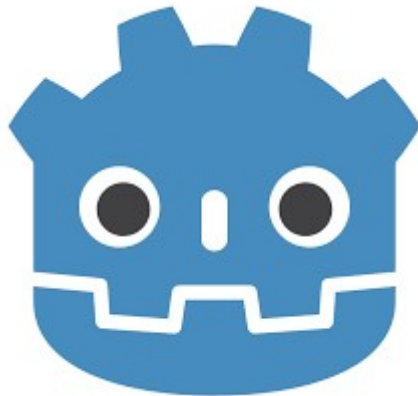


# RAPPORT DU PROJET

## GAME 2D



Préparé Par :  
Fennassi Oussama

Encadrant :  
Pr. EL AACHAK LOTFI  
Pr. Ben Abdel Ouahab Ikram

## PLAN DU TRAVAIL:

- Présentation du Engine Godot:
- Comment utiliser Godot:
- Création d'une bibliothèque dynamique (DLL) :
- Création du Projet Godot :
- Utiliser le nœud "TileMap" dans Godot :
- Créer un Personnage :
- Partie Code (Player) :
- Partie Code (Monster) :
- Partie Code (Thrones) :
- Principale (Main Menu) :
- Principale (Main Menu) :
- Levels :
- Conclusion générale :

## 1. Présentation du Engine GODOT :



- ➔ Godot est un moteur de jeu open source qui permet de développer des jeux 2D et 3D. Il est écrit en C++ et est utilisé pour créer des jeux pour de nombreuses plates-formes, notamment Windows, MacOS, Linux, iOS et Android.
- ➔ Godot offre un environnement de développement intégré (IDE) facile à utiliser avec une interface graphique intuitive pour la création de scènes et l'édition de code. Le moteur est également doté d'un langage de script personnalisé appelé GDScript, qui est basé sur Python et facile à apprendre.

## 2. Comment utiliser GODOT :

- ➔ Pour utiliser le langage C++ sur GODOT il faut utiliser une extension appelle GDNative :
- ➔ GDNative C++ est une extension de Godot qui permet d'étendre le moteur de jeu en utilisant du code C++ natif. Cela peut être utile lorsque vous avez besoin de performance supplémentaire ou si vous avez besoin d'accéder à des fonctionnalités spécifiques qui ne sont pas disponibles dans le langage de script GDScript de Godot.
- ➔ Pour utiliser GDNative C++, vous devez créer une bibliothèque dynamique qui contient votre code C++ et la charger dans Godot à l'exécution. Vous pouvez alors appeler des fonctions de votre bibliothèque depuis GDScript ou depuis un script C# dans le cas où vous utilisez l'extension GDNative C# de Godot.

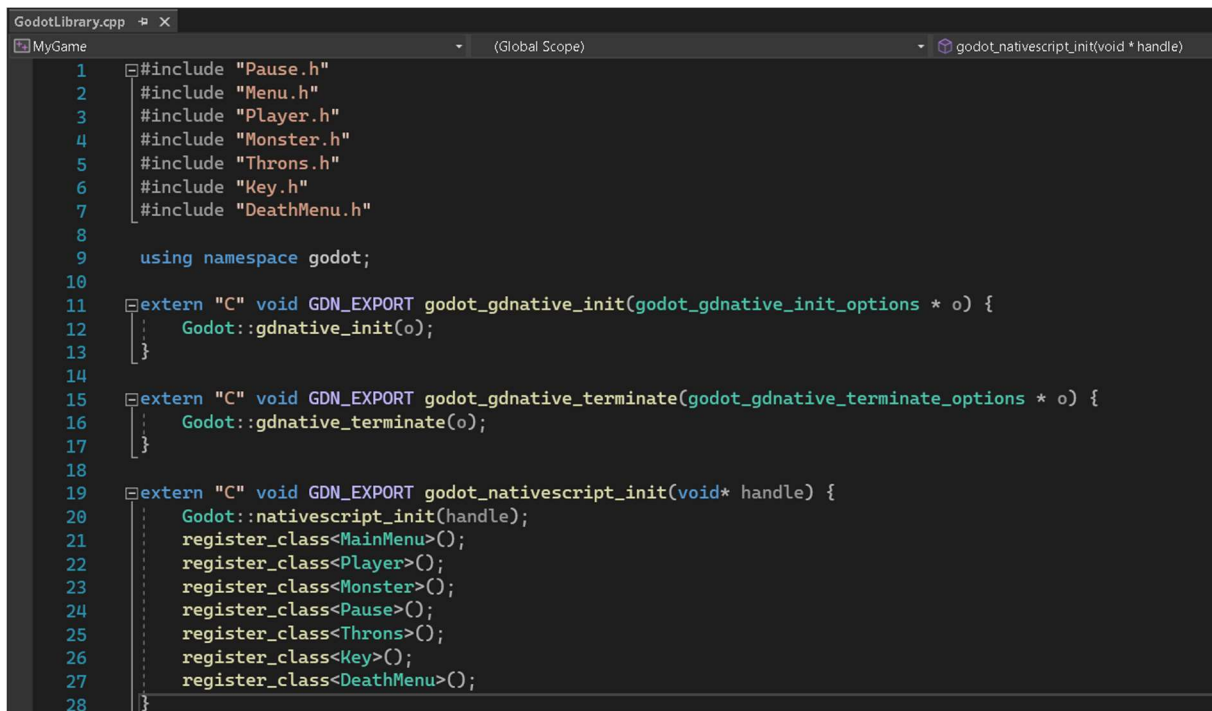
### 3. Création d'une bibliothèque dynamique (DLL) :

1/ Ouvrez Visual Studio et créez un nouveau projet en sélectionnant "Bibliothèque dynamique de bibliothèques de liens dynamiques (DLL)" dans le menu "Nouveau projet".

