Malikania Engine

Moteur de MMORPG

Introduction

MMORPG 2d

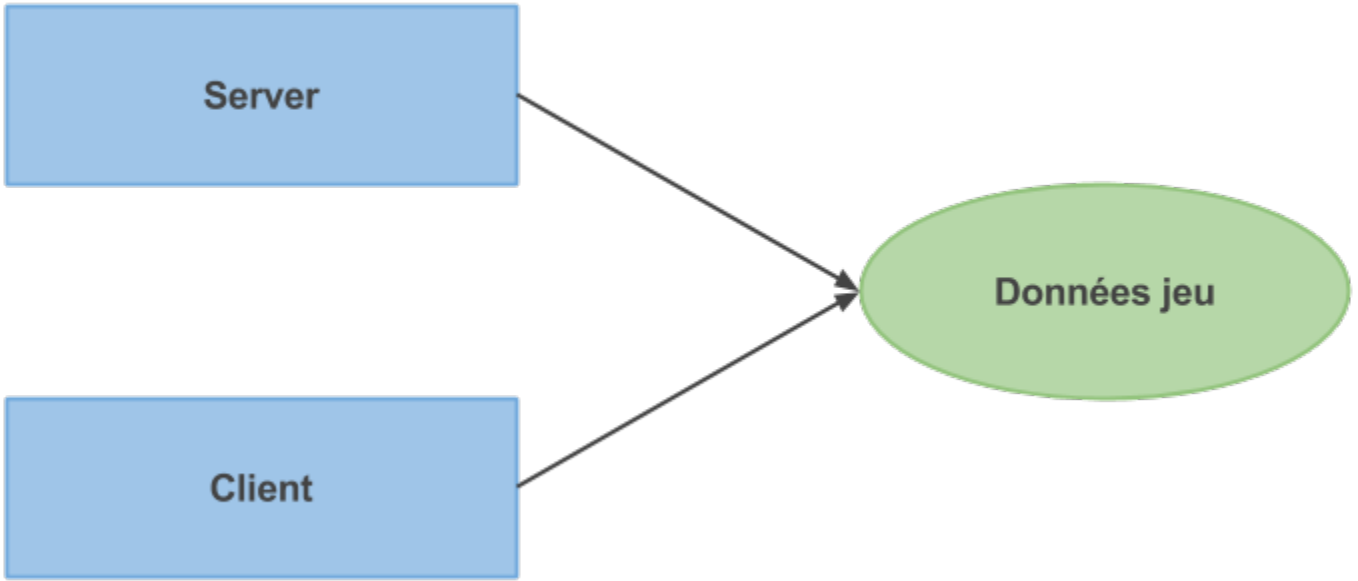
Combat tour par tour aléatoire (FF)

3 niveaux par sort

Idée

- Moteur de MMORPG portable
- Développement de jeu rapide (Lua)
- Facilité de création
- Contributions communautaires
- Flexible à l’extrême

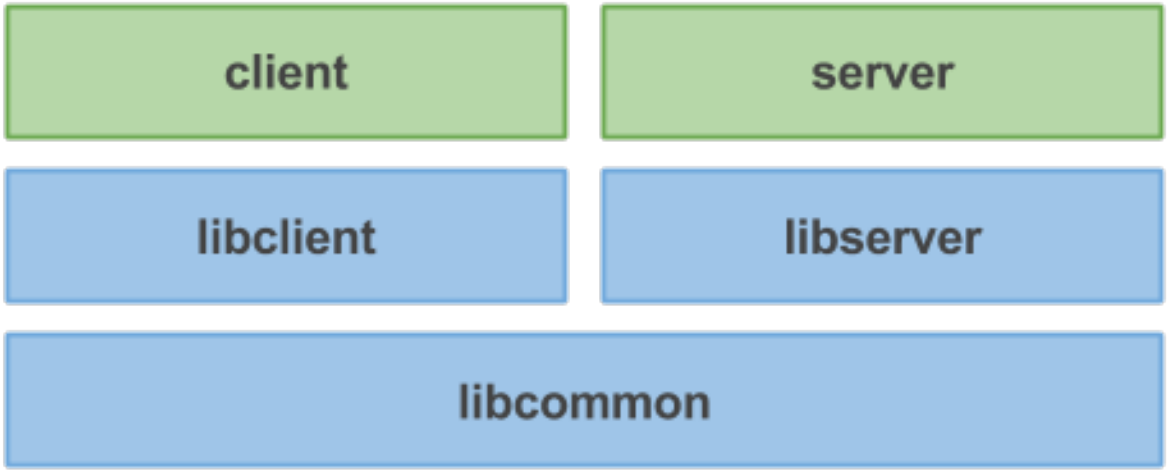
Concept (simple)



