

4 Collisions 2D

Les objets physiques (de type *PhysicsBody2D*) sont : 1) soit de type *StaticBody2D* quand ils ne bougent pas, comme les murs et les obstacles; 2) soit de type *RigidBody2D* quand ils bougent sans être contrôlé directement par le joueur, comme les objets lancés par le personnage principal ou les objets déplacés par l'environnement comme les chutes de pierres; 3) soit de type *KinematicBody2D* quand ils sont contrôlés par le joueur, comme un personnage principal contrôlé au clavier; pour contrôler entièrement la physique des objets manipulés, nous utilisons dans cette section des *KinematicBody2D* pour les objets physiques.

4.1 Détecter les collisions entre *sprite* et *tilemap*

On place *Mario* dans une *tilemap* correspondant à une plateforme avec une bordure gauche et une bordure droite comme le présente la Fig. 127; les déplacements de *Mario* vers la gauche et vers la droite sont limités par les bordures.



Fig. 127 – *Tilemap* utilisée.

Fig. 128 et 129 présentent l'arborescence de `Main.tscn` et de `Mario.tscn`; les noeuds `Mario`, `Tilemap` et `ParallaxBackground` sont des scènes (identifiées par des claps de cinéma); selon les cas, on pourra appliquer des modifications à une instance ou à l'ensemble des instances; toutes les opérations relatives aux instances sont dans les fichiers `.tscn`; la modification d'un fichier `.tscn` implique de sélectionner la scène concernée pour appliquer les opérations à toutes ses instances.

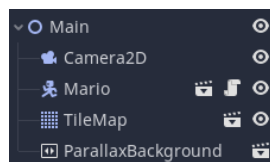


Fig. 128 – Arborescence principale.

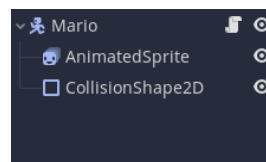


Fig. 129 – Arborescence de *Mario*.

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

- Ajouter les *assets*, les *visual script* et les fichiers `.tscn` pour les noeuds Mario, Tilemap, ParallaxBackground
- Dans le noeud Main de type *Node2D*, instancier les noeuds Mario, Tilemap, ParallaxBackground et ajouter un noeud Camera2D
- Cocher le champ *Current* de la caméra
- Redéfinir les tuiles de Tileset et définir la *tilemap* comme le présente la Fig. 127
- Pour les deux tuiles correspondant à des bordures montantes, ajouter des rectangles de collision aux tuiles comme le présentent les Fig. 130 et 131
- Ajouter un noeud CollisionShape2D avec une forme rectangulaire comme le présente la Fig. 132
- Ordonner les noeuds, exécuter et constater que les déplacements de *Mario* ne sont pas possibles sur les bordures montantes de la *tilemap*



Fig. 130 – Collision de bordure gauche.

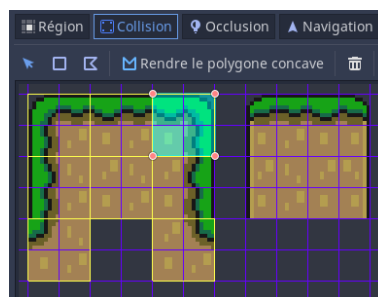


Fig. 131 – Collision de bordure droite.



Fig. 132 – Collision pour le *sprite* de *Mario*; à gauche, la forme de collision est un rectangle; à droite, la forme de collision est composée de deux cercles pour plus de précision dans la détection des collisions; la définition de formes de collision sans être à l'extérieur des sprites permet d'éviter la détection de fausses collisions; il s'agit de choisir le niveau de précision de détection de collision adéquate selon le jeu, les décors et l'interaction désirée.

4.2 Coordonner animation, collision et suppression

On place *Mario* dans une *tilemap* avec deux pièces comme le présente la Fig. 133 ; quand *Mario* est en collision avec une pièce, la pièce disparaît.



Fig. 133 – *Mario* avec deux pièces sur une plateforme fixe.

Fig. 134 et 135 présentent l'arborescence de la scène principale et de la scène *Coin*.

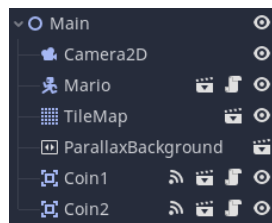


Fig. 134 – Arborescence principale.

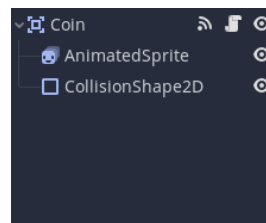


Fig. 135 – Arborescence de *Coin*.

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

- Ajouter les *assets*, les *visual script* et les fichiers *.tscn* pour les noeuds *Mario*, *Tilemap*, *ParallaxBackground*
- Dans le noeud *Main* de type *Node2D*, instancier les noeuds *Mario*, *Tilemap*, *ParallaxBackground* et ajouter un noeud *Camera2D*
- Cocher le champ *Current* de la caméra
- Définir la *tilemap* comme le présente la Fig. 133
- Définir une nouvelle scène *Coin* de type *Area2D*
- Ajouter un noeud *AnimatedSprite* avec une animation *Turn* présentée en Fig. 136
- Activer le champ *Playing*
- Ajouter un noeud *CollisionShape2D* de forme *RectangleShape2D* présentée en Fig. 137
- Associer un *visual script* nommé *Coin.vs*
- Ajouter les fonctions *_on_Coin_body_entered* et *queue_free* comme présenté en Fig. 138

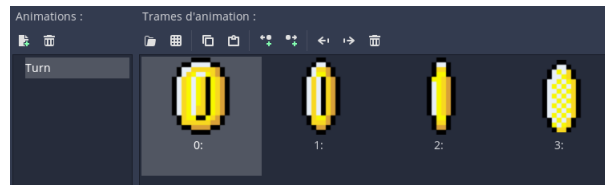


Fig. 136 – Animation Turn de la pièce.

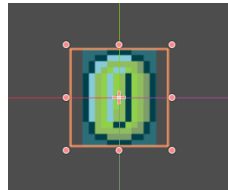


Fig. 137 – *RectangleShape2D* de la pièce; la zone de collision est dessinée en vert.

- [La fonction *queue_free* demande la suppression de l'objet courant]
- Instancier les noeuds *Coin1* et *Coin2* de type *Coin* dans *Main.tscn*
 - Exécuter et constater la disparition des pièces quand *Mario* passe dessus

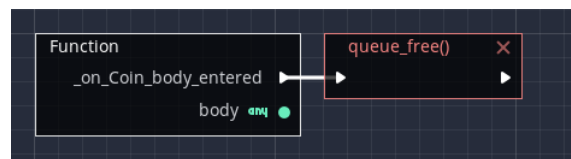


Fig. 138 – Suppression d'une pièce.

4.3 Coordonner animation, collision, scintillement et suppression

On place *Mario* dans une *tilemap* avec quatre pièces comme le présente la Fig. 139 ; quand *Mario* est en collision avec une pièce, la pièce disparaît, des étoiles scintillent puis disparaissent comme le présente la Fig. 140 ; quand le scintillement d'étoiles arrive à la dernière *frame* de l'animation, la pièce appelle la fonction *queue_free*⁴.



Fig. 139 – *Mario* et quatre pièces à collecter.



Fig. 140 – *Mario* en train de collecter des pièces ; les pièces collectées font apparaître des scintillements.

4. Le scintillement d'étoiles est ici réalisé différemment de la scène de la section 2.6. On utilise une animation constituée de quatre images ; on applique les fonctions *flip_h* et *flip_v* aléatoirement pour obtenir différents scintillements.

