hump.gamestate

```
Gamestate = require "hump.gamestate"
```

A gamestate encapsulates independent data and behaviour in a single table.

A typical game could consist of a menu-state, a level-state and a game-over-state.

Example:

```
local menu = {} -- previously: Gamestate.new()
local game = {}

function menu:draw()
    love.graphics.print("Press Enter to continue", 10, 10)
end

function menu:keyreleased(key, code)
    if key == 'return' then
        Gamestate.switch(game)
    end
end

function game:enter()
    Entities.clear()
    -- setup entities here
end

function game:update(dt)
    Entities.update(dt)
end

function game:draw()
    Entities.draw()
end

function love.load()
    Gamestate.registerEvents()
    Gamestate.switch(menu)
end
```

List of Functions

- :func:`Gamestate.new() <Gamestate.new>`
- :func:`Gamestate.switch(to, ...) <Gamestate.switch>`
- :func:`Gamestate.current() <Gamestate.current>`
- :func:`Gamestate.push(to, ...) <Gamestate.push>`
- :func:`Gamestate.pop(...) <Gamestate.pop>`
- :func:`Gamestate.<callback>(...) <Gamestate.<callback>>`
- :func:`Gamestate.registerEvents([callbacks]) <Gamestate.registerEvents>`

Gamestate Callbacks

A gamestate can define all callbacks that LÖVE defines. In addition, there are callbacks for initalizing, entering and leaving a state:

`init()`
: Called once, and only once, before entering the state the first time. See :func:`Gamestate.switch`.

`enter(previous, ...)`
: Called every time when entering the state. See :func:`Gamestate.switch`.

`leave()`
: Called when leaving a state. See :func:`Gamestate.switch` and :func:`Gamestate.pop`.

`resume()`
: Called when re-entering a state by :func:`Gamestate.pop`-ing another state.

`update()`
: Update the game state. Called every frame.

`draw()`
: Draw on the screen. Called every frame.

`focus()`
: Called if the window gets or loses focus.

`keypressed()`
: Triggered when a key is pressed.

`keyreleased()`
: Triggered when a key is released.

`mousepressed()`
: Triggered when a mouse button is pressed.

`mousereleased()`
: Triggered when a mouse button is released.

`joystickpressed()`
: Triggered when a joystick button is pressed.

`joystickreleased()`
: Triggered when a joystick button is released.

`quit()`
: Called on quitting the game. Only called on the active gamestate.

When using :func:`Gamestate.registerEvents`, all these callbacks will be called by the corresponding LÖVE callbacks and receive the same arguments (e.g. `state:update(dt)` will be called by `love.update(dt)`).

Example:

```
menu = {} -- previously: Gamestate.new()

function menu:init()
    self.background = love.graphics.newImage('bg.jpg')
    Buttons.initialize()
end

function menu:enter(previous) -- runs every time the state is entered
    Buttons.setActive(Buttons.start)
end

function menu:update(dt) -- runs every frame
```

```
    Buttons.update(dt)
end

function menu:draw()
    love.graphics.draw(self.background, 0, 0)
    Buttons.draw()
end

function menu:keyreleased(key)
    if key == 'up' then
        Buttons.selectPrevious()
    elseif key == 'down' then
        Buttons.selectNext()
    elseif
        Buttons.active:onClick()
    end
end

function menu:mousereleased(x,y, mouse_btn)
    local button = Buttons.hovered(x,y)
    if button then
        Button.select(button)
        if mouse_btn == 'l' then
            button:onClick()
        end
    end
end
```

Function Reference

Deprecated: Use the table constructor instead (see example)

Declare a new gamestate (just an empty table). A gamestate can define several callbacks.

Example:

```
menu = {}
-- deprecated method:
menu = Gamestate.new()
```

Switch to a gamestate, with any additional arguments passed to the new state.

Switching a gamestate will call the `leave()` callback on the current gamestate, replace the current gamestate with `to`, call the `init()` function if, and only if, the state was not yet inialized and finally call `enter(old_state, ...)` on the new gamestate.

> Note
>
> Processing of callbacks is suspended until `update()` is called on the new gamestate, but the function calling :func:`Gamestate.switch` can still continue - it is your job to make sure this is handled correctly. See also the examples below.

Examples:

```
Gamestate.switch(game, level_two)
```

```
-- stop execution of the current state by using return
if player.has_died then
    return Gamestate.switch(game, level_two)
end

-- this will not be called when the state is switched
player:update()
```

Returns the currently activated gamestate.

Example:

```
function love.keypressed(key)
    if Gamestate.current() ~= menu and key == 'p' then
        Gamestate.push(pause)
    end
end
```

Pushes the `to` on top of the state stack, i.e. makes it the active state. Semantics are the same as `switch(to, ...)`, except that `leave()` is *not* called on the previously active state.

Useful for pause screens, menus, etc.

> Note
>
> Processing of callbacks is suspended until `update()` is called on the new gamestate, but the function calling `GS.push()` can still continue - it is your job to make sure this is handled correctly. See also the example below.

Example:

```
-- pause gamestate
Pause = Gamestate.new()
function Pause:enter(from)
    self.from = from -- record previous state
end

function Pause:draw()
    local W, H = love.graphics.getWidth(), love.graphics.getHeight()
    -- draw previous screen
    self.from:draw()
    -- overlay with pause message
    love.graphics.setColor(0,0,0, 100)
    love.graphics.rectangle('fill', 0,0, W,H)
    love.graphics.setColor(255,255,255)
    love.graphics.printf('PAUSE', 0, H/2, W, 'center')
end

-- [...]
function love.keypressed(key)
    if Gamestate.current() ~= menu and key == 'p' then
```

```
        return Gamestate.push(pause)
    end
end
```

Calls `leave()` on the current state and then removes it from the stack, making the state below the current state and calls `resume(...)` on the activated state. Does *not* call `enter()` on the activated state.

> **Note**
>
> Processing of callbacks is suspended until `update()` is called on the new gamestate, but the function calling `GS.pop()` can still continue - it is your job to make sure this is handled correctly. See also the example below.

Example:

```
-- extending the example of Gamestate.push() above
function Pause:keypressed(key)
    if key == 'p' then
        return Gamestate.pop() -- return to previous state
    end
end
```

Calls a function on the current gamestate. Can be any function, but is intended to be one of the :ref:`callbacks`. Mostly useful when not using :func:`Gamestate.registerEvents`.

Example:

```
function love.draw()
    Gamestate.draw() -- <callback> is `draw'
end

function love.update(dt)
    Gamestate.update(dt) -- pass dt to currentState:update(dt)
end

function love.keypressed(key, code)
    Gamestate.keypressed(key, code) -- pass multiple arguments
end
```

Overwrite love callbacks to call `Gamestate.update()`, `Gamestate.draw()`, etc. automatically. `love` callbacks (e.g. `love.update()`) are still invoked as usual.

This is by done by overwriting the love callbacks, e.g.:

```
local old_update = love.update
function love.update(dt)
    old_update(dt)
    return Gamestate.current:update(dt)
end
```

> Note
>
> Only works when called in love.load() or any other function that is executed *after* the whole file is loaded.

Examples:

```
function love.load()
    Gamestate.registerEvents()
    Gamestate.switch(menu)
end

-- love callback will still be invoked
function love.update(dt)
    Timer.update(dt)
    -- no need for Gamestate.update(dt)
end
```

```
function love.load()
    -- only register draw, update and quit
    Gamestate.registerEvents{'draw', 'update', 'quit'}
    Gamestate.switch(menu)
end
```