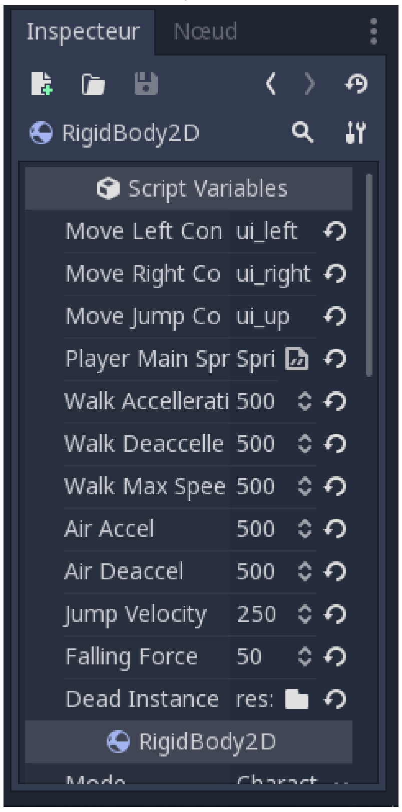
Summary:

Terms:

- A Node: Game object on Godot Engine
- exports variables: are parameters integrated to the inspector (the interface for modifying the data on a node) for the execution of the functions of the various elements listed below.



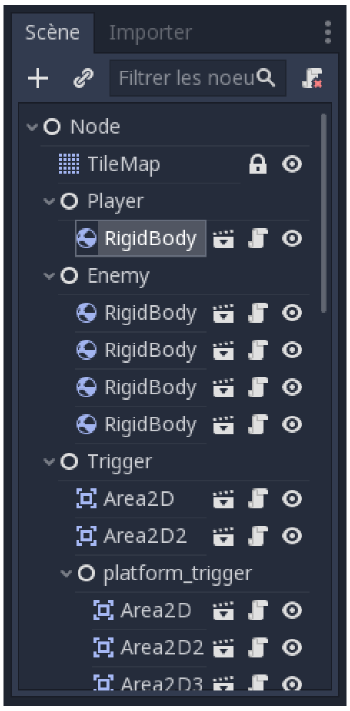
Move Left Con	ui_left
Move Right Co	ui_right
Move Jump Co	ui_up
Player Main Spr	Spri
Walk Accellerati	500
Walk Deaccelle	500
Walk Max Spee	500
Air Accel	500
Air Deaccel	500
Jump Velocity	250
Falling Force	50
Dead Instance	res:

For example, for the Player element, the variable export (or variables script)