Fig. 141 et 142 présentent l'arborescence de la scène principale et de la scène coin.

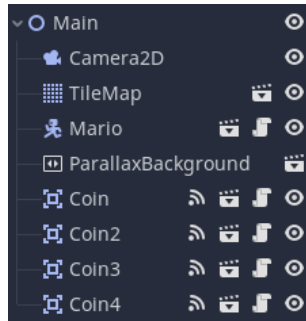


Fig. 141 – Scène principale.

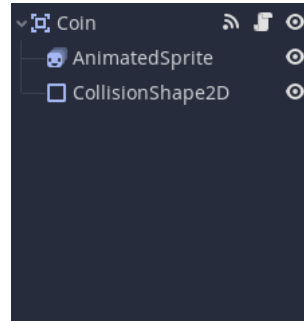


Fig. 142 – Scène coin.

Fig. 143 et 144 présentent les fonctions et les variables de la scène principale et de la scène coin.

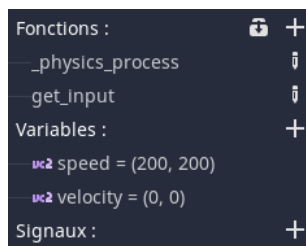


Fig. 143 – Eléments de Main .vs.

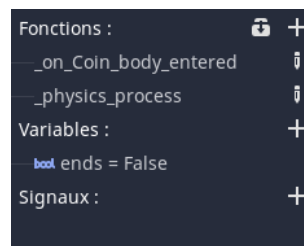


Fig. 144 – Eléments de Coin .vs.

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

- Ajouter les *assets*, les *visual script* et les fichiers *.tscn* pour les noeuds Mario, Tilemap, ParallaxBackground et Coin de la section 4.2
- Dans le noeud Main de type *Node2D*, instancier les noeuds Mario, Tilemap, ParallaxBackground, Coin et ajouter un noeud Camera2D
- Cocher le champ *Current* de la caméra
- Définir la *tilemap* pour obtenir une simple plateforme comme le présente la Fig. 140
- Modifier la scène *Coin*
- Ajouter une animation Firework présentée en Fig. 145

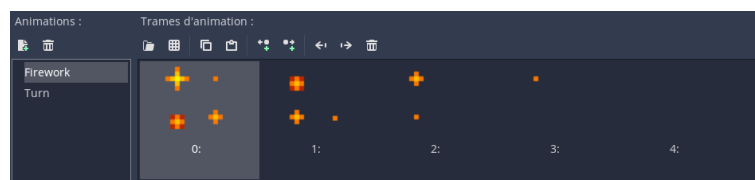





Fig. 145 – Animation des scintillements.

-

56

4.4 Détecter les bords de plateforme

On place *Mario* dans une *tilemap* avec cinq plateformes comme le présente la Fig. 147; on peut déplacer *Mario* sur la première plateforme avec ,  et ; on place un *Koopa* sur chacune des plateformes restantes; les *Koopa* réalisent des rondes de gauche à droite sur leur plateforme; au moment de réaliser leur demi-tour, les *Koopa* font un bref arrêt en position de demi-tour.

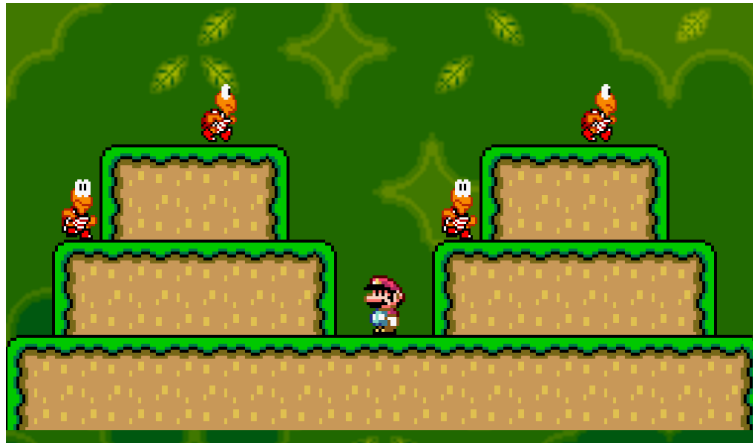


Fig. 147 – Mario et quatre Koopa.

Pour limiter les déplacements des *Koopa*, on place des *Area2D* aux niveaux des bords des plateformes comme présenté sur la Fig. 148; chaque *Koopa* est associé à deux *Area2D* qui déclenchent un signal pour réaliser un demi-tour et inverser le sens de déplacement.

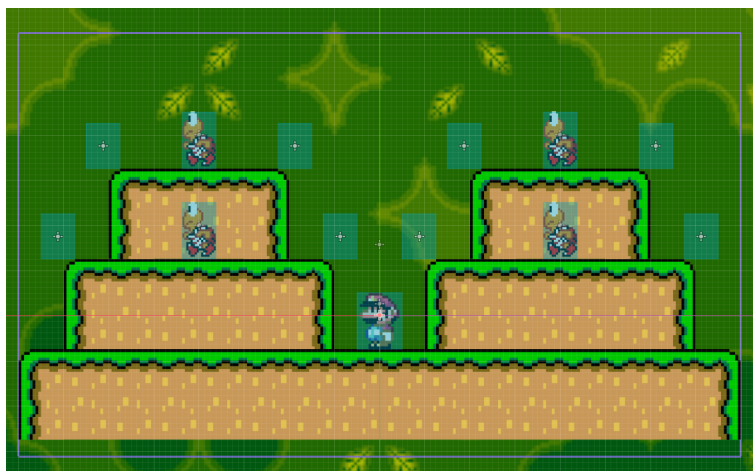


Fig. 148 – Mario, quatre Koopa et huit Area2D.

Fig. 149 présente l'arborescence de la scène principale; Fig. 150 présente les fonctions et les variables de la scène nommée `koopas.tscn` correspondant à l'animation d'un *Koopa*; la fonction `_on_Area2D_body_entered` (respect. *Area2D2*, *Area2D3* et *Area2D4*) correspond à la première plateforme (respect. deuxième, troisième et quatrième).

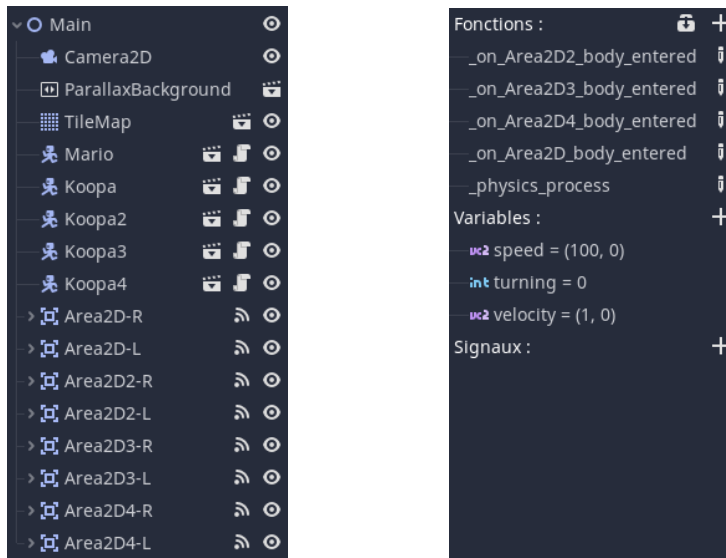


Fig. 149 – Arborescence.