Concept (simple)

- Un serveur
- Un client
- Des données partagées

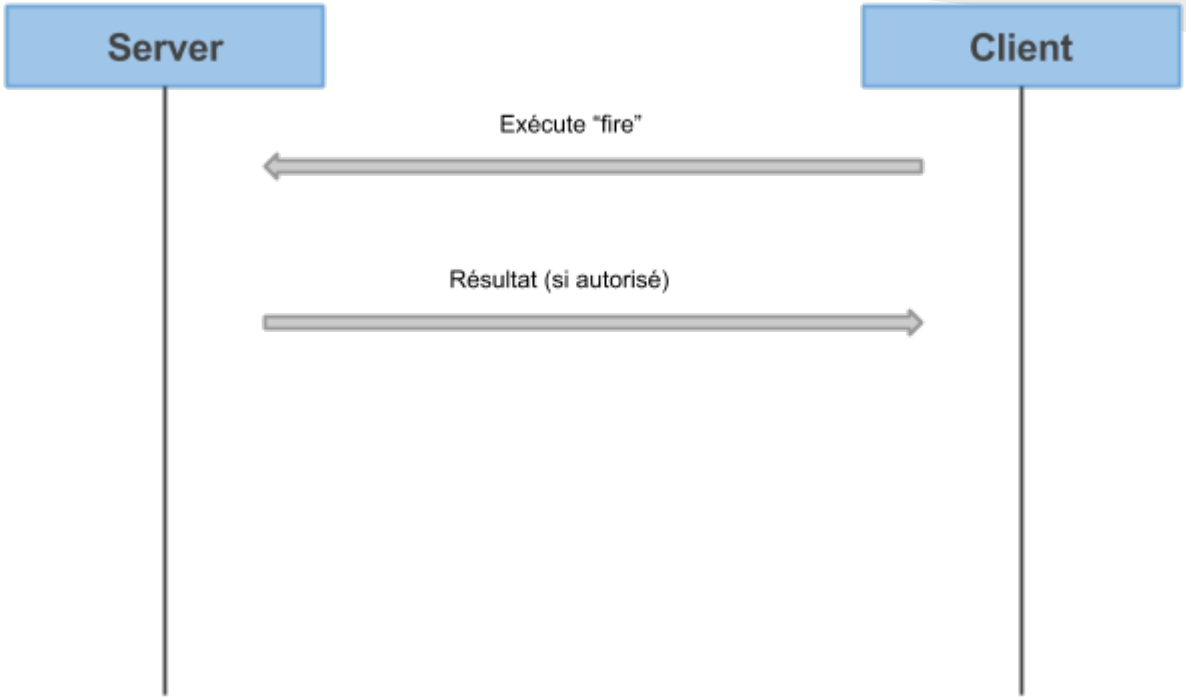
Concept (évolué)



Concept (évolué)

- Tout modulaire
- Faire son client modifié (libclient)