2/ Sélectionnez le répertoire de destination et entrez un nom pour votre projet. Cliquez sur "Créer" pour créer le projet.

3/ Dans l'Explorateur de solutions, ouvrez le fichier .cpp qui se trouve dans le répertoire "Source files". C'est là que vous allez écrire votre code C++.

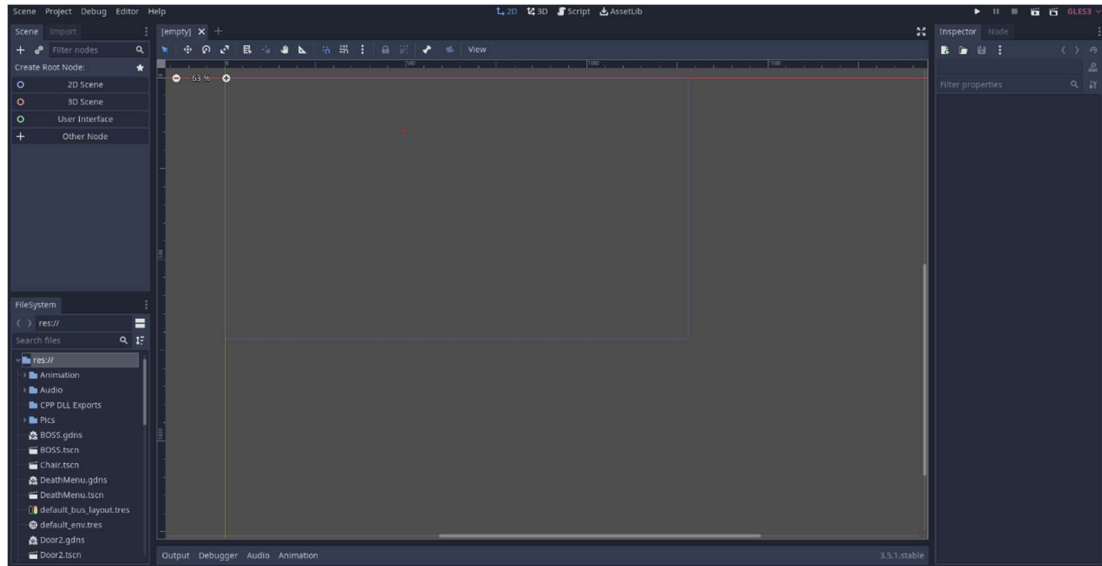
4/ Écrivez votre code C++ et utilisez les directives extern "C" pour déclarer les fonctions que vous voulez exporter depuis votre DLL :



```
GodotLibrary.cpp
MyGame (Global Scope) godot_nativescript_init(void * handle)
1  #include "Pause.h"
2  #include "Menu.h"
3  #include "Player.h"
4  #include "Monster.h"
5  #include "Throns.h"
6  #include "Key.h"
7  #include "DeathMenu.h"
8
9  using namespace godot;
10
11 extern "C" void GDN_EXPORT godot_gdnative_init(godot_gdnative_init_options * o) {
12     Godot::gdnative_init(o);
13 }
14
15 extern "C" void GDN_EXPORT godot_gdnative_terminate(godot_gdnative_terminate_options * o) {
16     Godot::gdnative_terminate(o);
17 }
18
19 extern "C" void GDN_EXPORT godot_nativescript_init(void* handle) {
20     Godot::nativescript_init(handle);
21     register_class<MainMenu>();
22     register_class<Player>();
23     register_class<Monster>();
24     register_class<Pause>();
25     register_class<Throns>();
26     register_class<Key>();
27     register_class<DeathMenu>();
28 }
```