Fig. 150 – Variables et fonctions d'un *Koopa*.

Les noeuds `Area2D-R` et `Area2D-L` activent la fonction `_on_Area2D_body_entered` quand le noeud *Koopa* entre dans l'une de ces *Area2D* (respect. *Area2D2-R* et *L* pour *Koopa2*, *Area2D3-R* et *L* pour *Koopa3*, *Area2D4-R* et *L* pour *Koopa4*).

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

- Ajouter les *assets*, les *visual script* et les fichiers `.tscn` pour les noeuds *Mario*, *Tilemap* et *ParallaxBackground* de la section 4.1
- Dans le noeud *Main* de type *Node2D*, instancier les noeuds *Mario*, *Tilemap*, *ParallaxBackground* et ajouter un noeud *Camera2D*
- Cocher le champ *Current* de la caméra
- Définir la *tilemap* pour obtenir les cinq plateformes de la Fig. 147
- Définir une nouvelle scène *Koopa.tscn* dont le noeud racine est de type *KinematicBody2D* [dans la scène *Koopa*]
- Ajouter un noeud *AnimatedSprite* avec une animation *Turn* composée de l'image présentée Fig. 155 et une animation *Walk* composée de deux images présentées en Fig. 153 et 154
- Ajouter un noeud *CollisionShape2D* utile à la détection de collision d'un *Koopa*
- Ajouter les variables *speed*, *turning* et *velocity* présentée en Fig. 150
- Associer *Koopa.vs* à cette scène comme le présente les Fig. 151 et 152

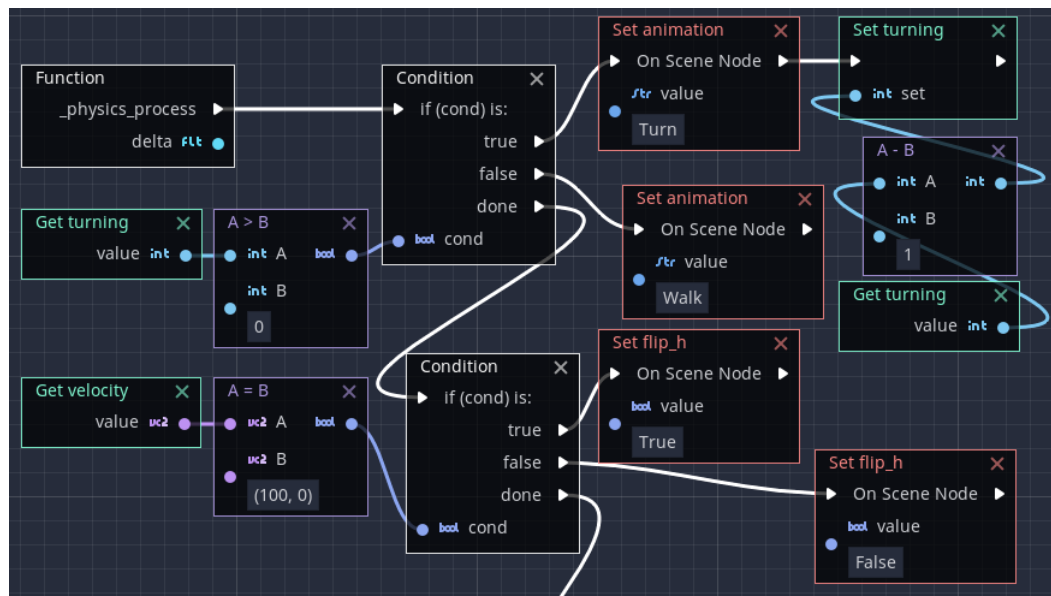


Fig. 151 – Début de *_physics_process* de Koopa .vs.

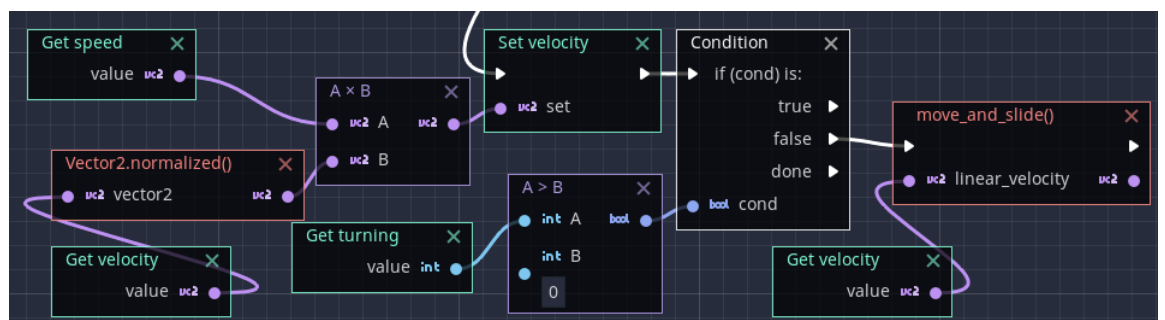


Fig. 152 – Fin de *_physics_process* de Koopa .vs.



Fig. 153 – koopa_006 .png. Fig. 154 – koopa_007 .png. Fig. 155 – koopa_008 .png.

[dans la scène principale]

- Dupliquer le *Koopa* pour obtenir les quatre *Koopa*
 - Ajouter à droite du premier *Koopa* une *Area2D* nommée *Area2D-R*
 - Associer à *Area2D-R* la fonction *_on_Area2D_body_entered* liée à *Koopa* [dans la scène *Koopa*]
 - Définir la fonction *_on_Area2D_body_entered* comme le présente la Fig. 156
 - Dupliquer *Area2D-R* pour obtenir *Area2D-L*
- [Quand *Koopa* entre dans *Area2D-R* ou *Area2D-L*, la fonction *_on_Area2D_body_entered* est appelée; le déplacement est inversé avec la fonction $\neg A$ obtenue à l'aide de la fonction *Math Negate*; la variable *turning* est fixée à 20 pour produire une pause]
- [Dans la fonction *_physics_process* de la Fig. 151, on commence par appeler *Set animation* à *Turn* en décrémentant *Turn*; quand *Turn* n'est pas supérieure stricte à 0, on appelle *Set animation* à *Walk*]