Architecture

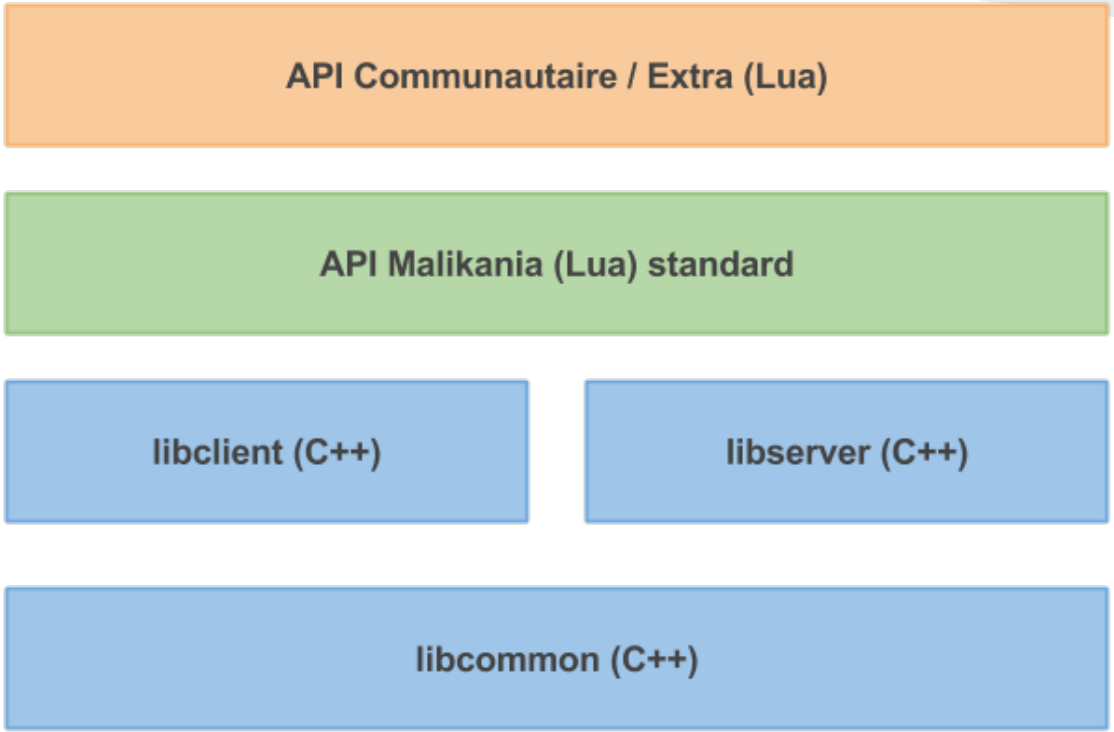


Architecture

Tout exécuté côté serveur

Client responsable graphique

Lua


API Communautaire / Extra (Lua)
API Malikania (Lua) standard
libclient (C++)
libserver (C++)
libcommon (C++)

Lua (intérêt)

- Pas de recompilation
- Téléchargement (serveurs)
- Partage
- Simple
- Évaluation paresseuse

Lua (fonctionnement)

- Partagé (serveur / client)
- Chargé au démarrage
- Renseigne des paramètres / fonctions
- Est utilisé quand nécessaire

Lua API

- malikania.server
- malikania.client
- malikania.battle
- malikania.spell
- …
- malikania.gui.button
- malikania.gui.frame

Exemple (sort)

```
local spell    = require "malikania.spell"

local s = {
    name     = "fire",
    title    = "Fire",
    help     = "Burn your enemies",
    type     = spell.type.Blast,

    -- leveled parameters
    mp       = { 3, 3, 4 },
    range    = { 3, 4, 5 },
    radius   = 3,                           -- Same for three levels, but only used at level 3
    select   = {
                    spell.select.Single,
                    spell.select.Single,
                    spell.select.Circle
               },

    action   = function (battle, owner, targets)
               end,

    draw     = function (battle, owner, targets, result)
               end
}

return s
```

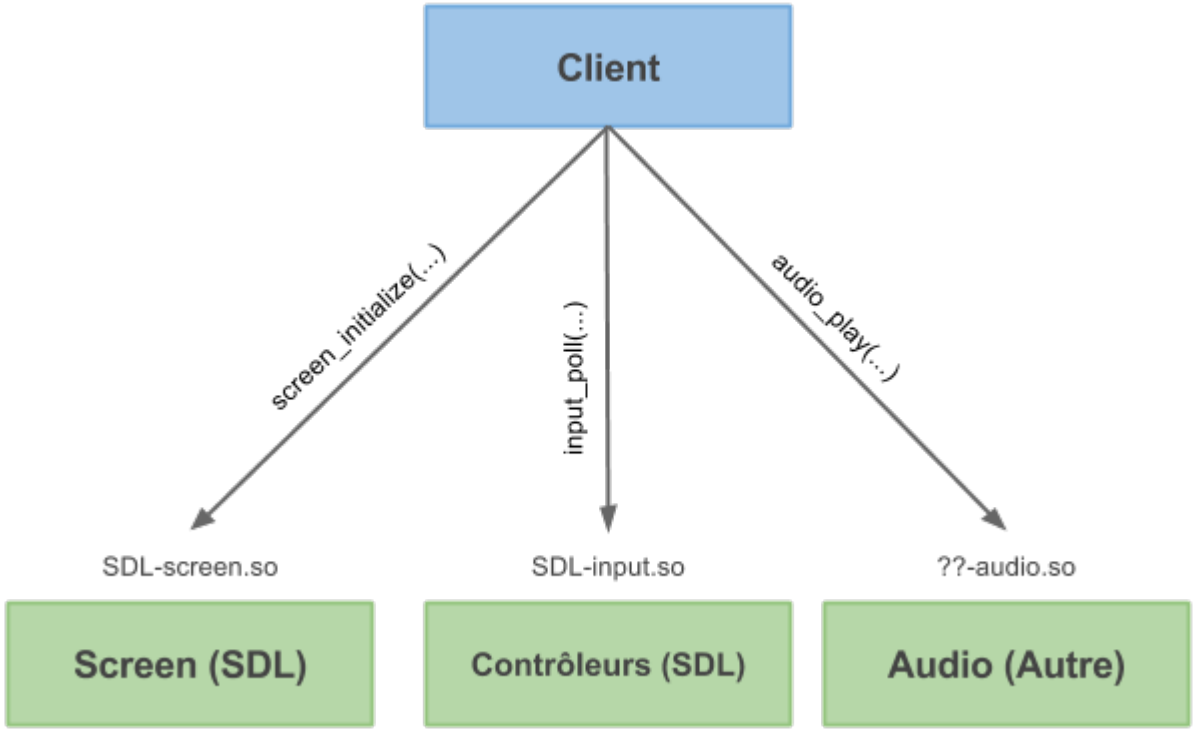
Exemple (objet)

```
local i = {
    name        = "elixir",
    title       = "Elixir",
    help        = "Recover all HP",
    weight      = 1,
}

function i.use(owner)
    owner:heal(35000)
end

return i
```