## 4. Création du Projet GODOT :

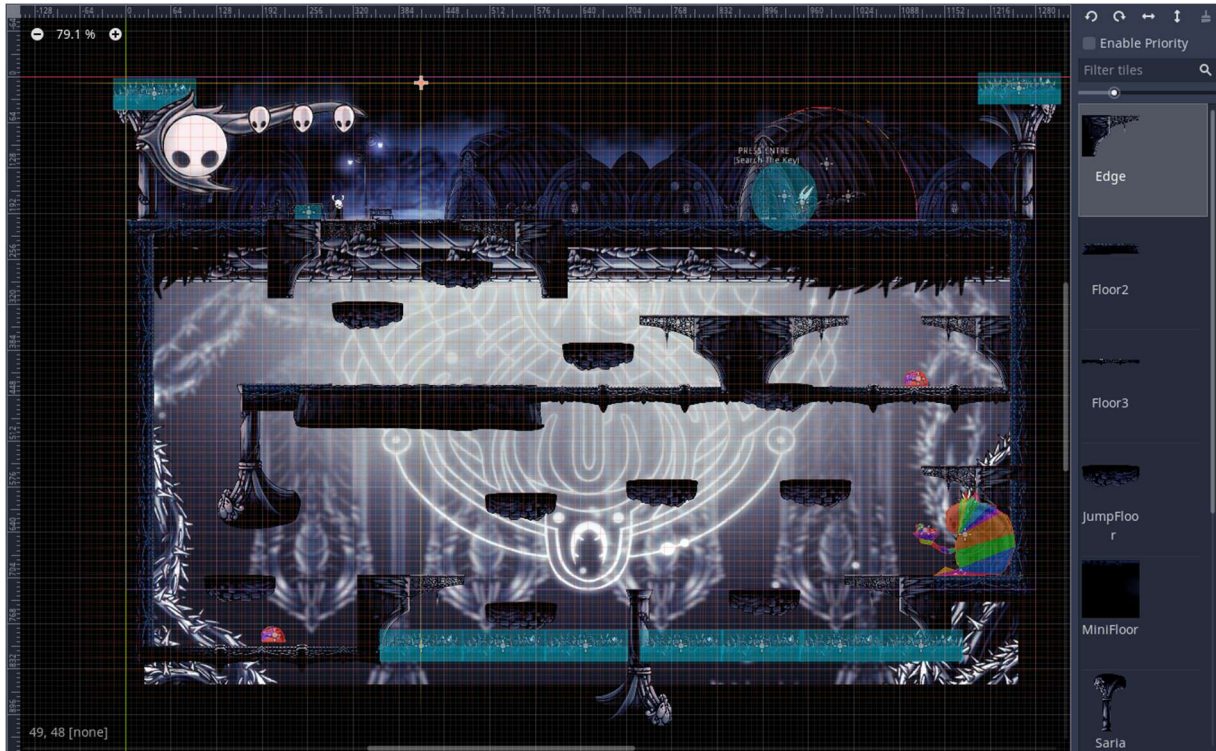
- ➔ Lancez Godot et créez un nouveau projet en sélectionnant "Jeux 2D" dans le menu "Nouveau projet".



- ➔ Dans l'interface de l'éditeur de scènes, créez votre scène de jeu en ajoutant des nœuds et en définissant leurs propriétés. Vous pouvez utiliser des nœuds tels que "Sprite" pour afficher des images et "CollisionShape2D" pour ajouter des colliders à votre scène.
- ➔ Écrivez du code en utilisant GDNative pour implémenter la logique de votre jeu. Vous pouvez utiliser les signaux et les événements d'entrée pour déclencher des actions en réponse à des interactions de l'utilisateur.
- ➔ Ajoutez de l'audio et des effets visuels pour améliorer l'atmosphère de votre jeu. Vous pouvez utiliser les nœuds "AudioStreamPlayer" et "Particles2D" pour cela.
- ➔ Testez votre jeu en utilisant l'option "Exécuter" dans le menu "Exécuter". Assurez-vous de vérifier que tout fonctionne correctement et de corriger tout bug que vous rencontrez.
- ➔ Une fois que votre jeu est prêt, vous pouvez le publier pour que d'autres puissent y jouer. Godot prend en charge la publication pour de nombreuses plates-formes, notamment Windows, MacOS, Linux, iOS et Android.

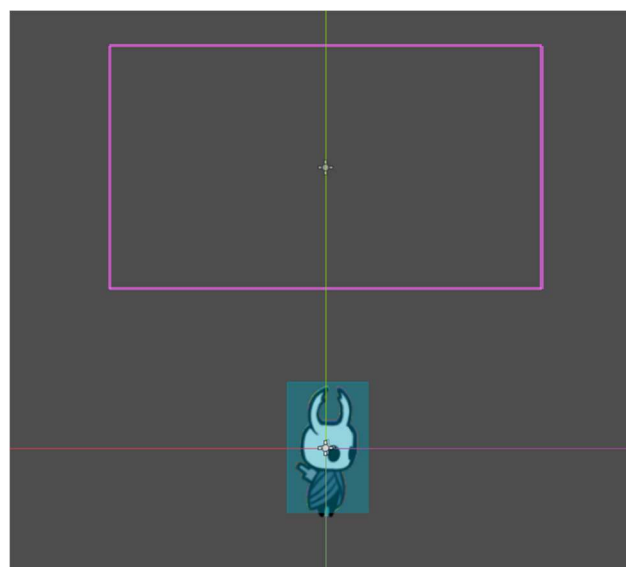
## 5. Utiliser le nœud "TileMap" dans GODOT :

- ➔ Le "TileMap" est un nœud de Godot qui permet de créer des cartes de tuiles (ou "maps") à partir de tuiles prédéfinies. Cela peut être utile pour créer des niveaux de jeu ou des décors répétitifs de manière rapide et efficace.