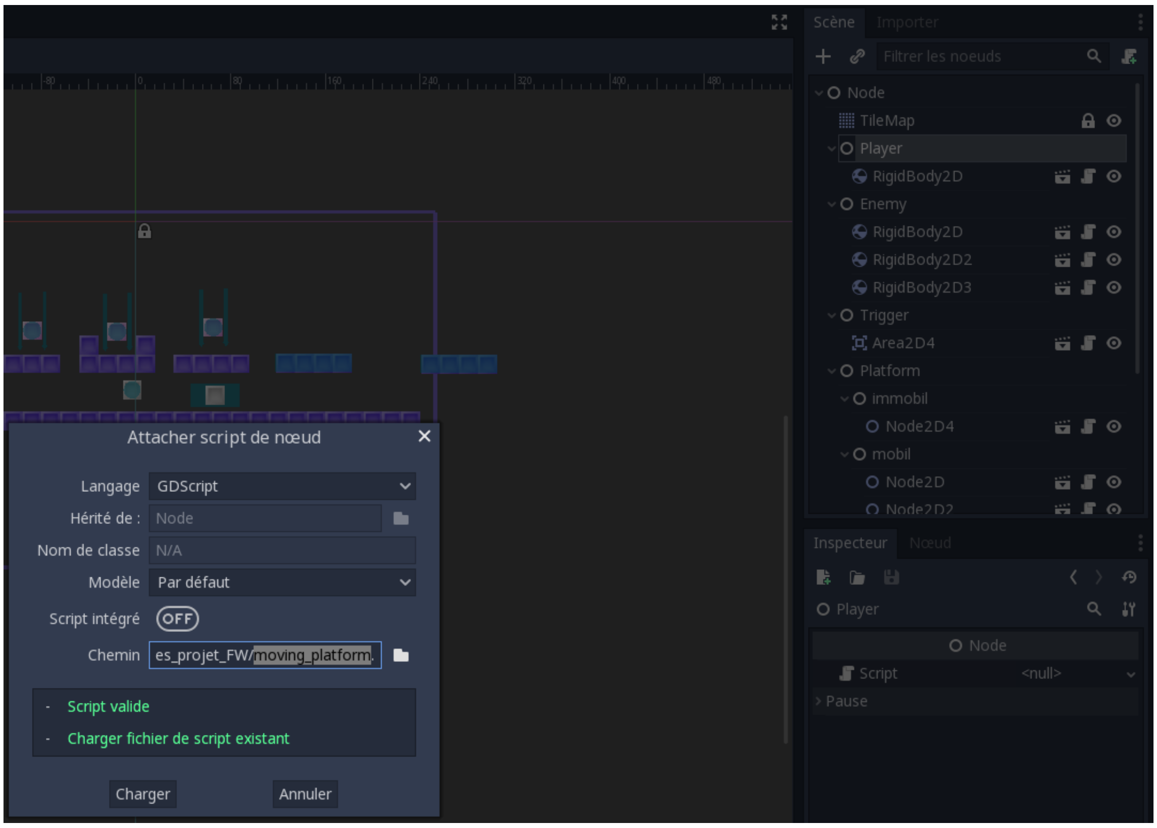
Walk_acceleration are a parameter for the application of the moving function, here, the acceleration parameter.

- A tree: In Godot Engine, all element of the game (Nodes) are placed in a tree. It is like a family tree, an interface that lists the parents and their children in the order of relationship.
- Scenes: Allows you to host a Node tree, such as a level for example. When creating a new game element (like a player) we will place the tree of nodes of this element in a new scene that we save once completed.

For example, for the Player, it is composed of collisions, images, physics and sound. The tree of its nodes is saved in a different scene of the level. Then, it is possible to import the Player scene to the level tree.



- A script: scripts are files containing code, they are in ".gd" format and can be attached to a Node. You can attach a script to a node via the button at the top right of the "scene" menu. The "attach node script" menu opens and you then assign a path, then click on load.



Attacher script de nœud

Langage: GDScript
Hérité de : Node
Nom de classe: N/A
Modèle: Par défaut
Script intégré: OFF
Chemin: es_projet_FW/moving_platform.