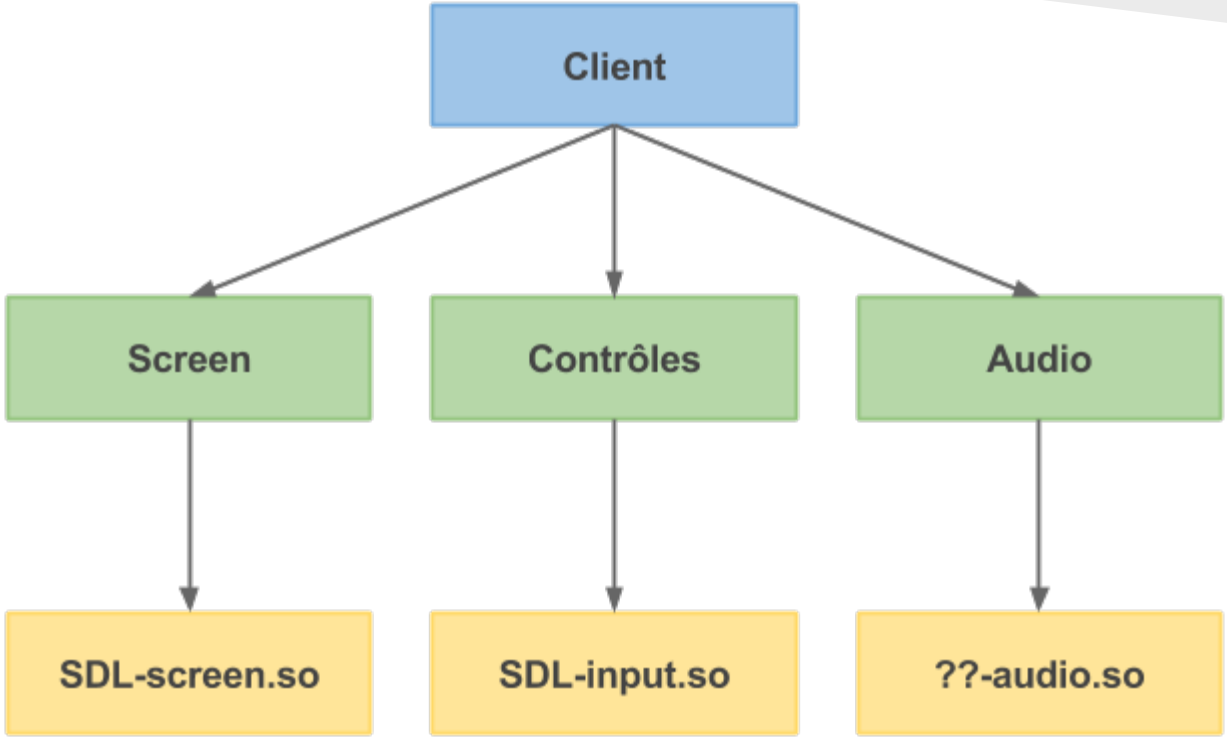
Système de backend (client)

