

2 *Sprite*, déplacement et *tilemap* 2D

2.1 Dessiner un *sprite*

On place simplement une image de *Mario* dans la scène comme le présente la Fig. ??.

Fig. 23 présente l'arborescence de la scène.

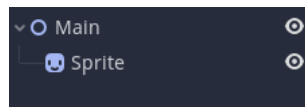


Fig. 23 – Arborescence.

Pour dessiner un *sprite* de *Mario*, suivre la procédure ci-dessous :

- Sélectionner l'élément *Node2D*, le renommer *Main* et l'enregistrer dans `Main.tscn`
[Dans le répertoire du projet (qui est nommé `res://` dans *Godot* et qui est nommé `le-nom-du-projet` dans le système de fichier), créer un répertoire *assets* et y placer une image de *Mario*]
- Ajouter un noeud *Sprite* et déplacer l'image de *Mario* du répertoire *assets* dans le champ *Texture*
[Déplacer le *sprite* (click-droit maintenu) dans la scène modifie le champ *Position* du *sprite*]
- Exécuter

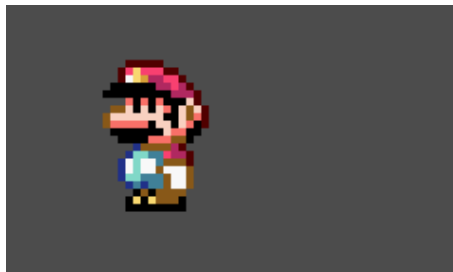


Fig. 24 – *Sprite* de *Mario* dans une scène 2D.

2.2 Dessiner des *sprite* animés

On place quatre *sprite* animés d'un *Block?* dans la scène.

Fig. 25 présente l'arborescence de la scène.



Fig. 25 – Arborescence.

Pour dessiner les quatre *sprite* animés de *Block?*, suivre la procédure ci-dessous :

- Sélectionner l'élément *Node2D*, le renommer *Main* et l'enregistrer dans *Main.tscn*
[Créer un répertoire *assets* et y placer les quatre images de *Block?*]
- Ajouter un nœud *AnimatedSprite* et créer un nouveau *SpriteFrame* dans le champ *Frames*
[Placer les images du répertoire *assets* dans l'animation par défaut du *SpriteFrame* comme le présente la Fig. 27]
- Appuyer sur **Ctrl**+**d** permet de dupliquer le nœud *AnimatedSprite* et créer automatiquement un nœud *AnimatedSprite2*
[Cette duplication peut s'appliquer à tous les nœuds de la scène²]
[Le nœud dupliqué est visuellement superposé avec le nœud original]
[Obtenir quatre *AnimatedSprite*]
- Déplacer les quatre *AnimatedSprite* pour les aligner dans l'interface
[Pour synchroniser les animations, sélectionner tous les nœuds dans la scène, désactiver *Playing*, fixer *Frame* à 0, activer *Playing*]
- Exécuter (et avoir une animation synchronisée comme présenté en Fig. 26)



Fig. 26 – Quatre *sprite* du *Block?* dans une scène 2D.

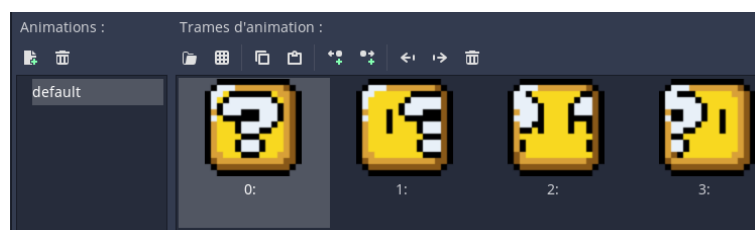


Fig. 27 – Animation par défaut du *SpriteFrame* du *Block?*.

2. A l'exception du nœud racine, qui ne peut en aucun cas être dupliqué.

2.3 Déplacer un *sprite* au clavier

L'interface contient un *sprite* de *Mario*; appuyer sur  et sur  permet de déplacer et orienter le *sprite* vers la gauche et vers la droite.

Fig. 28 et Fig. 29 présentent l'arborescence, les fonctions et les variables utilisées.

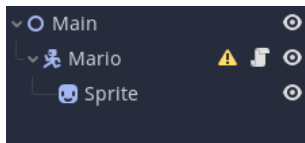


Fig. 28 – Arborescence.

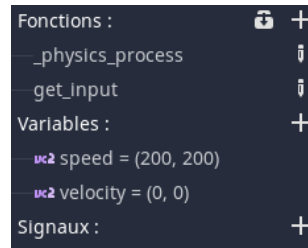







Fig. 29 – Fonctions et variables.

On utilise une seule image présentée en Fig. 30.



Fig. 30 – *Mario*.

Pour réaliser cette interface, suivre la procédure ci-dessous :

- Sélectionner l'élément *Node2D*, le renommer *Main* et l'enregistrer dans *Main.tscn*
[Créer un répertoire *assets* et y placer l'image de *Mario*]
- Ajouter un noeud *Mario* de type *KinematicBody2D*
[ indique l'absence de forme utile pour la détection de collision; comme ici nous ne l'utilisons pas, nous pouvons négliger ce *warning*]
- Ajouter un noeud *Sprite* avec l'image de *Mario*
[Ce noeud *Sprite* a le noeud *Mario* pour noeud parent]
[Avec l'inspecteur, fixer *x* et *y* du champ *Position* de *Mario* à 300 et 200]
- Ajouter dans les paramètres du projet les actions *Left*, *Right* et les associer à  et 
- Ajouter une variable *speed* de type *Vector2* initialisée à (200,200)
- Ajouter une variable *velocity* de type *Vector2* initialisée à (0,0)
- Attacher un *visual script* *Mario.vs* au noeud *Mario*
- Ajouter les fonctions *get_input* et *_physics_process*
- Réaliser les graphes correspondants
 - à la fonction *get_input* comme le présente la Fig. 31
 - à la fonction *_physics_process* comme le présente la Fig. 32
- [Sélectionner un élément du *visual script* puis appuyer sur  +  permet de dupliquer les éléments sélectionnés]
- Exécuter

La fonction *Set flip_h* est associée au noeud *Sprite*; elle permet d'orienter le *sprite* dans le sens du déplacement; la fonction *Action* permet de récupérer les touches clavier pressées; l'opération $A \times B$ sur le type *Vector2* (abrégé *vc2* en violet) est une multiplication terme à terme.

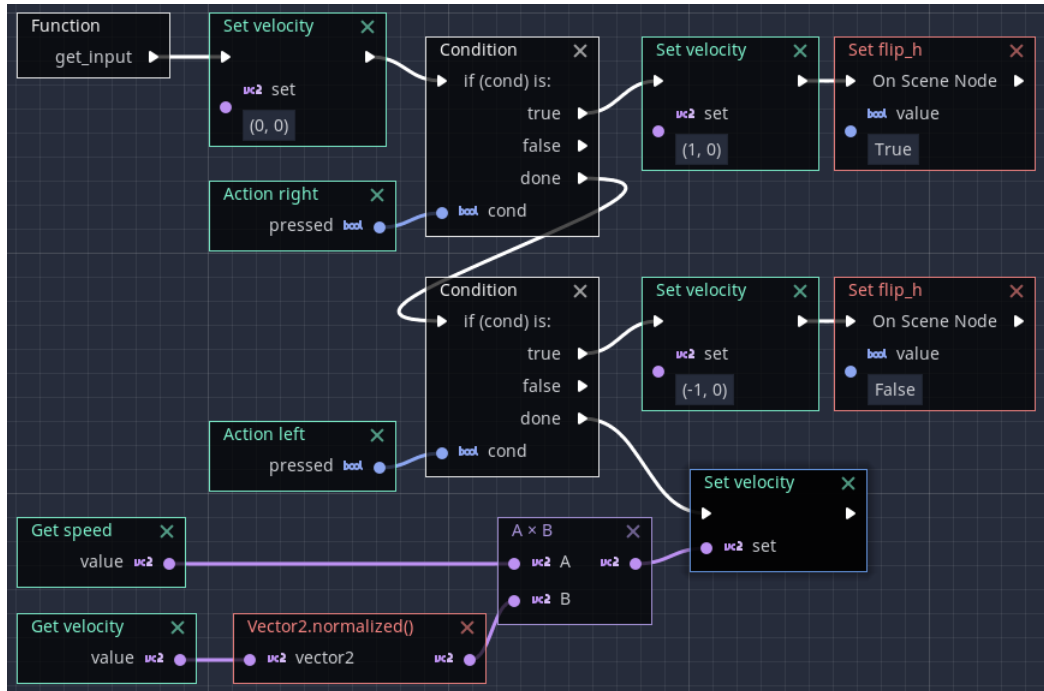


Fig. 31 – Fonction *get_input*.