- Script valide
- Charger fichier de script existant

Charger Annuler

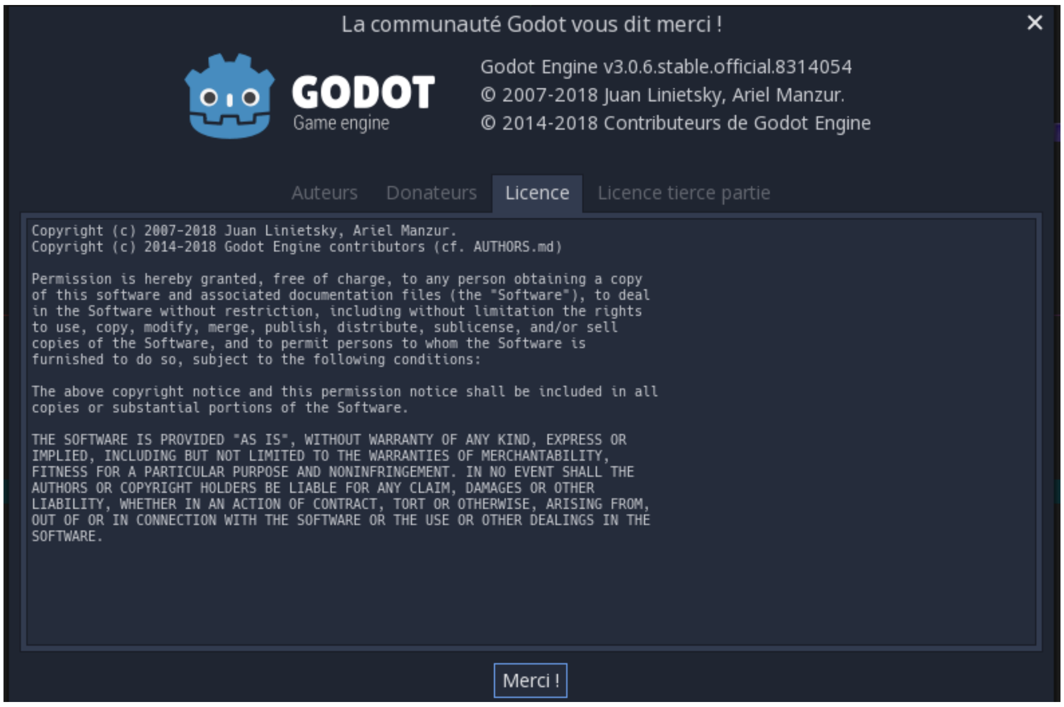
When the scene is launched, the script will be read and the function executed.

Links:

- Understand the physics of Godot Engine : (http://docs.godotengine.org/fr/latest/tutorials/physics/) (http://docs.godotengine.org/fr/latest/getting_started/workflow/assets/escn_exporter/physics.html)
- Know make a Tile Map: (https://docs.godotengine.org/en/3.0/tutorials/2d/using_tilemaps.html)
- Understand how Nodes works: (https://docs.godotengine.org/en/3.0/classes/class_node.html)

Info on the Framework:

- My version of Godot Engine:



La communauté Godot vous dit merci !

Godot Engine v3.0.6.stable.official.8314054
© 2007-2018 Juan Linietsky, Ariel Manzur.
© 2014-2018 Contributeurs de Godot Engine