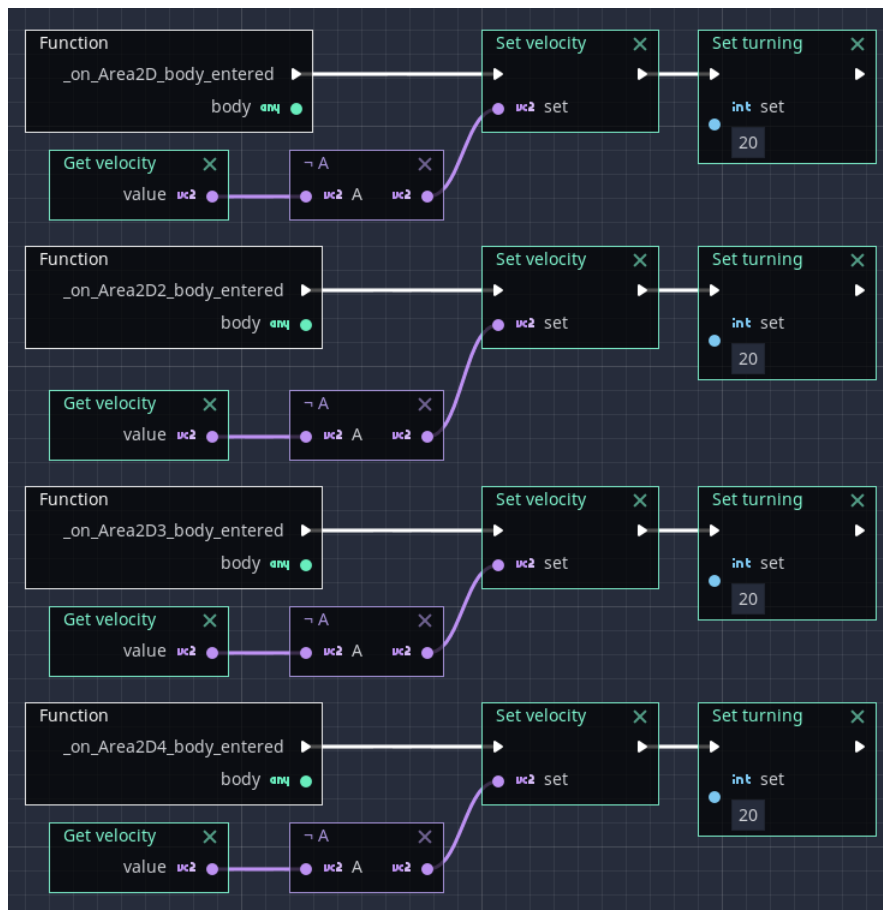


Fig. 156 – Fonctions *_on_Area2DXXX_body_entered* de *Koopa*. vs.

- [dans la scène principale]
- Ajouter à droite du deuxième *Koopa* une *Area2D* nommée *Area2D2-R*
 - Associer à *Area2D2-R* la fonction *_on_Area2D2_body_entered* liée à *Koopa2* [dans la scène *Koopa*]
 - Définir la fonction *_on_Area2D2_body_entered* comme le présente la Fig. 156
 - Dupliquer *Area2D2-R* pour obtenir *Area2D2-L* [dans la scène principale]
 - Ajouter à droite du troisième *Koopa* une *Area2D* nommée *Area2D3-R*
 - Associer à *Area2D3-R* la fonction *_on_Area2D3_body_entered* liée à *Koopa3* [dans la scène *Koopa*]
 - Définir la fonction *_on_Area2D3_body_entered* comme le présente la Fig. 156
 - Dupliquer *Area2D3-R* pour obtenir *Area2D3-L* [dans la scène principale]
 - Ajouter à droite du quatrième *Koopa* une *Area2D* nommée *Area2D4-R*
 - Associer à *Area2D4-R* la fonction *_on_Area2D4_body_entered* liée à *Koopa4* [dans la scène *Koopa*]
 - Définir la fonction *_on_Area2D4_body_entered* comme le présente la Fig. 156
 - Dupliquer *Area2D4-R* pour obtenir *Area2D4-L* [dans la scène principale]
 - Placer les *Area2D* autour de leur *Koopa* respectif
 - Exécuter permet d'obtenir la Fig. 147 quand les *Koopa* font leur ronde

4.5 Ramasser une carapace de Koopa





On place *Mario* sur une plateforme avec une carapace rouge de *Koopa*; on peut déplacer *Mario* avec , ; quand *Mario* est juste à droite de la carapace, il peut la ramasser avec  puis se déplacer avec la carapace dans les mains comme le présente la Fig. 157.



Fig. 157 – *Mario* avec une carapace de *Koopa* dans les mains.

Pour limiter la prise en main de la carapace, on place une *Area2D* à droite de la carapace et une *Area2D* sur la carapace comme présenté sur la Fig. 158; chaque *Area2D* déclenche un signal pour mettre à jour une variable booléenne; quand les variables booléennes ont les bonnes valeurs et que le joueur appuie sur , la carapace passe dans les mains de *Mario*.

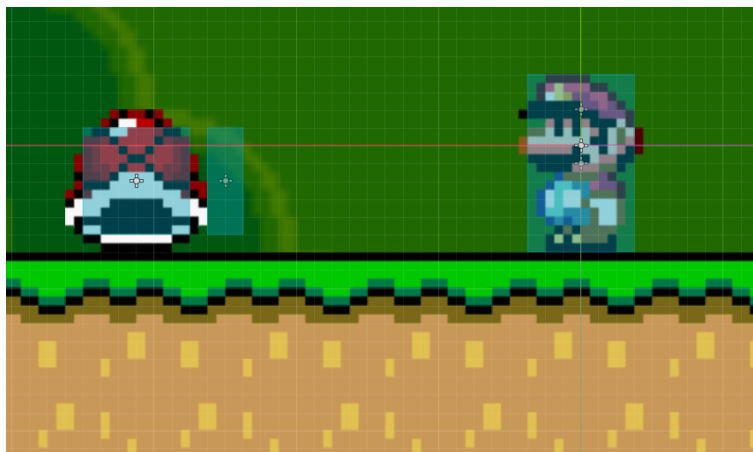


Fig. 158 – La carapace de *Koopa* est associée à deux *Area2D*: une pour la zone correspondant à son corps et une pour la zone à partir de laquelle il est possible de la saisir par la droite; *Mario* est associé à une *CollisionShape2D* de forme rectangulaire.

On reprend les animations *Stand* et *Walk* de *Mario* présentée à la section 2.5 et on ajoute les animations *Stand-Hold* et *Walk-Hold* présentée en Fig. 159 et 160.

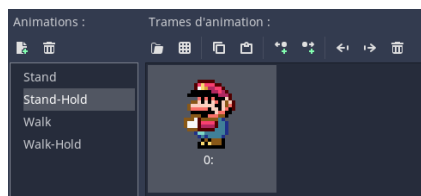


Fig. 159 – Animation *Stand-Hold*.

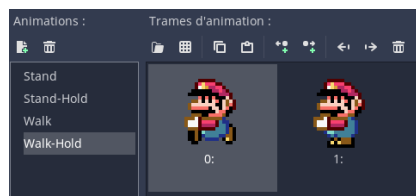


Fig. 160 – Animation *Walk-Hold*.

Fig. 161 présente l'arborescence de la scène principale; la carapace nommée *Shell* est de type *KinematicBody2D* et est associée à deux *Area2D*; la première est nommée *RightCatchArea* et définit sa zone de saisie; la seconde est nommée *BodyArea* et définit la zone de collision correspondant à son corps; le script *Main.vs* gère le déplacement de la caméra et le script *Mario.vs* gère l'animation de *Mario* et le positionnement de la carapace quand elle est portée par *Mario*.

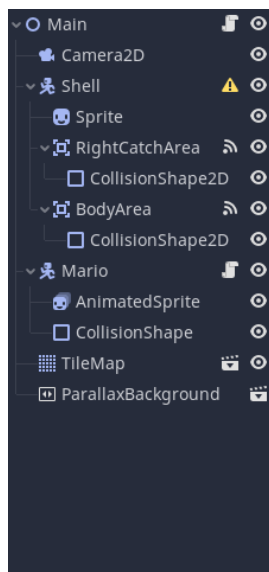


Fig. 161 – Arborescence.

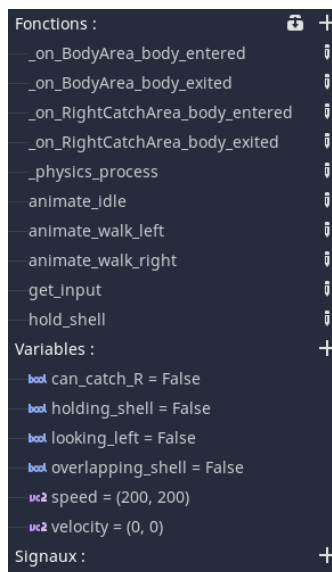



Fig. 162 – Variables et fonctions de *Mario.vs*.