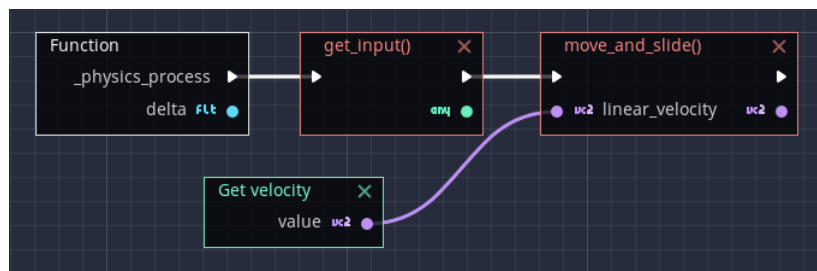





Fig. 32 – Procédure de déplacement de *Mario*.

_physics_process est appelée à la fréquence d'affichage (*i.e.* 60 fois par seconde); il est possible d'ajuster la fréquence d'appel avec le paramètre *delta*, dont la valeur par défaut est de 0.01666 pour 60 appels par seconde; le *Vector2* donné à *move_and_slide* est une vitesse en pixels par seconde.

Dans les sections suivantes, on utilise les actions prédéfinies *ui_left* et *ui_right* à la place des actions *Left* et *Right*, ce qui nous évite de les ajouter dans les paramètres du projet.

2.4 Déplacer un *sprite* associé à plusieurs images

L'interface contient un *sprite* de *Mario*; appuyer sur  et  permet respectivement de déplacer/orienter le *sprite* vers la gauche et la droite; appuyer sur  permet de baisser *Mario*.

On utilise deux images de *Mario* présentées en Fig. 33 et 34; pour réduire les problèmes d'alignement, les deux images sont de taille identique.



Fig. 33 – *Mario* debout.



Fig. 34 – *Mario* baissé.

On utilise les mêmes variables, les mêmes fonctions et la même arborescence que dans la section 2.11; dans le répertoire *assets*, on a les deux images de *Mario* (*Mario-001.png* pour *Mario* debout et *Mario-003.png* pour *Mario* baissé).

Pour réaliser cette interface, suivre la procédure ci-dessous :

- Dans le noeud *Main* de type *Node2D*, ajouter les noeuds *Mario* et *Sprite*
[Avec l'inspecteur, fixer *x* et *y* du champ *Position* de *Mario* à 300 et 200]
- Ajouter les variables *speed* et *velocity*
- Attacher un *visual script* *Mario.vs* au noeud *Mario*
- Ajouter les fonctions *get_input* et *_physics_process*
- Réaliser le graphe correspondant à la fonction *get_input* comme le présentent les Fig. 35 et 36
- La fonction *_physics_process* est identique à celle de la section 2.11
- Exécuter

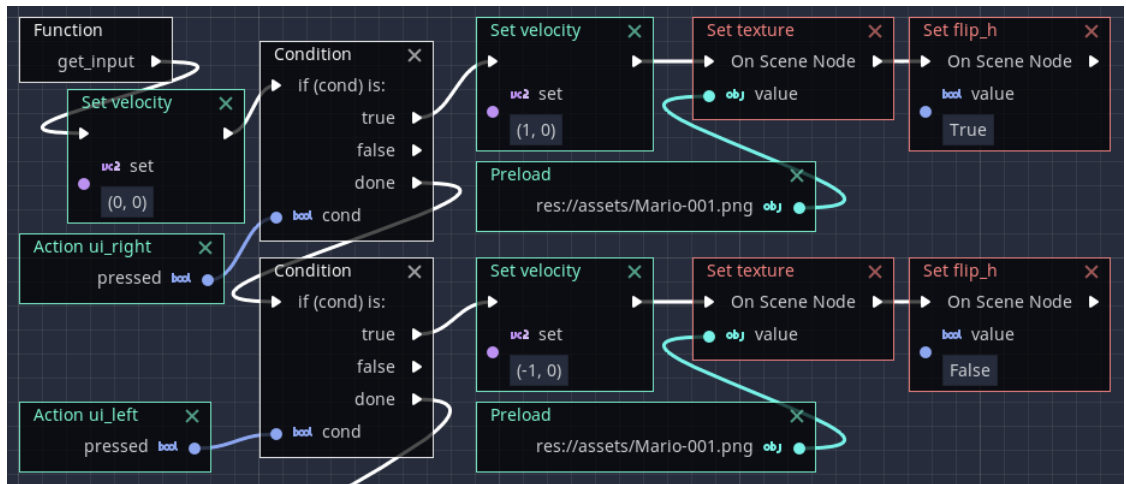


Fig. 35 – Gestion de  et  dans la fonction `get_input`.

La fonction *Set texture* utilisée dans la Fig. 35 est associée au noeud *Sprite* ; elle permet d'associer une image de *Mario* (debout ou baissé) correspondant à la position attendue ; les fonctions de *Preload* des images sont obtenues par glisser-déplacer d'une image dans le *visual script*.

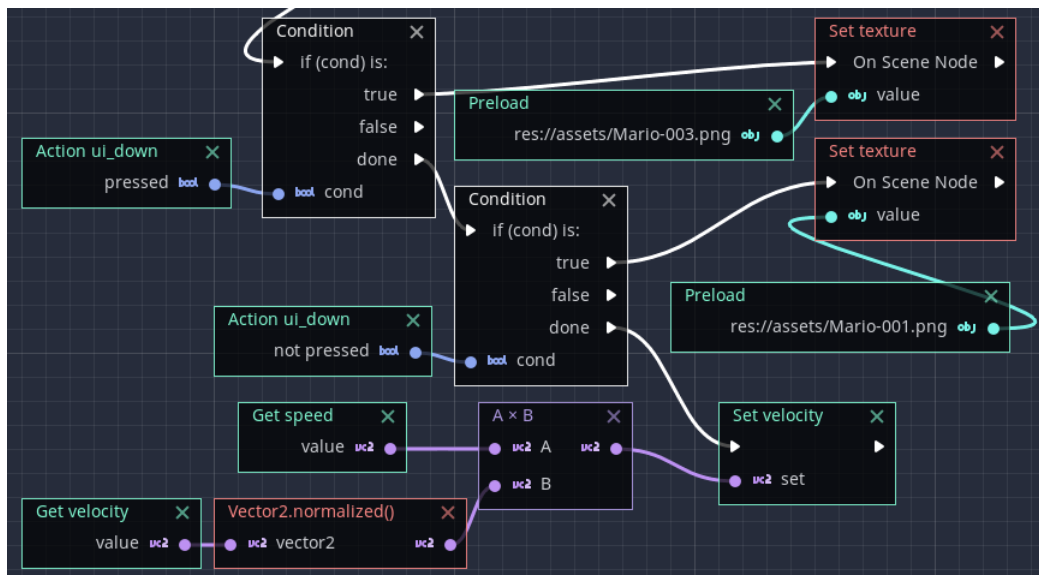





Fig. 36 – Gestion de  dans la fonction `get_input`.

2.5 Déplacer un *sprite* avec plusieurs animations

L'interface utilise un *sprite* animé avec trois animations (deux animations à une image et une animation à deux images); appuyer sur  et  permet respectivement de déplacer/orienter le *sprite* vers la gauche et la droite; appuyer sur  permet de baisser Mario; si aucune touche n'est pressée, Mario reprend sa position de départ, debout à l'arrêt.

On utilise les images de Mario de la section 2.4 et les deux images de Mario présentées en Fig. 37 et 38 permettant, si utilisées l'une après l'autre, de créer une animation de marche pour Mario.



Fig. 37 – Mario marche(0).



Fig. 38 – Mario marche(1).

Fig. 39 présente l'arborescence de la scène.

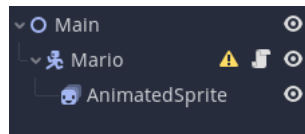


Fig. 39 – Arborescence.

Dans le répertoire *assets*, on a quatre images de Mario (Mario-001.png et Mario-003.png de la section 2.4, avec en plus les images Mario-005.png et Mario-006.png pour animer la marche de Mario).

