🏠 Apprendre le GD script

Apprendre le GD script

Ce document permet d’apprendre le GD script, language de programmation de godot

Info:

func _ready() -> void:----------execute le code au lancement
	{action}

func _process(delta: float) - > void:----------execute le code tout au long de partie
	{action}

Inputmap = paramètre du projet/contrôle

{quelque chose}----------choisissez le nom
⚠️{quelque chose}----------dans le script, les {} doivent être écrit

{code ou titre}(----------){explication}----------donne une explication à un code ou à un titre

😀 Les bases du GD script:

Les bases du GD script:

base:

Extends {noeud} —---------- Pour commencer un code

Pass —----------passer l'action
print({texte})----------écrire du texte dans la console
print({texte}, {variable})--------écrire un texte avec une variable
{texte} —----------écrire un commentaire

Variable:

var {nom} = {"texte" ou nombre ou true/false}----------faire des variables

Type de variable		
Type de variable	Exemple	Description
Nombre entier(int)	var score = 150	Stocke un nombre sans virgule (ex: score, points de vie)
Nombre à virgule(float)	var vitesse = 2.5	Stocke un nombre avec une virgule (ex: vitesse, gravité)
Texte(String)	var pseudo = "Player1"	Stocke du texte (ex: nom du joueur, message)
Vrai ou Faux(bool)	var game_over = false	Stocke une valeur vrai (true) ou fausse (false).
Liste (Array)	var inventaire = ["Épée", "Potion"]	Stocke plusieurs valeurs dans un même ensemble (ex:objets d'un inventaire).
Dictionnaire(Dictionary)	Var joueur = {"nom": "Link", "vie": 100}	Associe des valeurs avec les noms (ex: caractéristiques d'un personnage).

Calcule:

{variable ou nombre} = {variable ou nombre} {- ou + ou * ou /} {variable ou nombre} —--------faire des calculs
{variable ou nombre} {- ou + ou * ou /}= {variable ou nombre} —--------faire des calculs

Conditions:

If {variable} {type de condition} {variable ou nombre}:
    {action}

esle:
	{action}
Elif {variable} {type de condition} {variable ou nombre}:
	{action}

Type de condition		
Signe	sinification	exemple
==	Est égal à	if score ==100:
!=	Est différent de	if pseudo != “Joueur1”:
<	Plus petit que	if vitesse < 5:
>	Plus grand que	if niveau > 3:
<=	Plus petit ou égal	if vie <= 0:
>=	Plus grand ou égal	if argent >= 1000:

Opérateurs:

and —---------- et
or —---------- ou
not —---------non
Les boucles
for in range({variable ou nombre}):----------boucler nombre fois
	{action}

while {condition}:----------faire la boucle jusqu'à ce que la condition est fausse
	{action}

while true: —----------boucle infini(à éviter)
	action
Break—----------sortir de la boucle

😏 Avancé du GD script:

Avancé du GD script:

Input:

Input.is_action_pressed(“{input dans inputmap}”)----------touche {input de inputmap} pressé
Input.is_action_just_pressed(“{input dans inputmap}”)----------touche {input de inputmap} pressé une fois

Fonction:

func {nom fonction}():----------créé une fonction
    {action}

{nom fonction}()----------appeler une fonction

—----------

func {nom fonction}(message):----créé une fonction qui dit ce qui a dans l’appelle
    print(message)

{nom fonction}({variable ou texte})--appeler une fonction et écrire dans la fonction

ex:

```
dire_bonjour(“salut”)
func dire_bonjour(message):
    print(message)
```

Resultat:

Console: salut

Retourner une valeur:

ex:

```
Var resultat = additionner(5, 3)
Func additionner(a, b):
    return a + b
print(resultat)
```

resultat:

Console: 8

Envoyer des messages:

Envoyer:

Signal {texte}----------créé le signal
{signal}()----------appeler la fonction qui envoie(mettre cette ligne dans la fonction _process mais n'est pas performance)
emit_signal("{signal}", "{texte}")----------envoyer le signal(le mettre dans la fonction d'envoie)

Recevoir:

${label qui envoie le message}.connect("{signal}", {nom fonction qui exécute l'action après avoir reçu le signal})---------recevoir le signal et envoyé une fonction qui fait une action

Sécuriser les variables avec getters & setters:----------permet d'empêcher la triche sur les variables

var {nom variable} {opérateur} {valeur} :
    set(value):
        {variable} = clamp(value, {valeur minimum de la variable}, {valeur maximum de la variable})
    get:
        return{variable}

Array:----------permet de stocker plusieur donné dans une variable

var {nom variable} = ["{élément 1}", "{élément 2}", {etc}]----------faire une variable array
({variable}[{nombre}])----------avoir l'information(nom) de l'item d'une variable array en fonction de son emplacement(0=premier emplacement, etc)
({variable})----------avoir l'information de toute la variable array

Dictionnaire:----------permet de stocker des informations sur plusieurs objets

var {nom variable} = {"{élément 1}": "{information}", "{élément 2}": {information}, {etc}}----------faire une variable dictionnaire
({variable}[{élément}])----------avoir l'information(toute les infos donné dans la variable) de l'élément d'une variable dictionnaire en fonction de son nom
({variable})----------avoir l'information de toute la variable dictionnaire

{variable}["{nom d'élément}"] {opérateur} {nouvelle valeur}----------modifier un élément du dictionnaire
{variable}["{nom du nouvel élément}"] {opérateur} {valeur}----------ajouter un élément du dictionnaire

for cle in {variable}:----------pour toute les objets du dictionnaire faire {action}
    {action}
print(cle, " : ", {variable}[cle])----------écrire tous les éléments du dictionnaire en ajoutant : entre l'information et l'objet(à mettre dans la condition for cle in {variable:})

Enum:----------permet de ne pas se perdre en faisant des conditions

Ex:

var action = “AttaquePuissante”----------je fais une variable contenant du texte
if action == “AttaquerPuissante”:----------je vérifie si c’est bien AttaquePuissante mais j’ai fait une faute en ajoutant un r à attaque

---

var action = 2—----------je fais une variable contenant du chiffre
if action == 2:----------je vérifie si c’est bien 2 et je ne fait pas d’erreur mais ducoup je sais pas à quoi correspond 2 sans le noter quand j’ai plein de chiffre. Voila à quoi sert l’Enum

Pratique:

```
enum {nom de l'enum} {----------faire un enum
        {Élément 1},
        {Élément 2},
        {etc}
}

var {nom variable} = {enum}.{élément de l'enum}
if {variable} = {enum de la variable}.{élément de l'enum de la variable}
        {action}
```

Match:----------permet d’éviter les “if” répétitif

```
{faire un enum}
Func {nom fonction}(action):----------faire une fonction match
        Match action:
                {enum}.{élément de lenum}:
                        {action}
                {enum}.{élément de lenum}:
                        {action}
                {etc}
```

Appeler le match:

{fonction}({enum}.{élément de l’enum à appeler})----------appeler la fonction match

😎 Manipuler les node GD script:

Manipuler les node et les scene GD script:

Se connecter au node et au scene:

var {texte} = ${chemin du node(ex:$node/monmesh)}--------faire une var contenant un node(peut faire glisser déposer le node)

Var {texte}----------faire une var contenant une scène

Modifier un node:

{node}.text = “{texte}”----------modifier le texte que contient un node
{node}.add_theme_color_override(“font_color”, Color.{couleur})----------modifier la couleur du text que contient un node

Créé un node:

var {texte} = {type de node}.new()
add_child({variable})

Faire une classe en script:(créé un script qui contient cette classe)------(ex: au lieu de faire plusieurs script pour chaque ennemi du jeu pareil, faire une classe qui fait apparaître l’ennemi avec tout le code)

class_name {nom de la classe}----------créé une classe

{code de la classe(fonction, variable, etc)}----------code de la classe

Appeler une classe en script:

var {nom variable} = {nom classe}.new()

Faire un héritage en script:----------donner les capacités d’une classe à une autre

extends {classe qui a les caractéristiques à copier}
{créer une classe qui va prendre les caractéristiques}

Changer de scene:

get_tree().change_scene_to_file({variable ou “scene”})