- Portage simplifié
- Chargé dynamiquement
- Peut-être externe
- Transparent du client

Système de backend (client)



Système de backend (client)



Surcharge graphique

Possibilité de personnaliser

Changer les états

Exemple (fenêtre de code)

```
local client    = require "malikania.client"

local frame     = require "malikania.gui.frame"
local button    = require "malikania.gui.button"
local input     = require "malikania.gui.input"
local label     = require "malikania.gui.label"

function frame.default.lock(owner, houseId)
    local title = string.format("Enter %s house", owner:name())
    local f     = frame.new(title, 200, 90)
    local in    = input.new(input.Password)
    local lbl   = label.new("Password: ")
    local b     = button.new()

    b:connect("click",
        function ()
               client.openhouse(in:text())
        end
    )

    return f
end
```

Système à états (client)

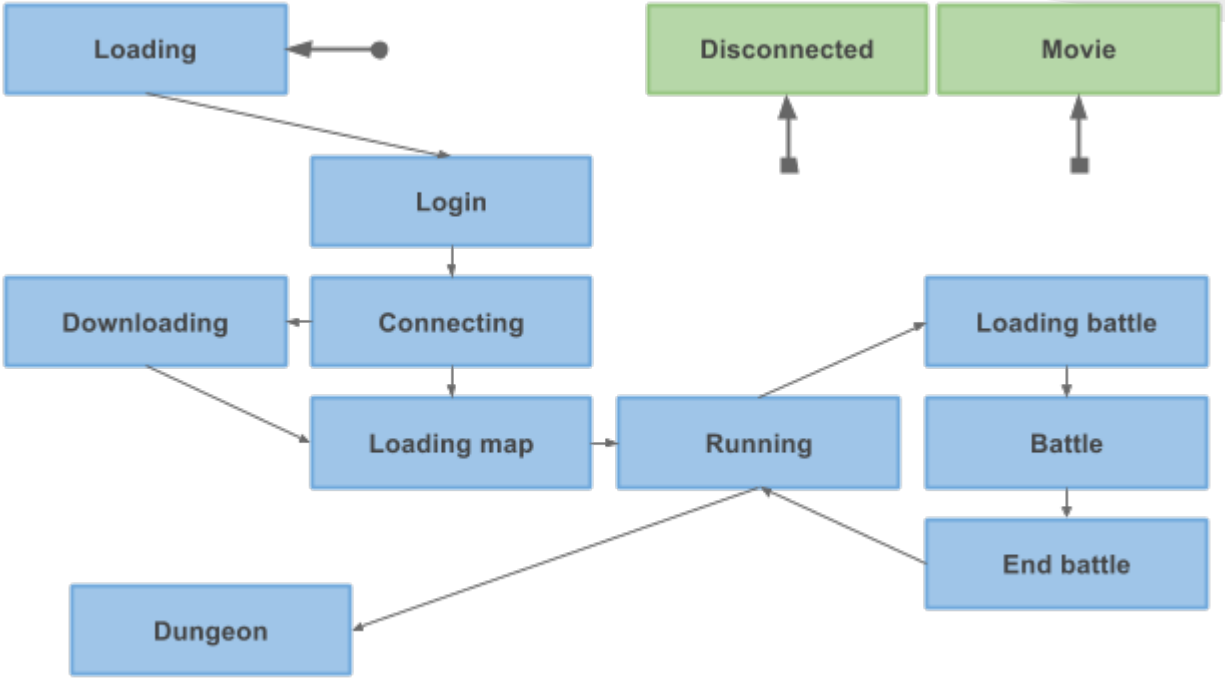
Réimplémenter des états clients

Dessiner différemment

Système à états (client)

- malikania.client.state.*
- malikania.client.state.loading
- malikania.client.state.login
- malikania.client.state.downloading

Système à états (client)



Réimplementer loading

```
local state     = require "malikania.client.state"
local event     = require "malikania.client.event"
local graphics  = require "malikania.graphics"

state.loading   = { }

local my_large_gpl_license_has_been_shown = false

function state.loading.update(dt)
    -- Changing state
    if my_large_gpl_license_has_been_shown then
        return state.login
    end
end

function state.loading.event(ev)
    if ev.type == event.Key and ev.key.which == event.key.Space then
        my_large_gpl_license_has_been_shown = true
    end
end

function state.loading.draw()
    graphics.print(10, 20, "GPL license here")
end
```