Pour réaliser cette interface, suivre la procédure ci-dessous :

- Dans le noeud Main de type *Node2D*, ajouter les noeuds Mario et AnimatedSprite
[Mario est de type *KinematicBody2D*]
[AnimatedSprite est de type *AnimatedSprite*]
[Avec l'inspecteur, fixer x et y du champ *Position* de Mario à 300 et 200]
- Définir les animations du noeud AnimatedSprite
 - Créer un nouveau *SpriteFrame* dans le champ *Frames*
 - Créer trois animations Duck, Stand et Walk comme le présentent les Fig. 40, 41 et 42
- Définir un *visual script* en copiant-modifiant celui de la section 2.4
[Copier-coller le fichier Mario.vs dans le répertoire courant du projet]
[Associer le *visual script* Mario.vs par glisser-déposer dans le champ *Script* du noeud Mario]
[Redéfinir les objets représentés en rouge dans Mario.vs; ceci est nécessaire quand on change les types ou l'organisation des noeuds associés à un script]
[On retrouve les variables et les fonctions de la section 2.4]

- Modifier la fonction `get_input`
 - Commencer par définir l'animation quand aucune touche n'est pressée comme le présente la Fig. 43
 - Définir l'animation selon la touche pressée comme le présente la Fig. 44 [Normaliser le vecteur `velocity` en fin de fonction `get_input`]
- La fonction `_physics_process` reste inchangée
- Exécuter

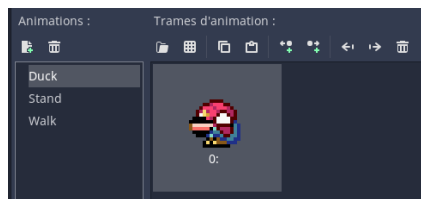


Fig. 40 – Animation Duck de *Mario*.

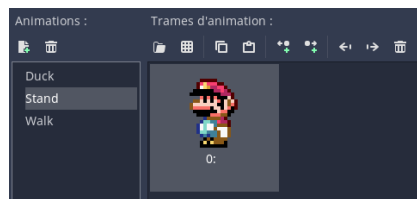


Fig. 41 – Animation Stand de *Mario*.

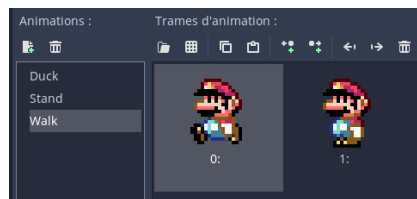


Fig. 42 – Animation Walk de *Mario*.

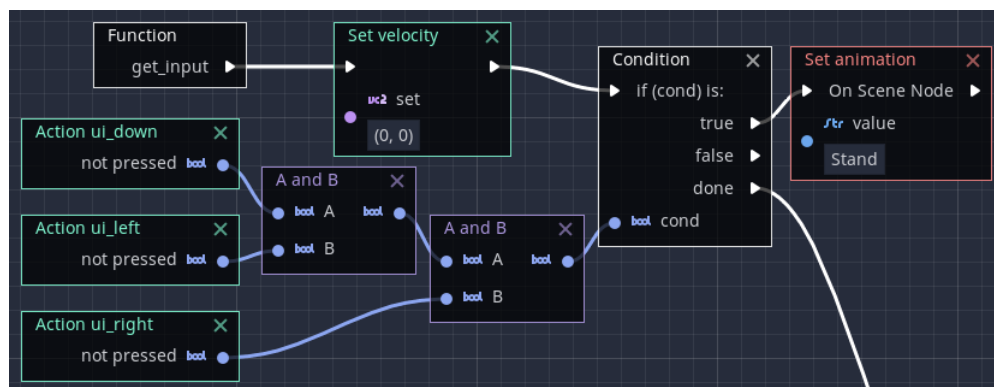





Fig. 43 – Quand aucune des touches ,  et  n'est pressée.

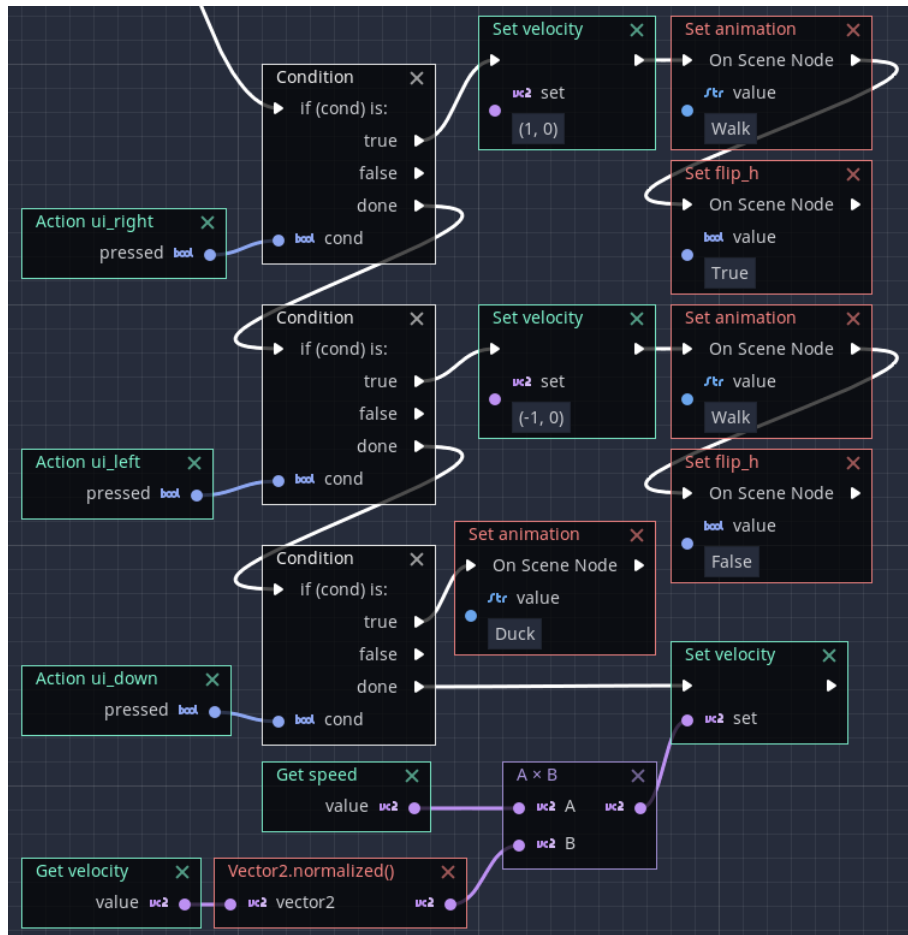





Fig. 44 – Quand une des touches ,  et  est pressée.

2.6 Animer quatre scintillements dans un cadre

On définit une animation de quatre scintillements en plaçant les scintillements dans les quadrants de l'image; chaque animation de scintillement commencent à des frames différentes.

On utilise les cinq images de scintillement présentées en Fig. 45, 46, 47, 48 et 49 permettant, si utilisées l'une après l'autre, de créer une animation d'un scintillement.

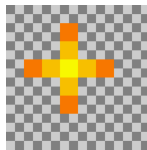


Fig. 45

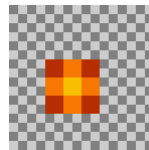


Fig. 46

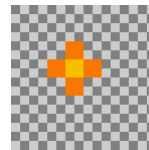


Fig. 47

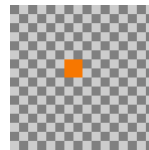


Fig. 48

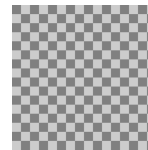


Fig. 49

Fig. 50 présente l'arborescence de la scène; Fig. 51 et 52 présentent les variables et les fonction de `Firework.vs` et de `Flicker.vs`.

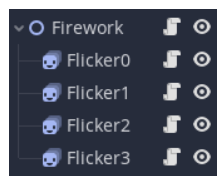


Fig. 50 – Arborescence de la scène.

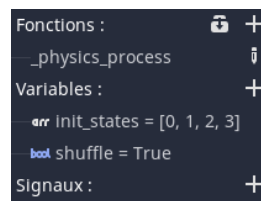


Fig. 51 – Fonctions de `Firework.vs`.

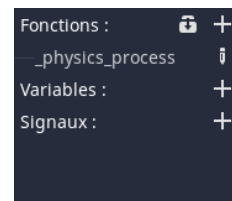


Fig. 52 – Fonctions de `Flicker.vs`.

Dans le répertoire `assets`, on a les cinq images de scintillement.