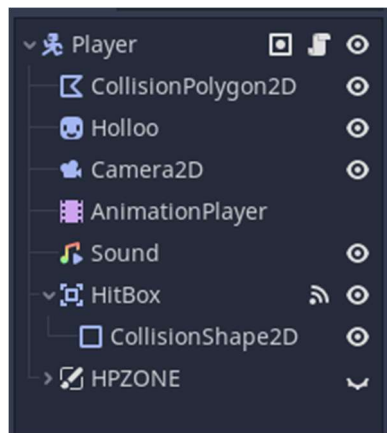


## 6. Créer un Personnage :

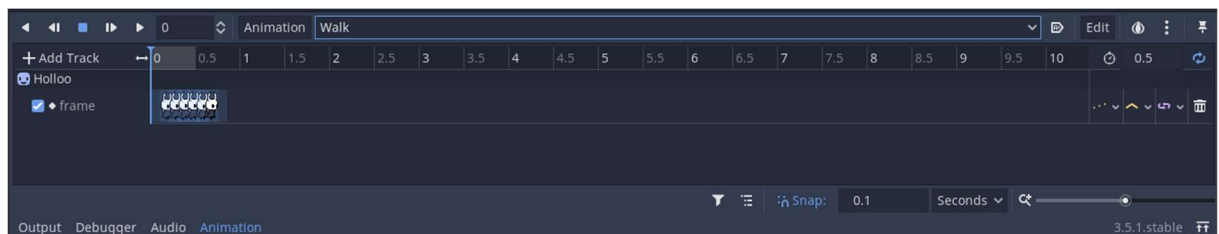
1/ Dans l'interface de l'éditeur de scènes, créez votre scène de jeu en ajoutant des nœuds et en définissant leurs propriétés. Vous pouvez utiliser un nœud "Sprite" pour afficher l'image de votre personnage et un nœud "CollisionShape2D" pour ajouter des colliders à votre personnage.



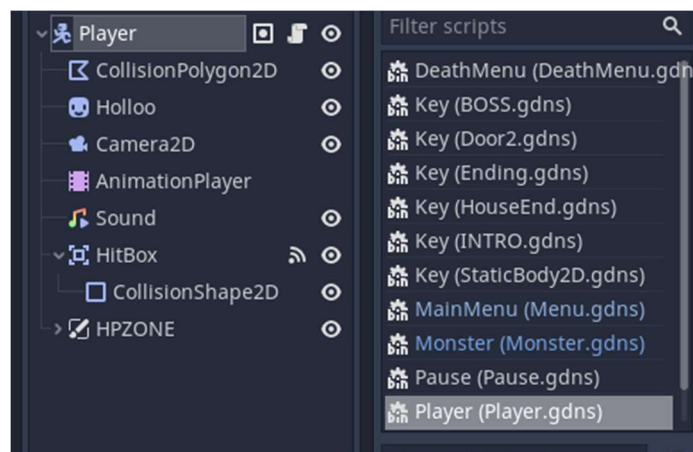
2/ Écrivez du code en utilisant GDScript ou un autre langage de script pris en charge par Godot pour implémenter la logique de votre personnage. Vous pouvez utiliser les signaux et les événements d'entrée pour déclencher des actions en réponse à des interactions de l'utilisateur.



3/ Ajoutez de l'animation à votre personnage en utilisant un nœud "AnimationPlayer" et en créant des animations pour chaque état de votre personnage (par exemple, marcher, sauter, etc.). Vous pouvez utiliser la propriété "Animation" du nœud "Sprite" pour lier une animation à l'image de votre personnage.



4/ Une fois que votre personnage est prêt, vous pouvez l'intégrer à votre jeu en le plaçant dans votre scène de jeu et en utilisant le code que vous avez écrit pour le faire interagir avec le reste de votre jeu.





## 7. Partie Code (Player) :

- ➔ Dans la partie header du Player nous avons créé une classe Player dont laquelle nous allons définir tous les prototypes des méthodes de notre caractère.

```
class Player : public KinematicBody2D
{
    // Godot structure
private:
    GODOT_CLASS(Player, KinematicBody2D)
public:
    Player();
    static void _register_methods();
    void _init();
    void _process(float delta);
    void _physics_process(float delta);
    void _on_shy_body_entered(Node* body);
    void _on_shy_body_exited(Node* body);
    void _on_spawn1_body_exited(Node* body);
    void _on_hitbox_body_entered(Node* body);
    void _on_spawn1_body_entered(Node* body);
    void _on_spawn2_body_entered(Node* body);
    void _on_spawn3_body_entered(Node* body);
    void _ready();
    ~Player();

    // Gameplay variables
public:
    const int speed = 100; //Character Speed
    const int gravity = 160; //World Gravity
    const int Maxfallspeed = 180; //Character Max fall Speed
    const int MaxSpeed = 300; //Character Max Speed
    const int jumpForce = 3300; //Character Jump
    const int JumpMax = 2; //Character Max jumps nombre
    const int Max_Wall_slide = 40; //Character Max wall slide speed
    const int Wall_slide_Speed = 100; //Character wall slide speed
    int JumpCount = 0; //Jumps Counter
    bool Jump_was_pressed = false; //Jump condition
    bool Can_jump = false; //Jump condition
    bool facingRight = true; //Character facing condition
    bool is_paused = false; //Paused scene condition
    bool visible = false; //visibility condition
    bool shy = false; //Character shy condition
    bool sit = false; //Character sit condition
    int HP = 3; //Character Max Hp
private:
    Vector2 motion; //vector
    Vector2 respawn_position; //Character respawn position
};
```

- ➔ Et au niveau de la fichier Player.cpp on définit toutes les méthodes :

**void Player::\_register\_methods(){}**

- ➔ Cette fonction est utilisée pour enregistrer les méthodes d'une classe de script GDScript auprès du moteur de jeu. Ces méthodes peuvent être appelées depuis d'autres scripts GDScript ou C#, ou depuis le code C++ de votre jeu.