Fig. 162 présente les fonctions et les variables associées à *Mario* :

- La variable *can_catch_R* est à *True* quand *Mario* est en collision avec *RightCatchArea*
- La variable *holding_shell* est à *True* quand *Mario* porte la carapace
- La variable *looking_left* est à *True* quand *Mario* est tourné vers la gauche
- La variable *overlapping_shell* est à *True* quand *Mario* est en collision avec *BodyArea*
- La fonction *_on_BodyArea_body_entered* est appelée quand la forme associée à *Mario* entre en collision avec *BodyArea*

- La fonction *_on_BodyArea_body_exited* est appelée quand la forme associée à *Mario* sort de collision avec *BodyArea*
- La fonction *_on_RightCatchArea_body_entered* est appelée quand la forme associée à *Mario* entre en collision avec *RightCatchArea*
- La fonction *_on_RightCatchArea_body_exited* est appelée quand la forme associée à *Mario* sort de collision avec *RightCatchArea*
- La fonction *animate_idle* gère les animations *Stand* et *Stand-Hold*
- Les fonctions *animate_walk_left* et *animate_walk_right* gèrent les animations *Walk* et *Walk-Hold*
- La fonction *Hold_shell* calcule la position de la carapace à partir de la position de *Mario*
- La fonction *get_input* gère les conséquences des actions du joueur sur les touches clavier
- La fonction *_physics_Process* appelle en séquence la gestion des touches clavier, le déplacement de *Mario* et le positionnement de la carapace

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

- Ajouter les *assets*, les *visual script* et les fichiers *.tscn* pour les noeuds *Mario*, *Tilemap* et *ParallaxBackground* de la section 4.1
- Dans le noeud *Main* de type *Node2D*, instancier les noeuds *Mario*, *Tilemap*, *ParallaxBackground* et ajouter un noeud *Camera2D*
- Cocher le champ *Current* de la caméra
- Définir la *tilemap* pour obtenir la plateforme de la Fig. 157
- Ajouter dans les paramètres du projet, la prise en compte de l'action associée à  nommée *Catch*
- Ajouter un noeud *Shell* de type *KinematicBody2D* à gauche de *Mario* comme présenté en Fig. 158
 - [Avec un noeud *Sprite* pour dessiner la carapace]
 - [Avec les *Area2D* *RightCatchArea* et *BodyArea* et leur *CollisionShape2D*]
- Ajouter les variables dans *Mario.tscn* vs présentées dans la Fig. 162
- Ajouter la fonction *animate_idle* comme présenté en Fig. 163

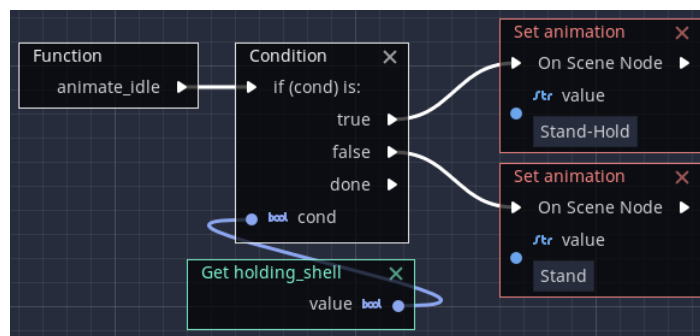


Fig. 163 – Fonction *animate_idle*.

- Ajouter les fonctions *animate_walk_left* et *animate_walk_right* comme présenté en Fig. 164

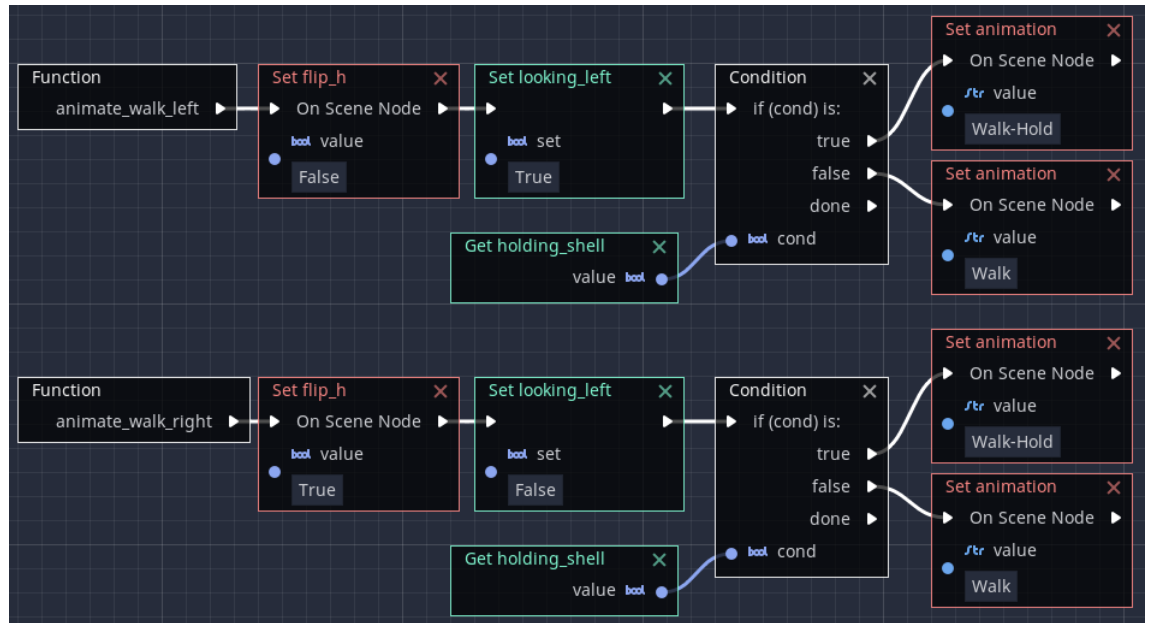


Fig. 164 – Fonctions *animate_walk_left* et *animate_walk_right*.

- Ajouter les signaux sur *RightCatchArea* et *BodyArea* vers *Mario.vs* comme présenté en Fig. 165

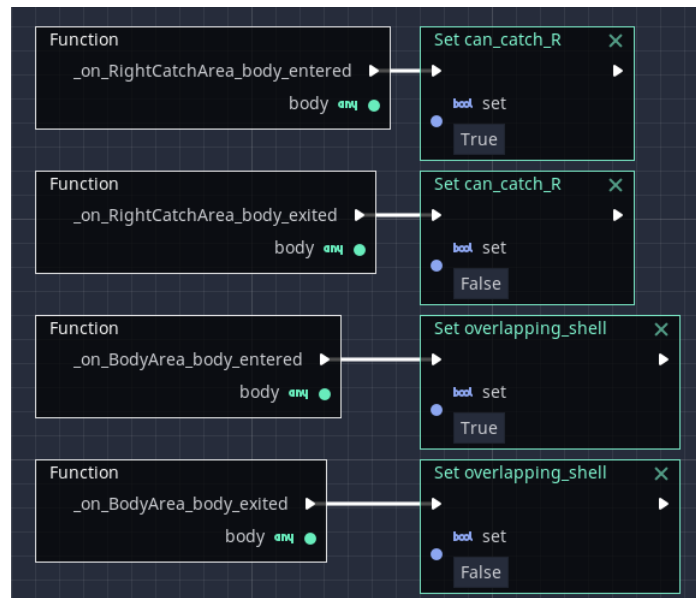


Fig. 165 – Fonctions *body_entered* et *body_exited*.