Pour réaliser cette interface, suivre la procédure ci-dessous :

- Dans le noeud `Firework` de type `Node2D`, ajouter un noeud `Flicker0` de type `AnimatedSprite`
- Définir les animations de `Flicker0` avec les cinq images de scintillement
- Activer le champ `Playing` et fixer le champ `Speed Scale` à 0.1
- Dans `Firework`, définir les variables `init_states` et `shuffle`
[`init_states` est de type `Array`, de taille 4 et initialisé à [0,1,2,3]]
[`shuffle` est une valeur booléenne initialisée à `True`]
- Associer à `Flicker0` le *visual script* `Flicker.vs` présenté en Fig. 53
- Dupliquer `Flicker0` pour obtenir quatre noeuds `Flicker0` à `Flicker3`
- Décaler les noeuds en modifiant le champ `Transform` avec les positions (0,0), (32,0), (0,32) et (32,32)
- Associer à `Firework` le *visual script* `Firework.vs` présenté en Fig. 54
[Les fonctions `Set frame` sont associées aux noeuds `Flicker0` à `Flicker3`]
- Exécuter permet d'obtenir les Fig. 55 à 59 (ou une autre combinaison de scintillements, selon le résultat de l'exécution de la fonction `Array.shuffle`)

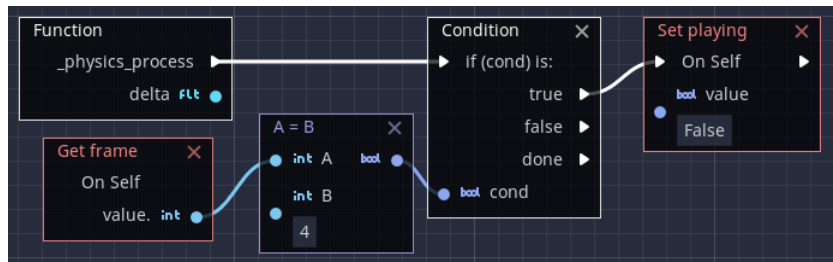


Fig. 53 – Flicker.vs.

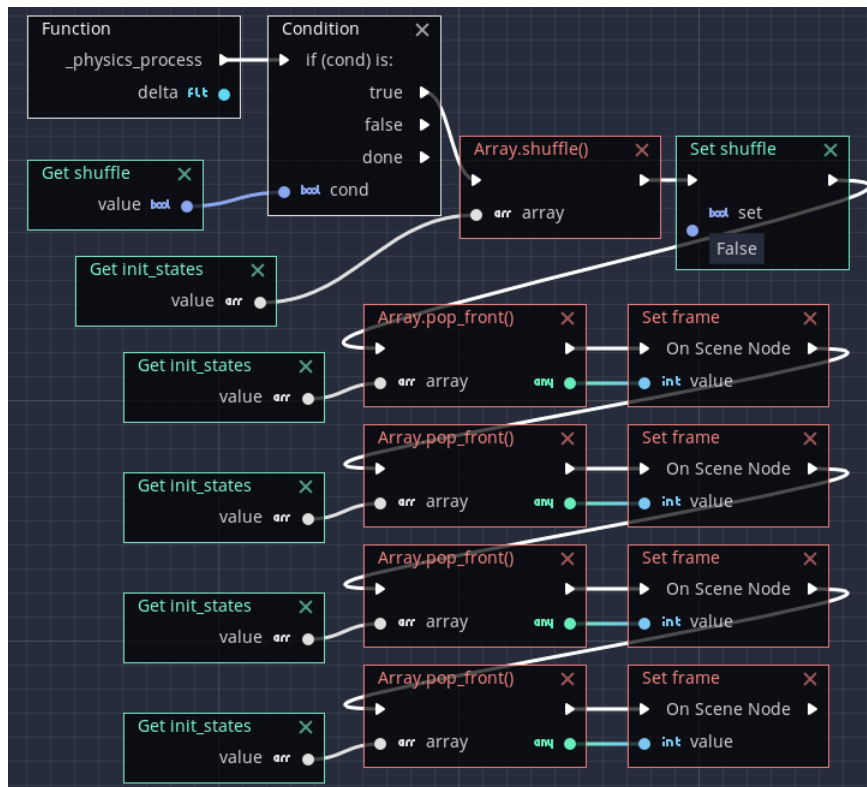


Fig. 54 – Firework.vs.



Fig. 55

Fig. 56

Fig. 57

Fig. 58

Fig. 59

2.7 Placer des décors et une caméra 2D

On place le *sprite* animé de la section 2.5 dans un décor en une image; la vue présentée par l'interface est centrée sur le personnage et le reste au fil de son déplacement; le déplacement de la caméra suit le déplacement du personnage.

Pour le décor, on utilise l'image présentée en Fig. 60.

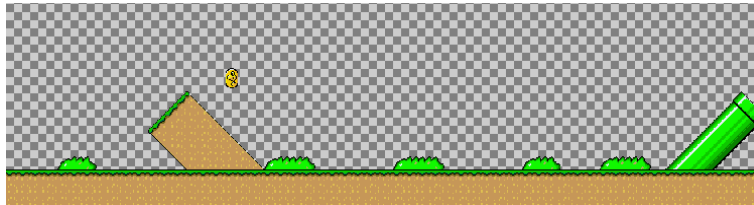


Fig. 60 – Décor utilisé.

Fig. 61 et 62 présentent l'arborescence de la scène, les fonctions et les variables de `Mario.vs`.

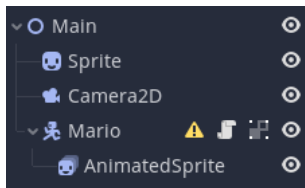


Fig. 61 – Arborescence.

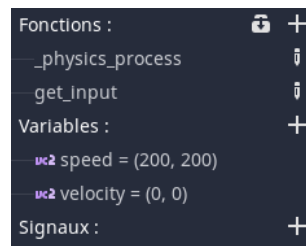


Fig. 62 – Fonctions de `Mario.vs`.

Pour réaliser cette interface, suivre la procédure ci-dessous :

- Dans le noeud `Main` de type *Node2D*, ajouter les noeuds `Mario` de type *KinematicBody2D* et `AnimatedSprite` de type *AnimatedSprite*
[Associer les noeuds `Mario` et `AnimatedSprite` permet de conserver l'alignement de leurs coordonnées; `AnimatedSprite` est en (0,0) par rapport à `Mario`; `Mario` est sur la ligne d'horizon de Fig. 60]
- Définir les animations du noeud `AnimatedSprite`
- Associer un *visual script* à `Mario` en copiant-modifiant celui de la section 2.5 [On retrouve les variables et les fonctions précédentes]
- Ajouter un noeud `Sprite` pour le décor (en lui associant l'image de la Fig. 60)
- Ajouter un noeud `Camera2D` de type *Camera2D*
- Cocher le champ *Current* de la caméra
- Placer la vue de la caméra dans l'interface centrée sur le personnage de `Mario`
[Avec l'inspecteur, on fixe *x* et *y* du champ *Position* en fonction de la position de `Mario` pour obtenir une vue centrée sur `Mario`]
- Modifier la fonction *_physics_process* en reliant un *Get position* de `Mario` vers un *Set position* de la caméra comme le présente la Fig. 63
- Exécuter permet d'obtenir la Fig. 64

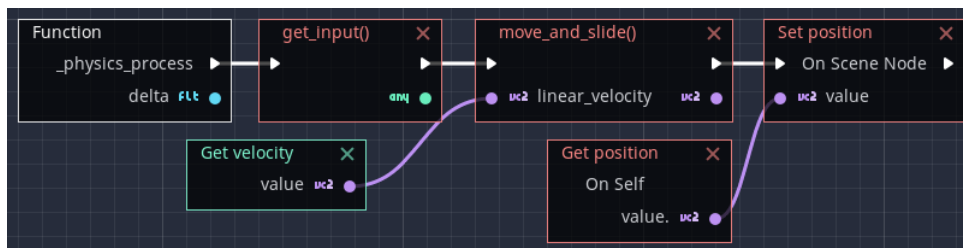


Fig. 63 – Fonction `_physics_process`.

La fonction `Get position` mentionne *On Self* pour indiquer qu'elle s'applique au noeud auquel est attaché le script (*i.e.* le noeud `Mario`) ; la fonction `Set position` mentionne *On Scene Node* pour indiquer qu'elle s'applique à un noeud de la scène (*i.e.* le noeud `Camera2D`).