**void Player::\_init() {}**

- ➔ Cette fonction est appelée automatiquement par le moteur de jeu lorsqu'un nœud est ajouté à la scène ou lorsqu'une instance de classe de script est créée.

**Player::Player() {}**

- ➔ Le constructeur de la classe Player.

**void Player::\_ready() {}**

- ➔ La fonction ready () est utilisée dans Godot pour initialiser un nœud lorsqu'il est prêt à être utilisé.



`Player::~~Player() {}`

- Destructeur de la classe Player.

`void Player::_process(float delta) {}`

- La fonction process () est utilisée dans Godot pour mettre à jour un nœud ou une classe de script à chaque frame du jeu.

`void Player::_physics_process(float delta){}`

- La fonction physics\_process () est utilisée dans Godot pour mettre à jour un nœud ou une classe de script impliqué dans les calculs de physique à chaque frame du jeu

les methodes du signal.

- Les signaux sont utilisés pour envoyer des événements entre différents nœuds et scripts du jeu. Ils peuvent être utilisés pour déclencher des actions en réponse à des événements tels que des collisions, des entrées utilisateur, etc.

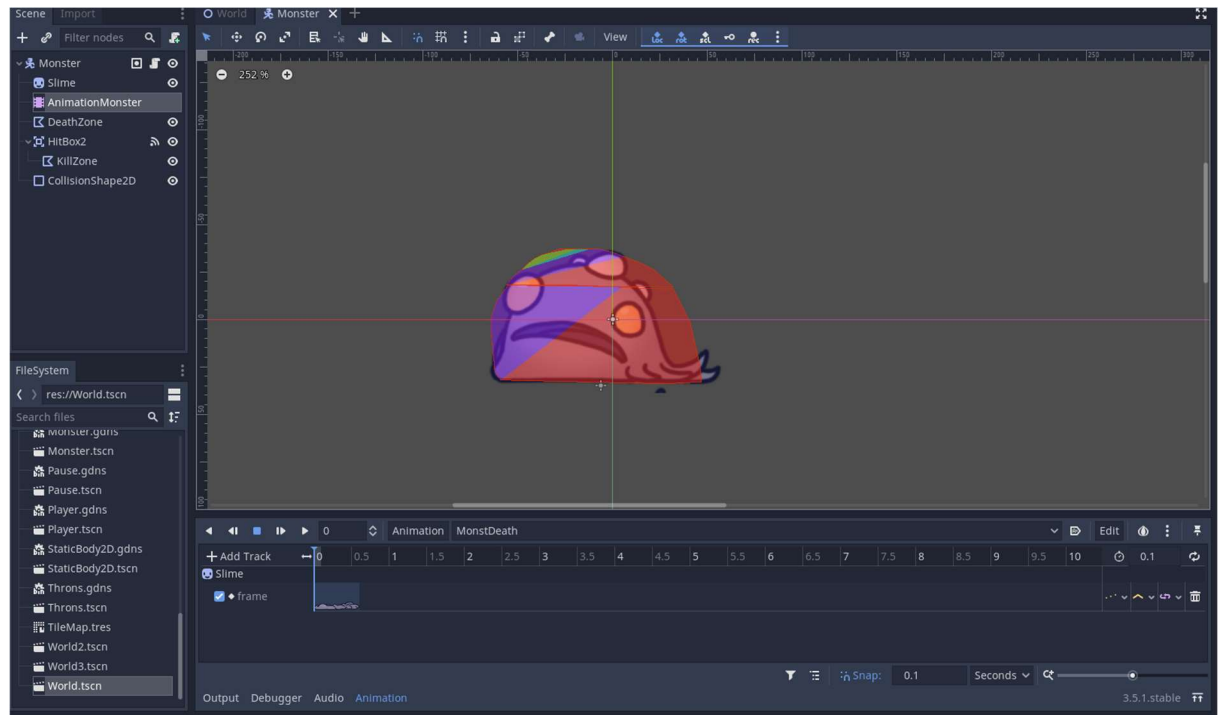
```
//Walk Left-----
if (i->is_action_pressed("ui_left")) {
    motion.x -= speed;
    facingRight = false;
    if (is_on_floor()) {
        if (i->is_action_pressed("run")) {
            _Anim->play("Run");
            motion.x -= 80;
            if (i->is_action_just_pressed("jump") && is_on_floor()) {
                motion.x -= 500;
            }
        }
        else if(i->is_action_pressed("Alt"))
            _Anim->play("worm_move");
        else
            _Anim->play("Walk");
    }
}

//Walk right-----
else if (i->is_action_pressed("ui_right")) {
    motion.x += speed;
    facingRight = true;
    if (is_on_floor()) {
        if (i->is_action_pressed("run")) {
            _Anim->play("Run");
            motion.x += 80;
            if (i->is_action_just_pressed("jump") && is_on_floor()) {
                motion.x += 500;
            }
        }
        else if (i->is_action_pressed("Alt"))
            _Anim->play("worm_move");
        else {
            _Anim->play("Walk");
        }
    }
}
```

