEntityProp(mapId, entity, key, value)` → Write a value for a specific map.
- `SaveState.setEntityPropCurrent(entity, key, value)` → Write for the current map.
- `SaveState.applyToMap(mapId)` / `SaveState.applyToMapCurrent()` → Apply saved values to the live STI map using `GameContext.setEntityProp`.
- `SaveState.save()` / `SaveState.load()` / `SaveState.reset()` → Persistence controls.

Integration examples

main.lua (startup)

```
local SaveState = require('save_state')

function love.load()
```

```lua
    -- One-shot session (no disk):
    SaveState.setPersistent(false)
    SaveState.init('save/slot1.lua')

    Map:load(2)
    SaveState.setCurrentMapId(Map:getCurrentLevel())
    SaveState.applyToMapCurrent()
  end
```

map.lua (after loading a map and on transitions)

```lua
-- Inside Map:init (after loading STI and GameContext.setLevel(level))
SaveState.setCurrentMapId(base)      -- e.g., 'tiled/map/1'
SaveState.applyToMapCurrent()        -- overlay props onto the live map

-- After switching levels in _switchLevelAndTeleport
SaveState.setCurrentMapId(destMapPath)
SaveState.applyToMapCurrent()
```

bell.lua (persist on success)

```lua
-- When the code sequence completes
SaveState.setEntityPropCurrent('bell1', 'isSolved', true)
if SaveState.persistent then SaveState.save() end
```

Common patterns

- Save timing: Save immediately on important changes, or batch and save on map switch / `love.quit` .
- Multiple slots: Call `SaveState.init('save/slot2.lua')` for a different slot.
- Designer editing: the file is plain Lua. Designers can open `save/slot1.lua` , tweak, or reset.

- Custom map ids: If you add `levelId` to your Tiled map properties, use that instead of the path and call `SaveState.setCurrentMapId(levelId)`.

Gotchas and tips

- Always call `applyToMapCurrent()` after the map is (re)constructed and `GameContext` points at it.
- If an entity is renamed in Tiled, old saved keys won’t apply—safe no-op.
- Consider a debug panel entry to show current mapId and how many overrides are active.