— Ajouter la fonction *hold_shell* comme présenté en Fig. 166

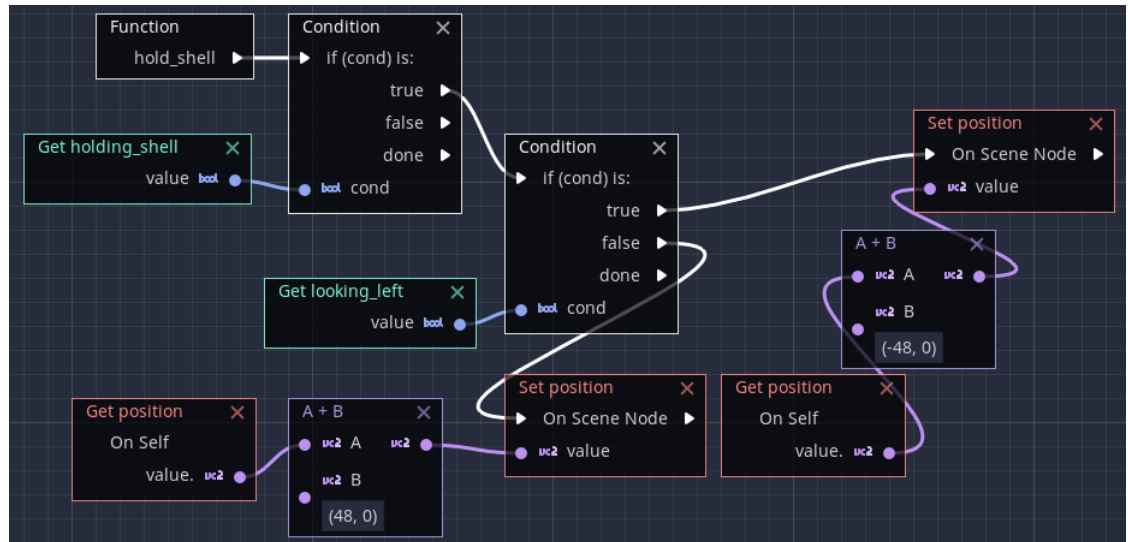


Fig. 166 – Fonction *hold_shell*.

— Ajouter la fonction *_physics_process* dans *Main.vs* comme présenté en Fig. 167

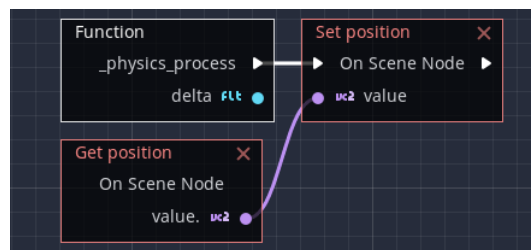


Fig. 167 – Fonction *_physics_process* de *Main.vs*; gestion de la caméra en fonction des déplacements de *Mario*.

— Ajouter la fonction *get_input* comme présenté en Fig. 168, 169 et 170

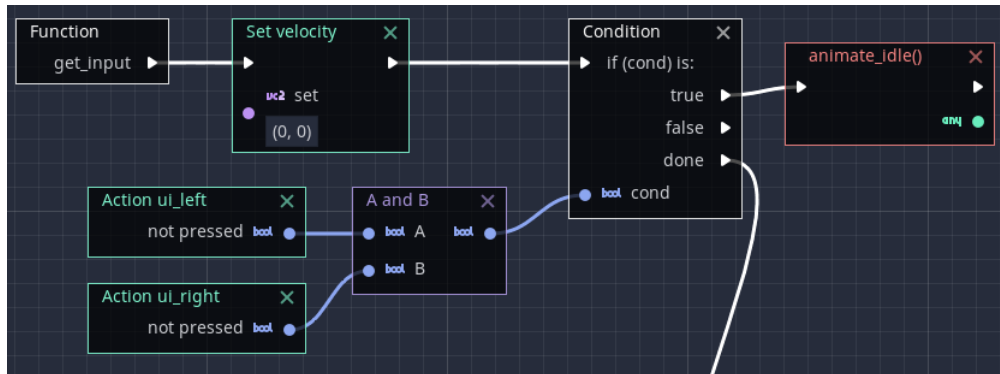


Fig. 168 – Début de la fonction *get_input*; gestion de l'absence de déplacement gauche-droite.

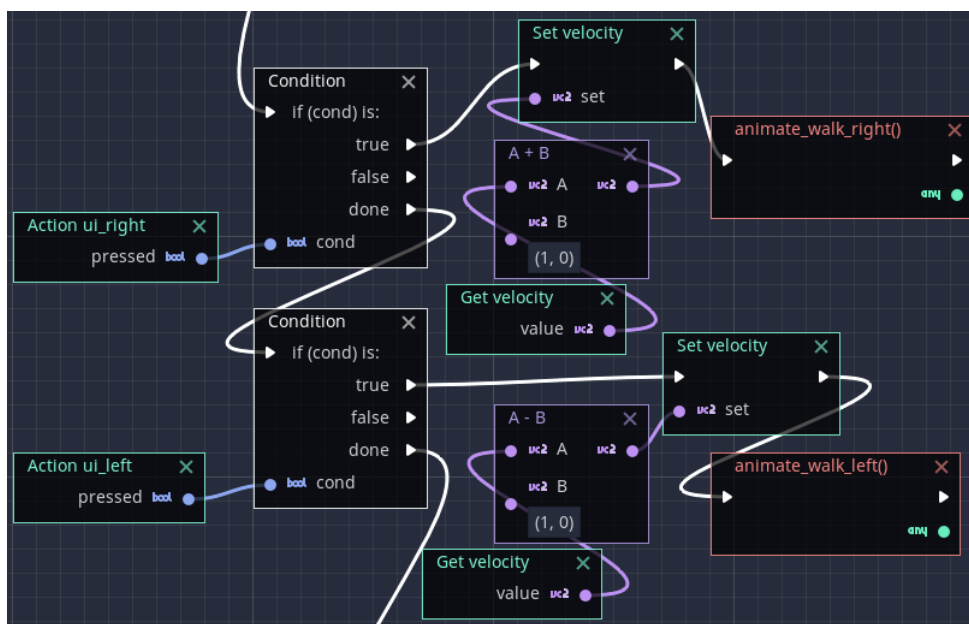




Fig. 169 – Suite de la fonction *get_input*; gestion des déplacements avec  et .