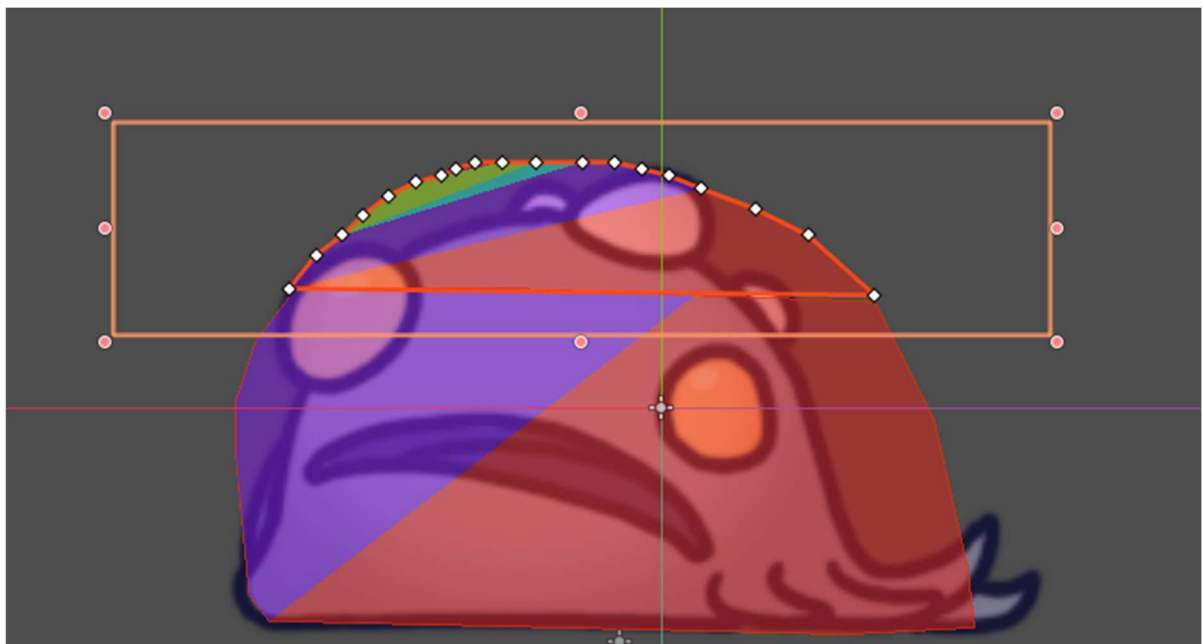
- Nous avons utilisé la bibliothèque <input> pour déclarer un pointeur < i > qui nous permet de lie le mouvement à une touche du clavier.
- On définit chaque touche au niveau des paramètres de Godot.

## 8. Partie Code (Monster) :

- ➔ Pour ajouter des adversaires on crée une classe Monster avec les mêmes méthodes que la classe Player, mais cette fois on définit un mouvement automatique au Monster.

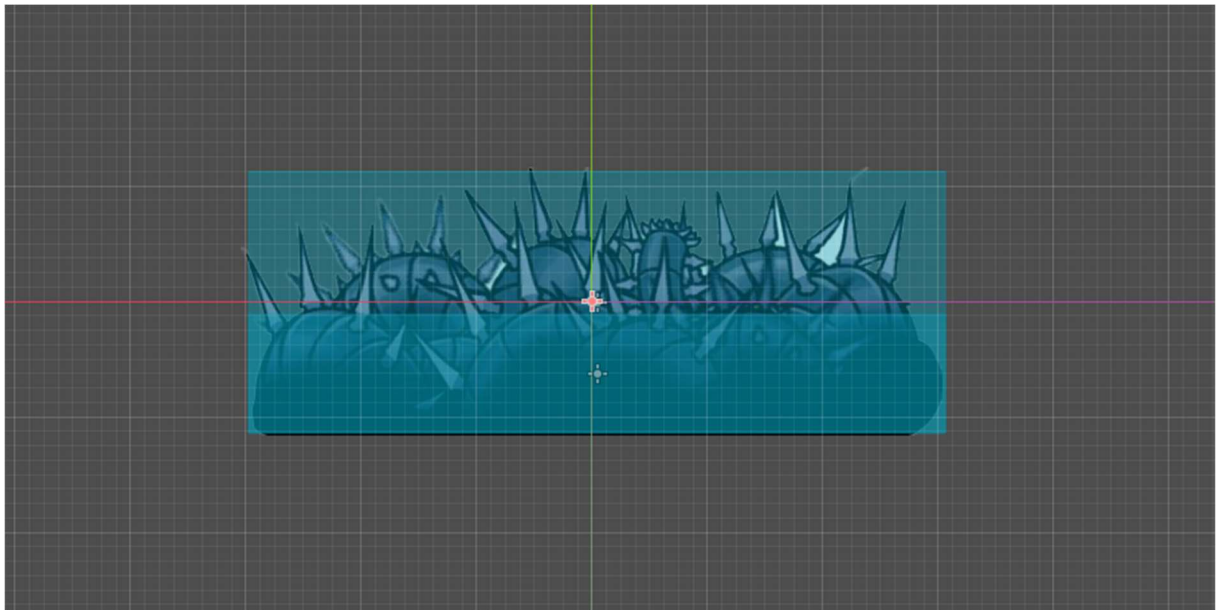


- ➔ Le nœud Area2D et La méthode `on_Area2D_body_entered ()` sont utilisées pour créer une zone qui peut être utilisée pour détecter les collisions, les entrées entre le Player et le Monster pour tuer l'un des deux.