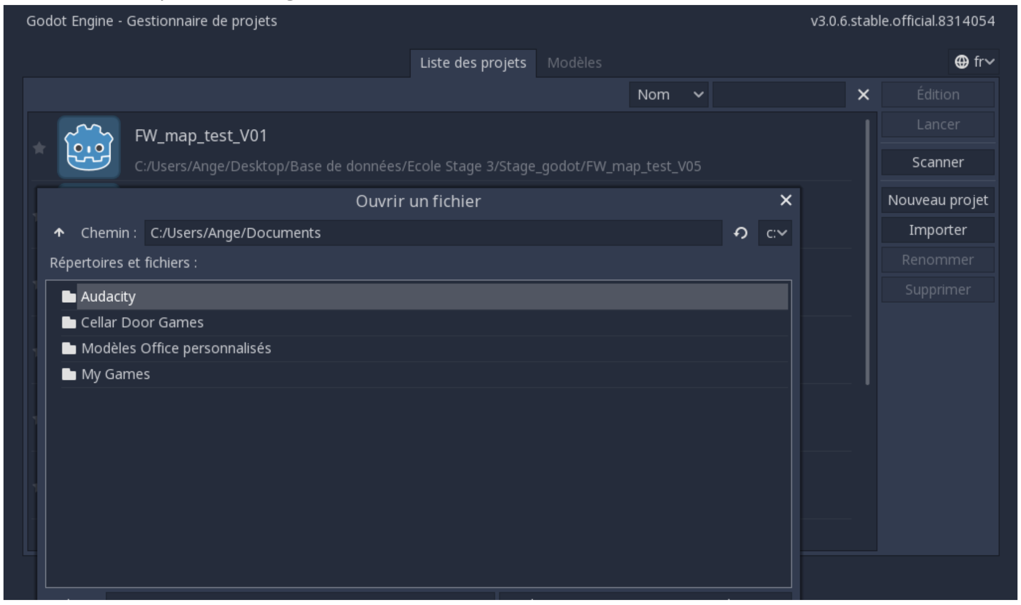
```
Copyright (c) 2007-2018 Juan Linietsky, Ariel Manzur.
Copyright (c) 2014-2018 Godot Engine contributors (cf. AUTHORS.md)

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

- How to import this Framework on your computer:
    - Download the project source files and place them in the files on your computer.
    - Open Godot Engine and in the project list menu, click the import button.
    - A new menu appears, enter the source file path of the Framework or click the brows button and open the ". godot" file in the source files.



Godot Engine - Gestionnaire de projets   v3.0.6.stable.official.8314054

FW_map_test_V01
C:/Users/Ange/Desktop/Base de données/Ecole Stage 3/Stage_godot/FW_map_test_V05

Ouvrir un fichier
Chemin : C:/Users/Ange/Documents
Répertoires et fichiers :
Audacity
Cellar Door Games
Modèles Office personnalisés
My Games

    - The Framework now appears in the list of projects, it is possible to open and modify it.

- Use the Framework: we can find the "scenes_projet_FW" directory in the source files of the Framework which contains all the scripts and the typical scenes used. It is possible to create your own game with my Framework by taking only this folder.

The Map test:

In the map test, you can find the game elements that I created. To make this map, I created empty Nodes that serve as a “folder” and allow me to organize the tree. At these nodes, I add the other scenes of the game elements as children.

Create our own game elements:

TO CREATE A PLAYER

The Palyer is the avatar that the player will be able to control in the game.

To created Player, you must recreate the attached tree in a new scene.

- RigidBody2D (adapt it physic to our game (Mass: 23 / Friction: 1 / Gravity scale: 1 / Contact Reported: 0)) (attach to it the Player script)
    - Sprite (We define an image)
    - CollisionShape2D (We define a shape)
    - Camera2D (Check current for activation)

For a good creation, I advise you to:

- Set your main Node, The Player (the following parameters are not a standard!):
    - Node Type: RigidBody2D
    - Mode: Character
    - Mass: 3
    - Weight: 210
    - Friction: 0
    - Bounce: 0
    - Gravity Scale: 1
    - Custom Integrator: True
    - Continuous Cd: Disabled
    - Contacts Reported: 3
    - Contact Monitor: False
    - Sleeping: False
    - Can Sleep: True
- Created the following Nodes as the child of the main Node:
    - Sprite: Set an image to your sprite

    - Shape2D: define a shape
    - Camera2D: You can set the following setting, which aren't a standard:
        - Offset: (0,0)
        - Anchor Mode: Drag Center
        - Rotating: False
        - Current: True
        - Zoom: (1,1)
        - Drag Margin H Enabled: True
        - Drag Margin V Enabled: True

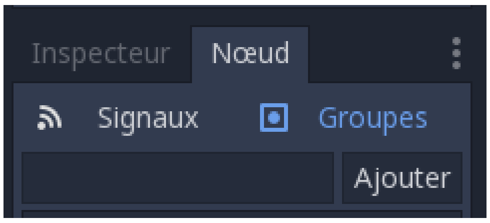
We import the scene into the level, then fill in the variable export:

- We begin by indicating the name of the character's commands, found in the « Project » / « Parameter of the project» / on the tab « Control ». We an create our own controls or use the default ones.
- Then, we indicate the parameters of moving:
    - Player main Sprite: We define the main Sprite that will turn around if the entire player Node turns around.
    - Walk_acceleration: The speed of the avatar acceleration.
    - Walk_deaccelleration: The speed of the avatar deceleration.
    - Walk_max_speed: Maximum accumulation of avatar acceleration.
    - Air_accel: Power of the acceleration of movement in the air (not related to physics but left and right movement).
    - Air_deaccel: Power of deceleration of movements in the air (not related to physics but left and right movement).
    - Jump_velocity: Power of the jump.
    - Falling_force: Power of the fall.
- Finally, we specifu the scene we will restart in the event of a collision with a Node of the "enemy" group, often the scene where we are. We will be able to modify this variable once the Player will be imported in the main scene.

TO CREATE AN ENEMY

An enemy is an element of the game that will have as main function to kill the Player. He will be able to move and raise the level