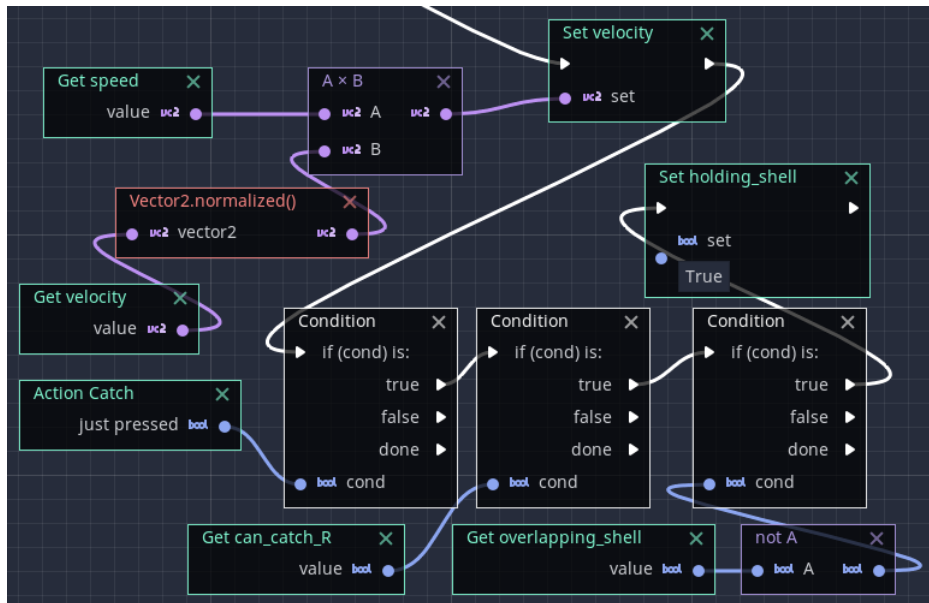


Fig. 170 – Fin de la fonction *get_input*; normalisation de *velocity* de *Mario* et gestion de la saisie avec `a`.

— Ajouter la fonction *_physics_process* dans *Mario.vs* comme présenté en Fig. 171

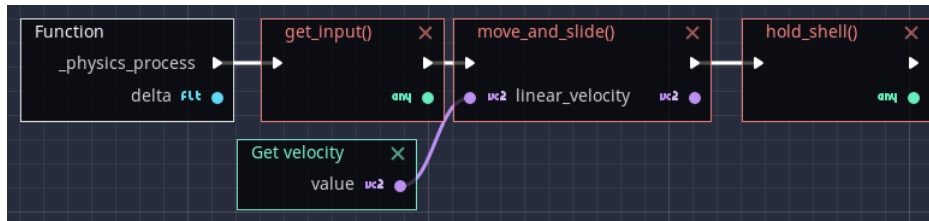


Fig. 171 – Fonction *_physics_process* de *Mario.vs*.

— Exécuter et appuyer sur `a` pour saisir la carapace comme le présente la Fig. 157

4.6 Lancer une carapace de Koopa


On ajoute une action à la scène de la section 4.5 : avec  *Mario* lance la carapace comme le présente la Fig. 172 ; pour supprimer de la scène les objets qui sortent de l'écran et qui ne reviendront pas, on utilise la fonction *queue_free* ; pour savoir si un objet sort de l'écran, on utilise un noeud de type *VisibilityNotifier2D*.



Fig. 172 – Mario lance une carapace de Koopa.

Comme le présente l'arborescence de la Fig. 173, on ajoute à la carapace :

- Un noeud de type *VisibilityNotifier2D*
- Un *visual script* *Shell.vs* pour gérer le déplacement de la carapace et sa suppression de la scène

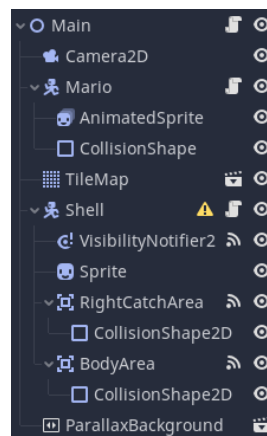


Fig. 173 – Arborescence de la scène.

La Fig. 174 présente les variables de Mario .vs; la variable booléenne `throwing_shell` indique si la carapace doit être lancée.

La Fig. 175 présente les fonctions et les variables de Shell .vs.

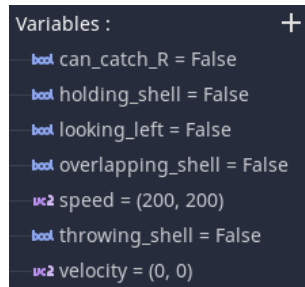


Fig. 174 – Variables de Mario .vs.

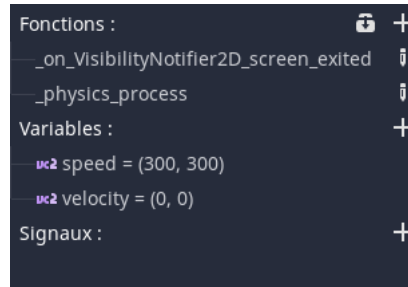


Fig. 175 – Shell .vs.

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

- Ajouter les *assets*, les *visual script* et les fichiers *.tscn* pour les noeuds Mario, Shell, Tilemap et ParallaxBackground de la section 4.5
 - Dans le noeud Main de type *Node2D*, instancier les noeuds Mario, Shell, Tilemap, ParallaxBackground et ajouter un noeud Camera2D
 - Cocher le champ *Current* de la caméra
 - Définir la *tilemap* pour obtenir la plateforme de la Fig. 172
 - Ajouter dans les paramètres du projet, la prise en compte des actions associées à `a` et `z` (en les nommant Catch et Throw)
 - Remplacer les signaux de RightCatchArea et BodyArea vers Mario .vs
 - Ajouter Shell .vs et les variables *speed* et *velocity* pour la carapace
 - Ajouter la fonction *_physics_process* dans Shell .vs comme présenté en Fig. 176
 - Ajouter un noeud de type *VisibilityNotifier2D* au noeud Shell
 - Ajouter une fonction *_on_VisibilityNotifier2D_screen_exited* dans Shell .vs comme présenté en Fig. 176
- [Quand la carapace sort de l'écran, on appelle *queue_free* pour retirer la carapace des objets de la scène]

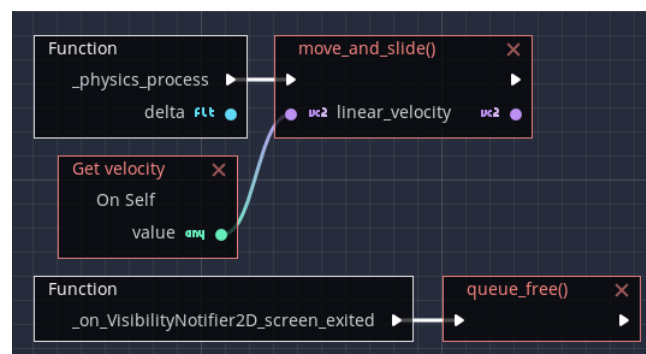


Fig. 176 – Shell .vs.

- Ajouter la variable *throwing_shell* dans Mario .vs comme présenté en Fig. 174

- Ajouter la prise en compte de la touche clavier à la fin de la fonction *get_input* comme présenté en Fig. 177

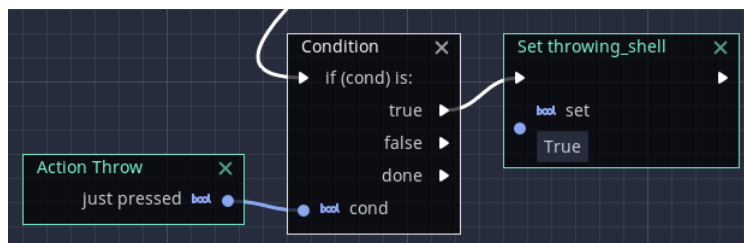


Fig. 177 – Fin de fonction *get_input_end*.