## 9. Partie Code (Monster) :

- ➔ Pour rendre le jeu difficile, nous avons créé un petit obstacle statique appelé <Throne> qui tue le Personnage au cas du contact physique. (-1 HP)



- ➔ La fonction `_on_HitBox_body_entered()` sert à détecter les collisions entre les nœuds. On donne un groupe pour chaque nœud pour différencier entre la zone du joueur et celle de l'obstacle.

```
namespace godot {
class Throne :public KinematicBody2D
{
    // Godot structure
private:
    GODOT_CLASS(Throne, KinematicBody2D)
public:

    static void _register_methods() {
        register_method("_on_HitBox_body_entered", &Throne::_on_HitBox_body_entered);
    }

    void _ready() {}

    Throne() {}

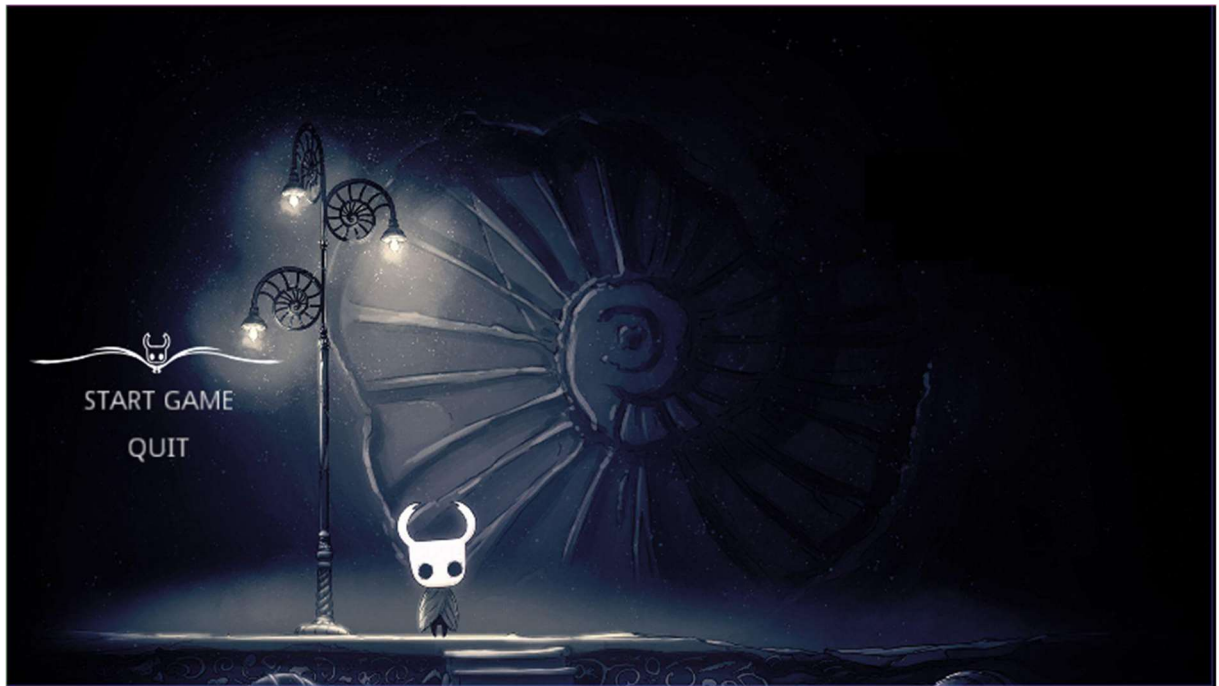
    ~Throne() {}

    void _init(){}

    void _on_HitBox_body_entered(Node* body) {
        if (body->is_in_group("Player"))
        {
            KinematicBody2D* Player = (KinematicBody2D*)body;
            //set_global_position(respawn_position);
            Player->set_global_position(get_global_position());
            //get_tree()->reload_current_scene();
        }
    }

public:
    Vector2 respawn_position;
};
}
```

## 10. Page Principale (Main Menu) :



- ➔ On ajoute un nœud Button au nœud Control et on le personnalise en utilisant les propriétés de l'éditeur. Ce bouton sera utilisé pour lancer le jeu lorsqu'il sera sélectionné.
- ➔ Puis On ajoute un nœud Button supplémentaire au nœud Control et on le personnalise également. Ce bouton sera utilisé pour quitter le jeu lorsqu'il sera sélectionné.
- ➔ Créez une fonction GDNative qui sera appelée lorsque le bouton de lancement du jeu est sélectionné. Cette fonction peut par exemple charger la scène de jeu principale ou afficher une fenêtre de chargement.

```
void _on_StartButton_pressed() {  
    get_tree()->change_scene("res://INTRO.tscn");  
}  
  
//void _on_OptionsButton_pressed() {  
//    // This function is called when the options button is pressed.  
//    // You can use it to open the options menu or display the options screen.  
//}  
  
void _on_QuitButton_pressed() {  
    get_tree()->quit();  
}
```

## 11. Pause Menu :

- ➔ Pour avoir la possibilité de quitter le jeu, nous avons un menu de pause qui nous permet de suspendre le jeu, et le quitter en cliquant sur la Button Quit.



- ➔ On utilise la Button echap ou P pour afficher le menu de pause lorsque le jeu est en cours.