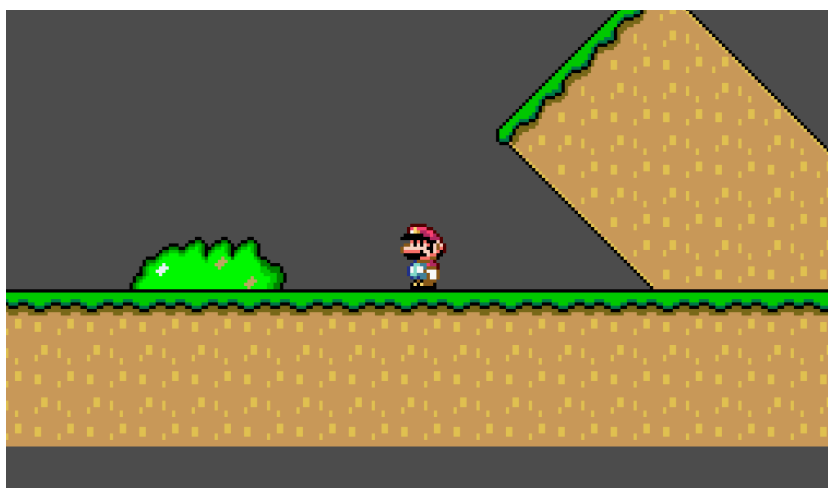





Fig. 64 – Appuyer sur  fait courir *Mario* vers la gauche, appuyer sur  fait courir *Mario* vers la droite, appuyer sur  fait se baisser *Mario* ; au fil des déplacements, la vue reste centrée sur *Mario*.

2.8 Définir une *tilemap*

On utilise une *tilemap* pour définir un niveau; le *sprite* animé de la section 2.7 se déplace dans ce niveau et la vue reste centrée sur le personnage.

L'image *tileset* (utilisée pour définir la *tilemap*) est présentée en Fig. 68.

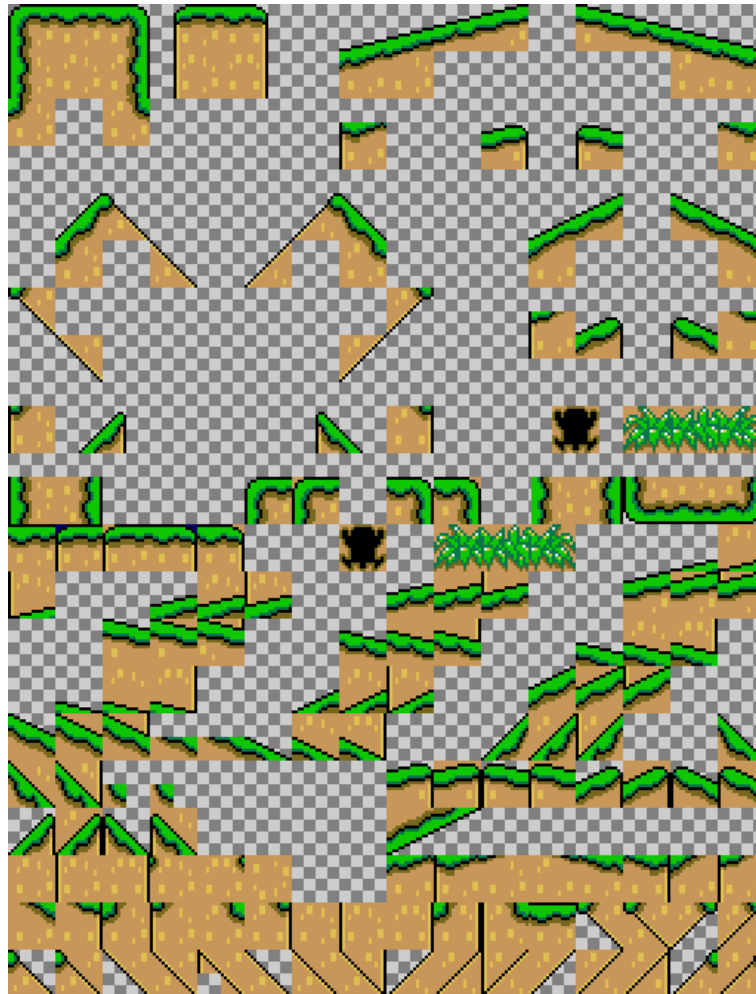


Fig. 65 – Image *tileset* utilisée.

Fig. 66 présente l'arborescence de la scène; *Mario* est dessiné sur la *tilemap*; les noeuds sont dessinés dans leur ordre d'apparition dans l'arborescence; les fonctions et les variables sont identiques à celles de la section précédente.

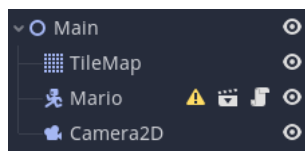


Fig. 66 – Arborescence.

Pour réaliser cette interface, suivre la procédure ci-dessous :

- Dans le noeud *Main* de type *Node2D*, ajouter les noeuds *Mario* et *AnimatedSprite*
- Ajouter les *assets* et définir les animations du noeud *AnimatedSprite*
- Associer un *visual script* au noeud *Mario* en copiant-modifiant celui de la section 2.5
[On retrouve les variables et les fonctions précédentes]
- Ajouter un noeud *Tilemap* de type *Tilemap*
[Glisser-déposer l'image *tileset* présentée en Fig. 68 dans le champ *Tileset*]
[Sélectionner l'image permet d'afficher le détail des tuiles sur une grille]
[Le quadrillage de cette grille est défini par les champs *Cell* de *TileMap*]
[Pour définir une nouvelle tuile, appuyer sur *+Nouvelle Simple Tuile* et sélectionner une zone correspondant à la tuile désirée]
[Le bouton « Région » est sélectionné]
[La tuile sélectionnée apparaît légèrement en plus foncée que les autres]
[Les tuiles définies sont délimitées par des contours jaunes comme le présente la Fig. 67]
[Les tuiles ajoutées sont affichées pour être glisser-déposer dans l'interface]
[Un click-gauche ajoute la tuile sélectionnée dans la case correspondant à la position de la souris dans l'interface]
[Un click-droit supprimer la tuile à la position souris dans l'interface]
- Définir une *tilemap* avec des tuiles comme le présente la Fig. 68
[Le personnage est placé en (0,0)]
[La caméra est placée sur le personnage en (0,0)]
[La *tilemap* est déplacée en (-128,80)]
- Ajouter un noeud *Camera2D* comme dans la section 2.7
- Ordonner les noeuds comme présenté sur la Fig. 66
- Exécuter permet d'obtenir la Fig. 69



Fig. 67 – Définition de tuiles dans un *tileset*; le quadrillage est de 32x32; les tuiles désirées sont de 64x64; les coordonnées des coins supérieurs gauche des tuiles en jaune sont (0,0), (2,0), (4,0), (7,0), (9,0), (0,2), (1,2), (2,2), (3,2), (4,2), (7,2), (8,2), (9,2), (0,4), (4,4).

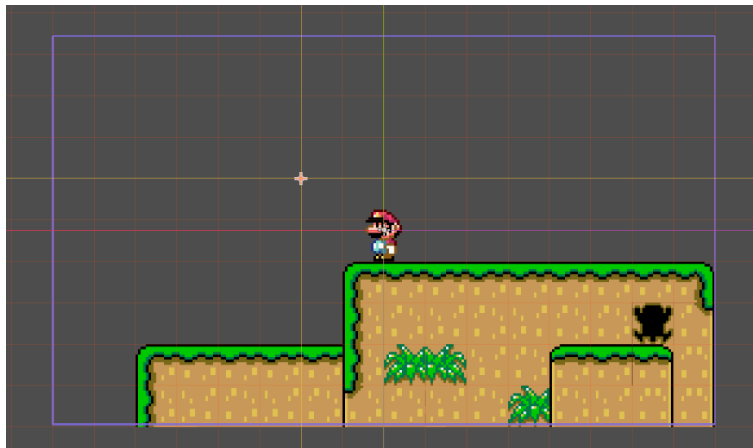


Fig. 68 – Construction d'une *tilemap* avec position initiale de la vue caméra.

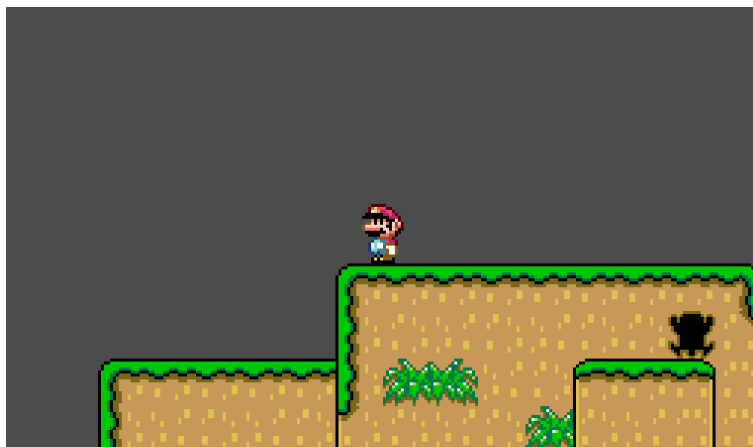


Fig. 69 – Vue initiale.