**/ ! **: *To create an enemy otherwise, it is also possible to integrate an element in the group "ennemi". Find the group tab in the Node menu on the same interface as the inspector menu, then in the group tab.*



If the player hits an element of the "ennemi" group, the scene is restarted.

To create an Enemy, you need to create the following tree in a new scene:

- RigidBody2D (adapt it physic to our game (Mass: 1 / Weight: 70 / Friction: 0 / Gravity scale: 1 / Contact Reported: 4)) (we attach the enemy script to it)
    - Sprite (We define an image)
    - CollisionShape2D (We define a shape)
    - RayCast2D (check enabled to activate) left side
        - The RayCast2D can be used to check if the Node is about to fall into the void, so you need them on each side of the Node. If the void is detected, the Node makes half turns.
    - RayCast2D (check enabled to activate) right side

We import the scene into the level, then fill in the variable exports:

- Stay_on_plateform: check for the enemy to detect the void and the walls with RayCast to turn around.
- RC_(left / right)_path: Specify both RayCast that the Node must take into account.
- Ennemi_main_Sprite: We define the main Sprite that will turn around if the enemy turns around.
- Walk_speed: The speed of enemy movement
- Direction: indicate the direction to take: 1 = right, -1 =left, 0 = null (stay on place).

TO CREATE A MOVING PLATFORM

Any physic element of the game can move on a moving platform surface.

**/ ! **: *It is possible to create a stationary platform from a moving platform (a wall or a floor). To do this, simply enter 0 in the Motion speed and rotation speed variable export parameter.*

To create a moving platform, you need to recreate the attached tree in a new scene:

- Node2D (We attach the moving_platform script to it)
    - RigidBody2D (Adapt it physic to our game (I only change the mode: Kinematic / Static))
        - Sprite (We define an image)
        - CollisionShape2D (We define a shape)

We import the scene into the level, then fill in the variable exports:

- Moving_platform_main_body: We define the Node that will serve as a platform
- Motion: We indicate the round trip that the moving platform will go through (if 0 is set to 10, It will go to a distance of 10 to the left, then 20 to the right, then 20 to right, etc.)
- Motion_speed: We specify the speed of movement, the higher the value is, the lower the speed will be.
- Rotation_speed: The speed of the rotation.
- Rotation_side: the direction of rotation (The Clock wise).

TO CREATE A TRIGGER

A trigger is a detection zone that will launch functions according to parameters if certain conditions are reached.

To create a Trigger, you need to recreate the attached tree in a new scene:

- Area2D (We attach the Trigger_full_V01 script to it)
    - CollisionShape2D (We define a shape)

We import the scene into the level, then fill in the variable exports:

- if_node: Specify the node to detect in the trigger (often the player)

Her's how to fill in the action's events:

- do_: Tick to activate all the events of this action
- so_die_: queue_free () The list of given Nodes
- so_hide_: hide () The list of given Nodes
- so_show_: show () The list of given Nodes
- so_instance_: Define a node that will be instantiate in the node list given in so_in_instance_
- so_in_instance_: Indicate the list of Nodes which so_instance will instantiate in
- so_loadScene_: indicate the new scene to load

- so_playSound_: indicate the list of sound to play (Sound must be Nodes)
- so_killOwn_: Tick if the Trigger must be queue_free after activated

The exported variables are all repeated 3 times with the only difference being their suffix after the _ which indication the action. The 3 different actions are:

- 1: enter (if_node enter trigger),
- 2: extire (if_node get out trigger),
- 3: button (The player presses a control key).
  - We find for example the variable so_die_enter

For variable export buttons:

- only_on_trigger_shape: Indicate here if the action of a control must be done in the CollisinoShape of the trigger.
- do_press_controll: indicate the name of the desired action. Find the controls in the Project menu, Project Settings and in controls there will be a list of actions.