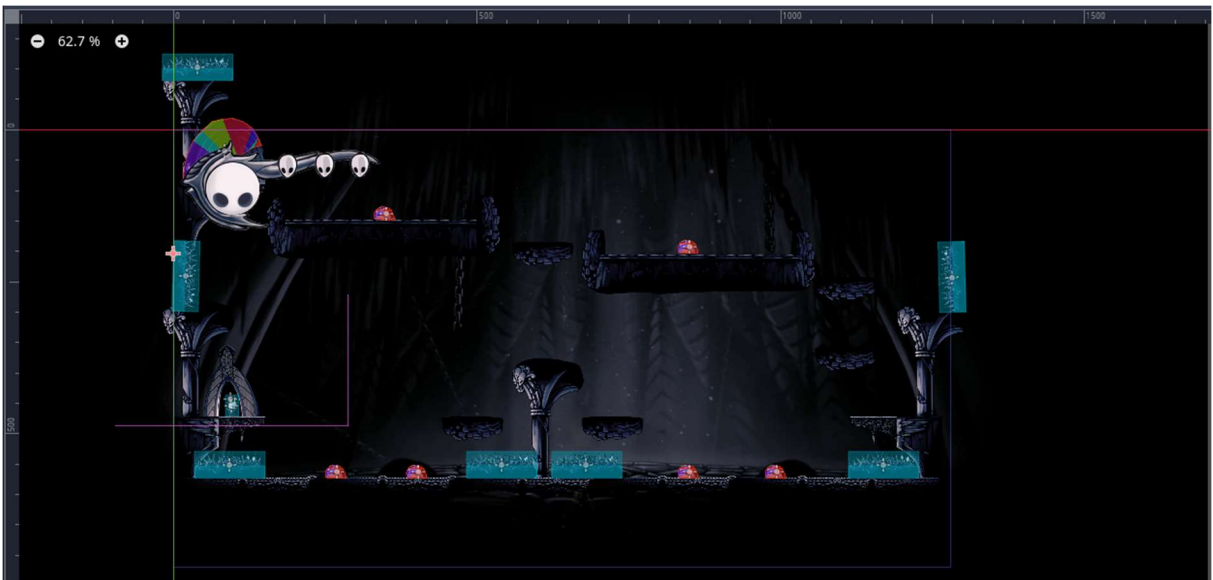
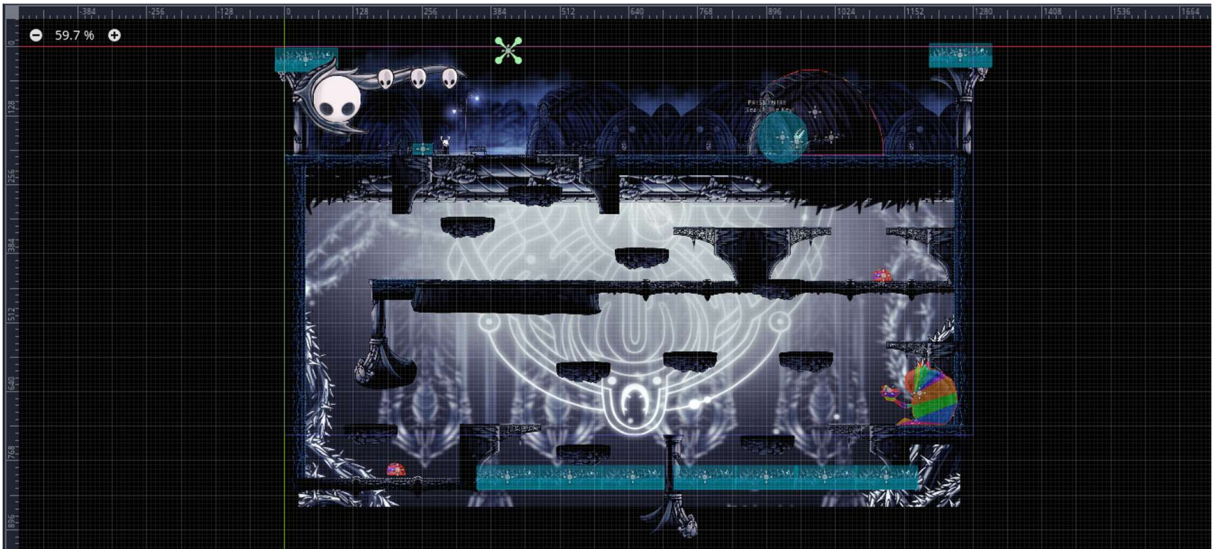
```
//Pause Menu-----  
CanvasLayer* PauseMenu = (CanvasLayer*)get_node("PauseMenu");  
Control* child = (Control*)PauseMenu->get_node("Pause");  
if (i->is_action_pressed("Pause")) {  
    get_tree()->set_pause(true);  
    child->set_visible(true);  
}
```

```
void _on_CONTINUE_pressed() {  
    get_tree()->set_pause(false);  
    set_visible(false);  
}  
  
void _on_QUIT_pressed() {  
    get_tree()->quit();  
}
```



## 12.Levels :

- ➔ Notre Game est composé de 3 Levels :
- ➔ Chaque Stage contient une TileMap différentes :



### 13. Conclusion Générale :

- ➔ Tout au long de la préparation de notre jeu, nous avons essayé de pratiquer les connaissances requises durant notre cours de programmation orienté objet, et aussi notre connaissance sur ce qu'on a trouvé concernant Godot. L'objectif c'est de concevoir et programmer un jeu, il nous a donné la possibilité de maîtriser et découvrir une nouvelle approche de la programmation.