2.9 Définir un *parallax background*

On place *Mario* dans une *tilemap* avec un fond composé d'une couche se déplaçant moins vite que le décor de premier plan³.

L'image utilisée en *background* est présentée en Fig. 70.

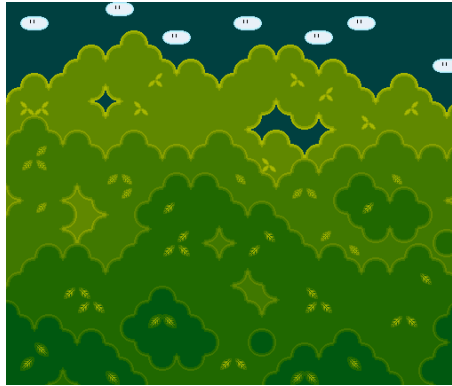


Fig. 70 – *Background* utilisé.

Fig. 71 présente l'arborescence de la scène.

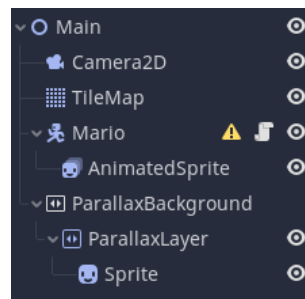


Fig. 71 – Arborescence.

Pour réaliser cette interface, suivre la procédure ci-dessous :

- Dans le noeud *Main* de type *Node2D*, ajouter les noeuds *Tilemap*, *Camera2D*, *Mario* et *AnimatedSprite*
- Ajouter les *assets* et définir les animations du noeud *AnimatedSprite*
- Associer un *visual script* au noeud *Mario* en copiant-modifiant celui de la section 2.8
- Définir les tuiles et la *tilemap* présentée en Fig. 72
- Ajouter les noeuds *ParallaxBackground* et *ParallaxLayer*
- Ajouter un noeud *Sprite* en lui associant l'image de 2048x1728 [Dans le champ *Transform*, fixer la valeur de *Position* en y à -140] [On obtient la Fig. 73]

3. Dans un *parallax background*, plus une couche est loin, plus elle se déplace lentement.

- Dans le champ *Motion* du noeud *ParallaxLayer*, fixer la valeur de *Scale* à 0.5 en x et la valeur de *Mirroring* à 2048 en x
[On obtient la Fig. 74]

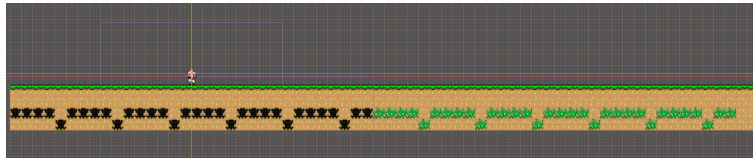


Fig. 72 – *Tilemap* de 66 tuiles de large; de gauche à droite, 32 tuiles avec des grenouilles cramées dessous, puis 32 tuiles avec des feuillages dessous, puis 2 tuiles.

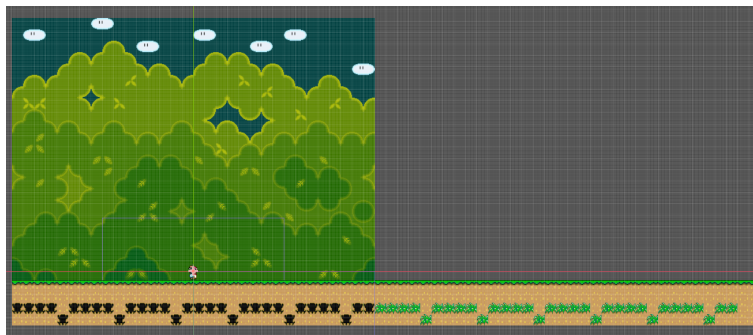


Fig. 73 – *tilemap* avec *background*.

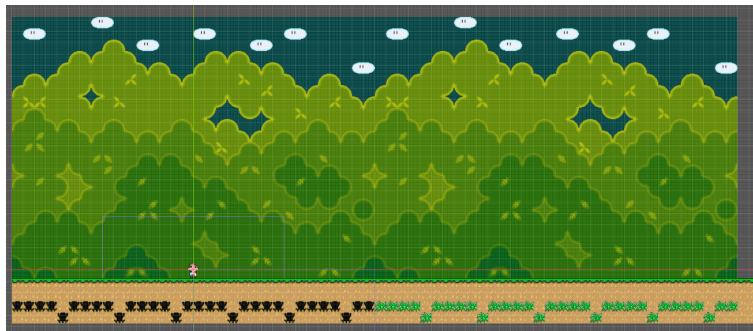


Fig. 74 – *Background* avec *Mirroring*.

- Exécuter permet d'obtenir la Fig. 75; constater la différence de vitesse de déplacement du fond et les bonnes transitions de *background* comme le présente à gauche la Fig. 76 et à droite la Fig. 77 (juste à gauche du groupe de quatre feuilles également à gauche de *Mario* dans les Fig. 76 et 77)



Fig. 75 – Position de départ.



Fig. 76 – Transition sur le *background*.



Fig. 77 – Répétition automatique du *background*.

2.10 Coordonner plusieurs *sprite*

L'interface définit l'animation de *Mario* placé sur un *Yoshi*; le *Yoshi* se déplace à gauche ou à droite; en appuyant sur **a**, le *Yoshi* gonfle ses joues; en appuyant sur **z**, le *Yoshi* dégonfle ses joues.

Pour *Mario*, on utilise une seule image présentée en Fig. 78; pour le *Yoshi*, on utilise les images présentées en Fig. 79, 80, 81, 82, 83 et 84.



Fig. 78 – *Mario* sur le *Yoshi*.



Fig. 79



Fig. 80

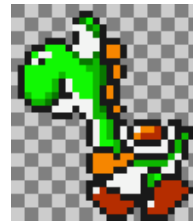


Fig. 81



Fig. 82



Fig. 83

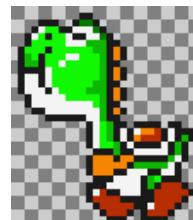


Fig. 84

On définit quatre animations pour le *Yoshi* :

- Idle avec la Fig. 79
- IdleLoaded avec la Fig. 82
- Run avec la séquence des Fig. 80, 81, 80 et 79
- RunLoaded avec la séquence des Fig. 83, 84, 83 et 82
- Le *Yoshi* a un *offset* de $(-36, -32)$ quand il va vers la gauche, et un *offset* de $(36, -32)$ quand il va vers la droite
- *Mario* a un *offset* de $(0, -64)$ quand le *Yoshi* est associé aux Fig. 79 et 82
- *Mario* a un *offset* de $(0, -60)$ quand le *Yoshi* est associé aux Fig. 80, 81, 83 et 84

Les variations d'*offset* produisent deux effets quand le *Yoshi* avance : 1) *Mario* monte et descend; 2) la tête oscille vers l'avant; pendant un cycle de Run correspondant à la suite des Fig. 80, 81, 80 et 79, les positions de *Mario* sont bas, bas, haut et les positions de la tête du *Yoshi* sont arrière, avant, arrière, avant.

Fig. 85 et 86 présentent l'arborescence de la scène, les fonctions et les variables de `Main.vs`.

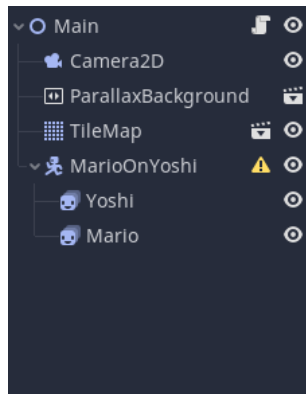


Fig. 85 – Arborescence.

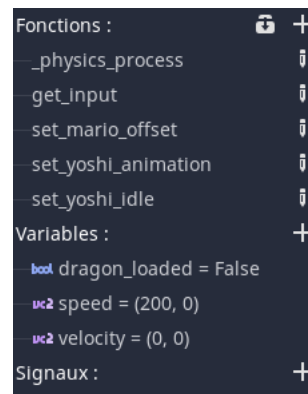


Fig. 86 – Fonctions et variables.

Pour réaliser cette interface, suivre la procédure ci-dessous :

- Dans le noeud `Main` de type *Node2D*, ajouter les noeuds `Camera2D`, `ParallaxBackground`, `Tilemap`, `MarioOnYoshi` de type *KinematicBody2D* et deux noeuds `Yoshi` et `Mario` de type *AnimatedSprite*
[En suivant la procédure de la section 3.1, on peut réutiliser les noeuds `ParallaxBackground` et `Tilemap` de la section 2.8]
- Ajouter dans les paramètres du projet, la prise en compte des actions associées à `[a]` et `[z]` (en les nommant respectivement `Load_dragon` et `Dragon`)
- Ajouter les *assets* de *Mario* et du *Yoshi*
- Définir les animations `Idle`, `IdleLoaded`, `Run` et `RunLoaded` du *Yoshi*
- Définir une animation `Idle` pour *Mario*
- Définir la *tilemap* présentée en Fig. 87
- Ajouter le *visual script* `Main.vs` au noeud `Main`



Fig. 87 – Simple *tilemap* avec *Mario* sur un *Yoshi*.

- Définir les fonctions *set_yoshi_animation* et *set_yoshi_idle* comme présenté en Fig. 88
[La variable *dragon_loaded* sélectionne la bonne séquence d’animation pour le Yoshi]

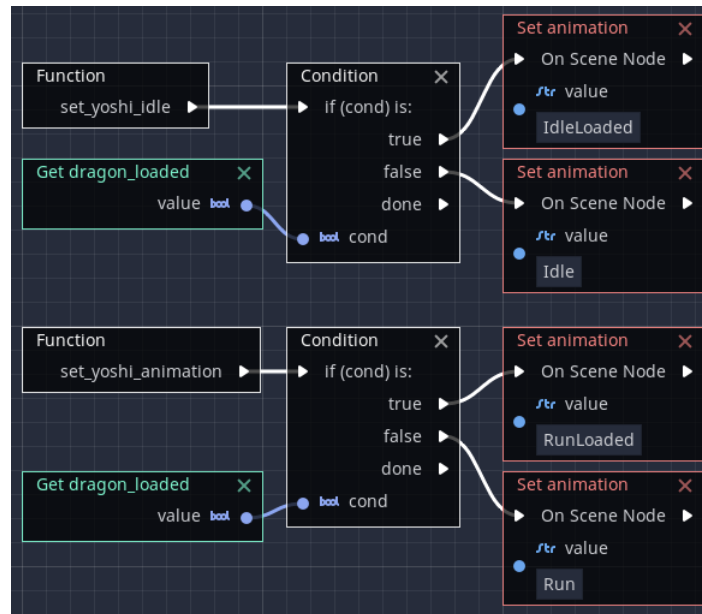


Fig. 88 – Fonctions *set_yoshi_animation* et *set_yoshi_idle*.

- Définir la fonction *set_mario_offset* comme présenté en Fig. 89

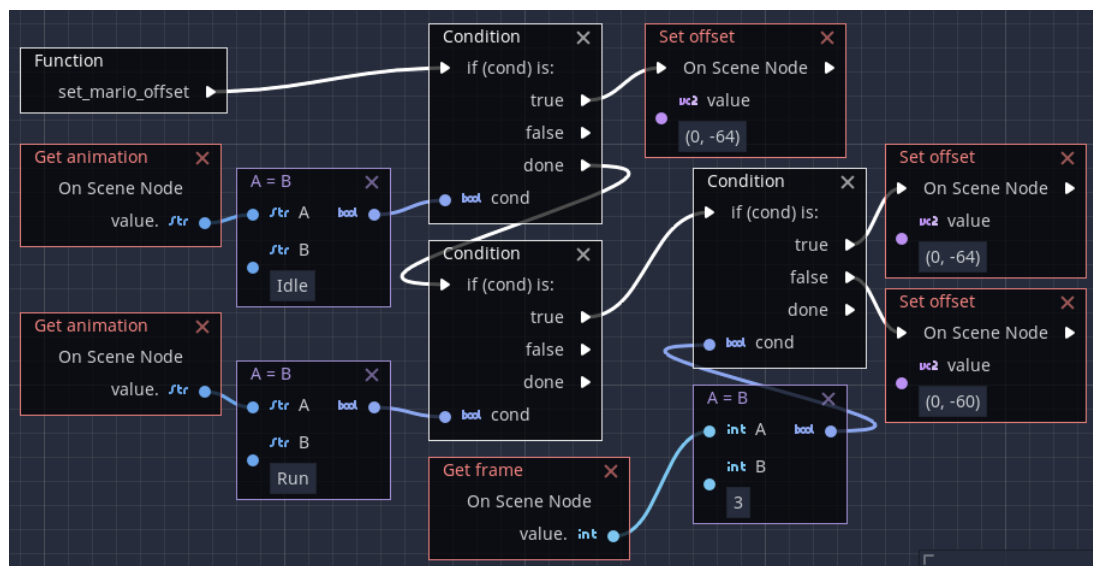


Fig. 89 – Fonctions *set_mario_offset*.

— Définir la fonction *get_input* comme présenté en Fig. 90 et 91

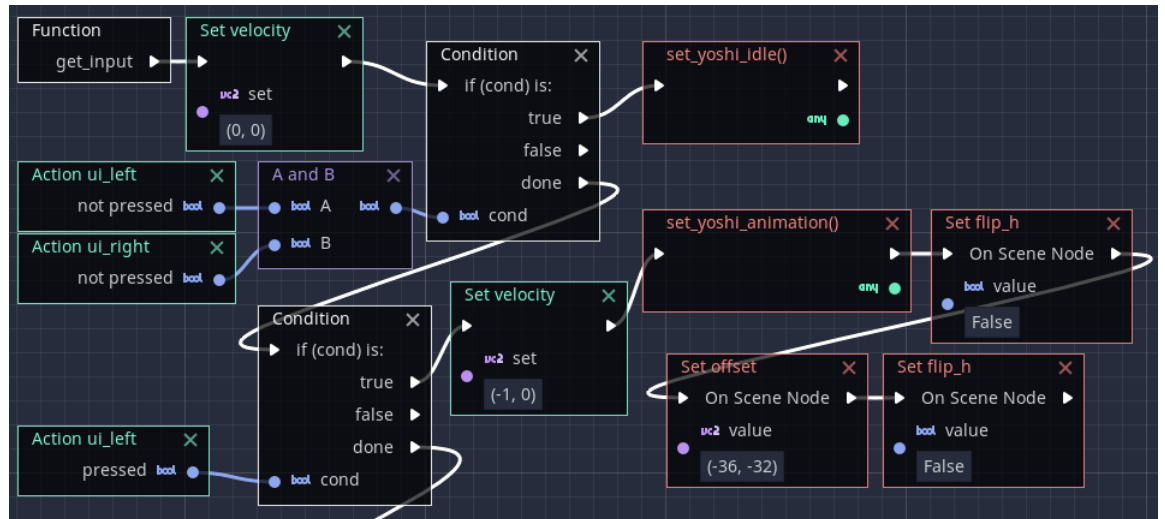


Fig. 90 – Début de la fonction *get_input*.

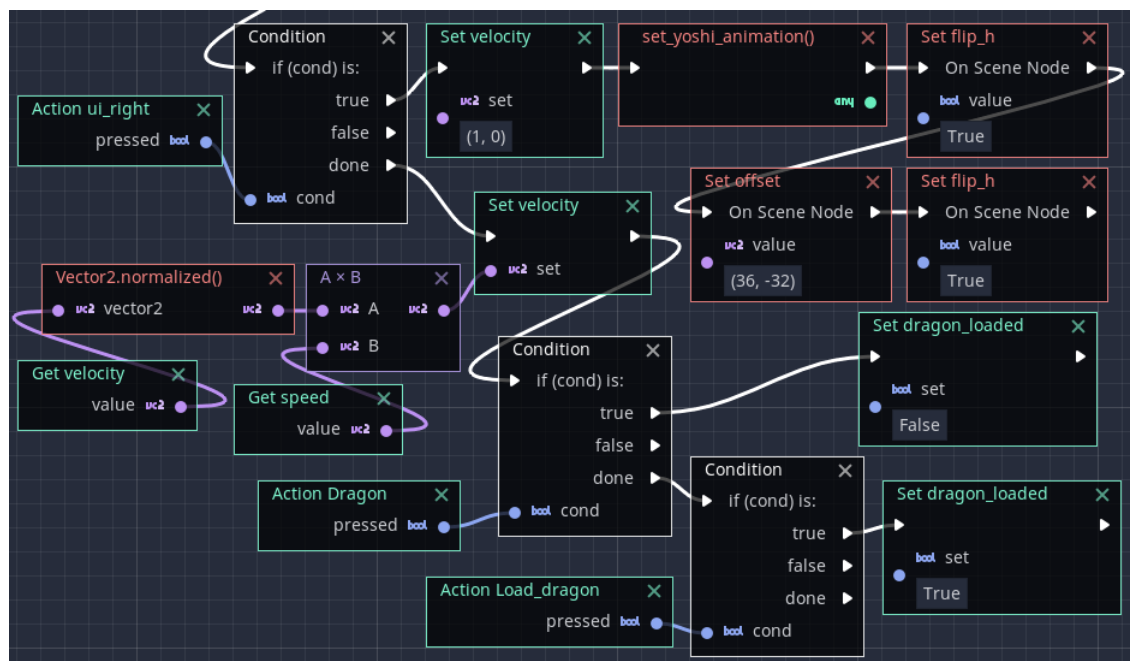


Fig. 91 – Fin de la fonction *get_input*.

— Définir la fonction `_physics_process` comme présenté en Fig. 92

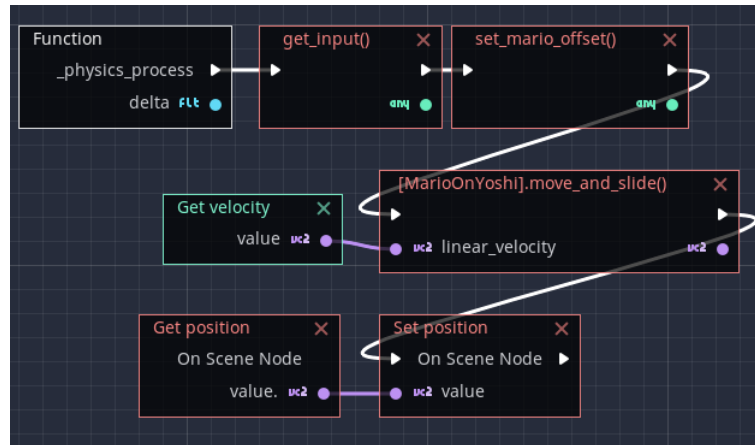


Fig. 92 – Fonction `_physics_process`.

2.11 Amortir les déplacements d'un *sprite*



L'interface contient un *sprite* de *Mario*; appuyer sur  et sur  permet de déplacer et orienter le *sprite* vers la gauche et vers la droite; l'amortissement est réalisé avec une interpolation linéaire; il permet d'obtenir une accélération et une décélération progressives au début et à la fin du déplacement.

Fig. 93 et Fig. 94 présentent l'arborescence, les fonctions et les variables utilisées.

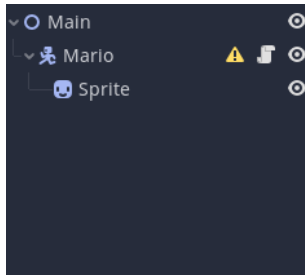


Fig. 93 – Arborescence.

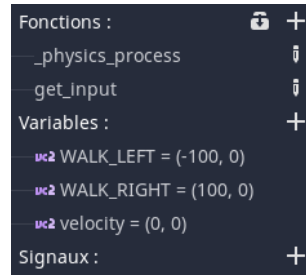


Fig. 94 – Fonctions et variables.

Pour réaliser cette interface, suivre la procédure ci-dessous :

- Dans le noeud Main de type *Node2D*, ajouter un noeud Mario de type *KinematicBody2D* et un noeud Sprite de type *Sprite*
[Associer au *sprite* l'image de *Mario* présentée en Fig. 24]
[Avec l'inspecteur, fixer x et y du champ *Position* de Mario à 300 et 200]
- Ajouter une variable *velocity* de type *Vector2* initialisée à (0,0)
- Ajouter une variable *WALK_LEFT* de type *Vector2* initialisée à (-100,0)
- Ajouter une variable *WALK_RIGHT* de type *Vector2* initialisée à (100,0)
[Par convention, les constantes sont en majuscules]
- Attacher un *visual script* *Mario.vs* au noeud Mario
- Définir la fonction *get_input* comme présentée en Fig. 95 et 96

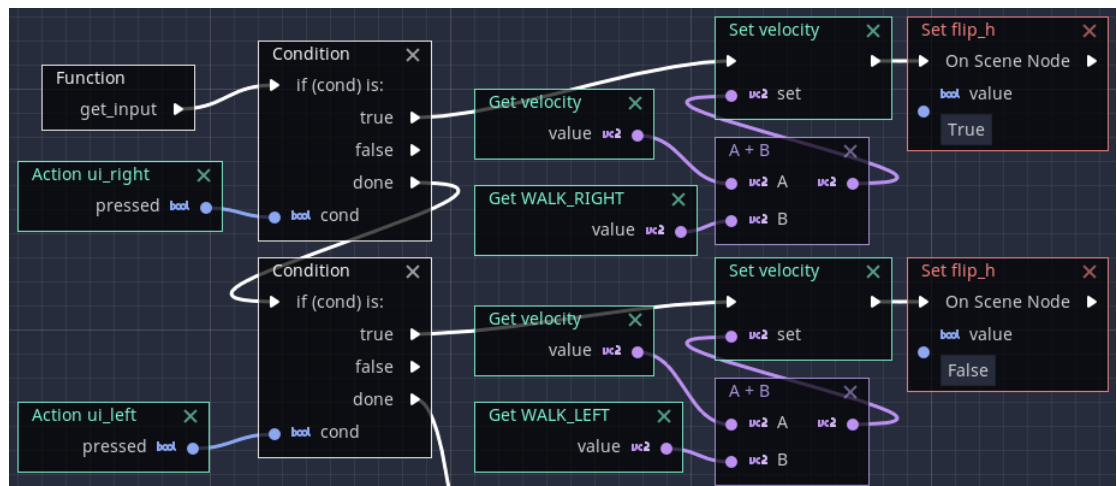


Fig. 95 – Fonction *get_input*.

[Fig. 95 fixe la valeur de `velocity` en fonction des touches clavier pressées]
 [Fig. 96 réalise une interpolation linéaire sur la valeur en x de `velocity`]

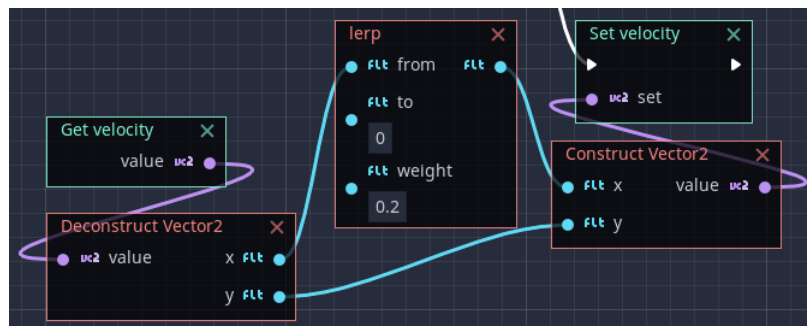


Fig. 96 – Début de la fonction `get_input`.

— Définir la fonction `_physics_process` comme le présente la Fig. 97

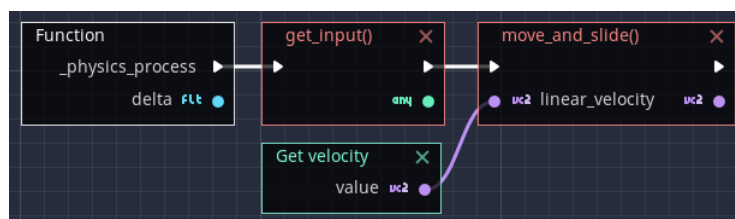




Fig. 97 – Fin de la fonction `_physics_process`.

— Exécuter et constater l'amortissement des déplacements

Comme dans la section :

- Une seule image présentée en Fig. 30 est utilisée
- `flip_h` oriente l'image dans le sens du déplacement

Contrairement à la section :

- Les actions `ui_left` et `ui_right` sont utilisées pour  et 
- Les deux variables `WALK_LEFT` et `WALK_RIGHT` remplacent la variable `speed`
- La variable `velocity` n'est pas normalisée et varie dans `[-100, 100]`
- La fonction `lerp` amortit les déplacements et fait évoluer `velocity` vers zéro avec le temps