- Ajouter la prise en compte de la variable *throwing_shell* à la fin de la fonction *hold_shell* comme présenté en Fig. 178
[Les fonctions *Get* et *Set* sont ici appliquées aux variables *speed* et *velocity* de *Shell.vs*]

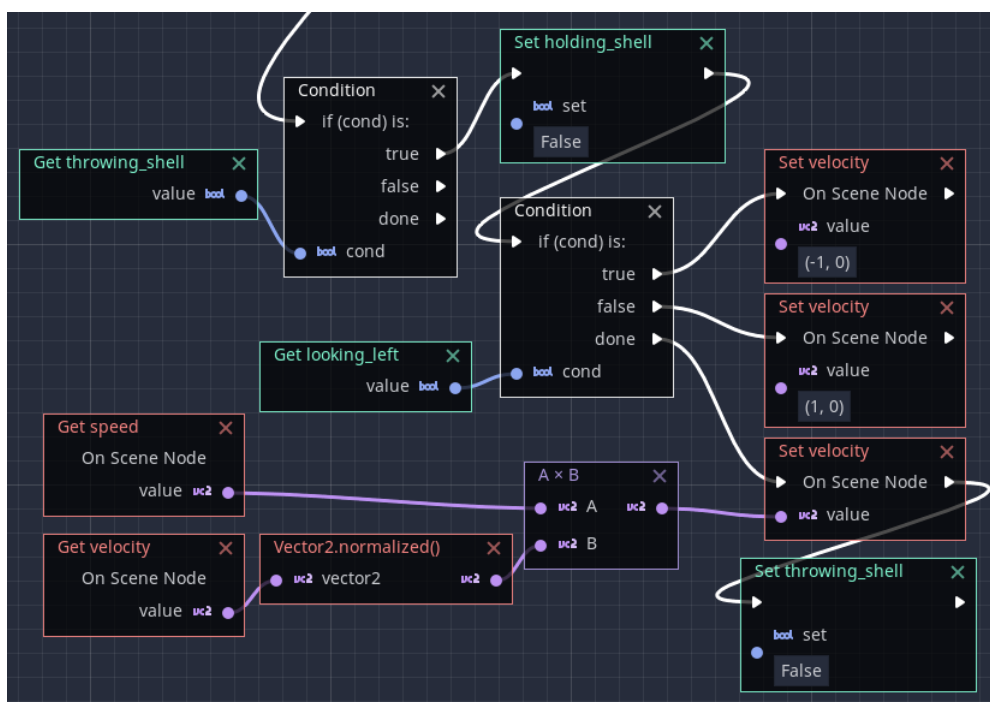


Fig. 178 – Fin de fonction *hold_shell*.

- Exécuter, ramasser avec et lancer la carapace avec comme le présente la Fig. 172
[Vérifier le bon comportement de la disparition de la carapace en la lançant vers la gauche et vers la droite]

4.7 Lancer des *fireball*


On ajoute une action à la scène de la section 2.10 : avec  le Yoshi lance une *fireball* comme le présente la Fig. 179; lancer des objets non précédemment définis dans la scène implique de les instancier avec la fonction `PackedScene.instance` et de les ajouter dans la scène avec la fonction `add_child`; quand la *fireball* sort de l'écran, on la supprime avec la fonction `queue_free`.



Fig. 179 – Mario sur un Yoshi lance une *fireball*.

Fig. 180 présente l'arborescence de la scène principale; on retrouve les mêmes noeuds que dans la scène de la section 2.10; la Fig. 181 présente les fonctions et les variables de la scène principale correspondant à l'animation de Mario sur un Yoshi; la variable `dragon_velocity` définit la direction de chaque nouvelle *fireball*: pour $(1, 0)$, le Yoshi lance une *fireball* vers la droite; pour $(-1, 0)$, le Yoshi lance une *fireball* vers la gauche; elle est initialisée à $(1, 0)$ conformément à l'orientation initiale du Yoshi.

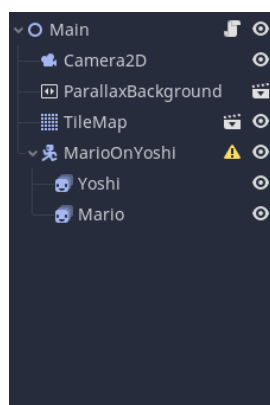


Fig. 180 – Arborescence.

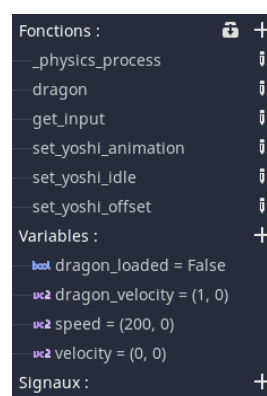


Fig. 181 – Variables et fonctions de `Main`. vs.

Fig. 182 présente l'arborescence de la scène correspondant à une *fireball*; la Fig. 183 présente les fonctions et les variables de cette scène.

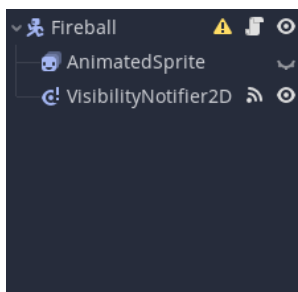


Fig. 182 – Arborescence d'une *fireball*.

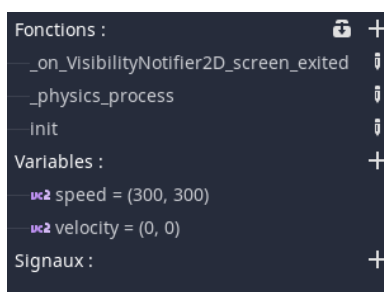


Fig. 183 – Fonctions et variables d'une *fireball*.

Pour réaliser la scène `Fireball.tscn`, suivre la procédure ci-dessous :

- Dans le noeud Main de type *Node2D*, ajouter les noeuds *AnimatedSprite* et *VisibilityNotifier2D*
- Créer un nouveau *SpriteFrame* dans le champ *Frames* de *AnimatedSprite* [Définir l'animation *Idle* comme présenté en Fig. 184]

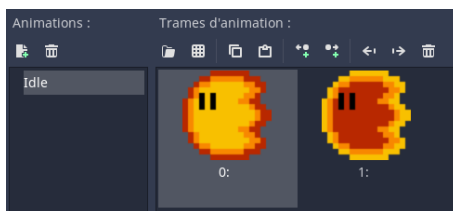


Fig. 184 – Animation *Idle* de *fireball*.

- Définir les variables `speed` et `velocity` comme présenté en Fig. 183
- Associer au noeud Main un *visual script* `Fireball.vs`
- Définir la fonction *init* comme présenté en Fig. 185 et 186

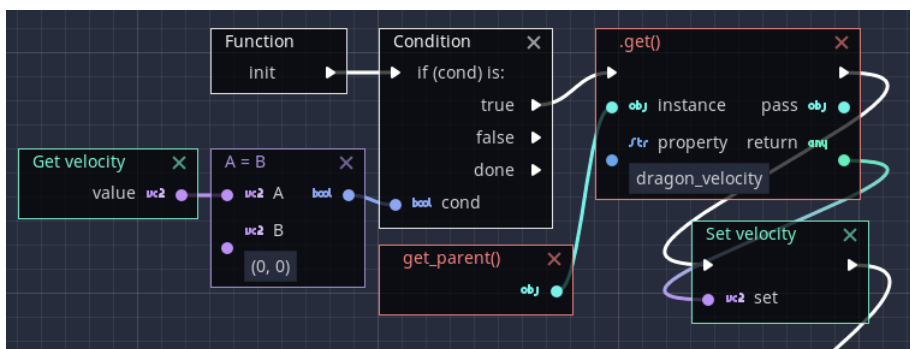


Fig. 185 – Début de fonction *init* de `Fireball.vs`; on récupère l'orientation de la *fireball* dans la variable `dragon_velocity` du noeud parent.

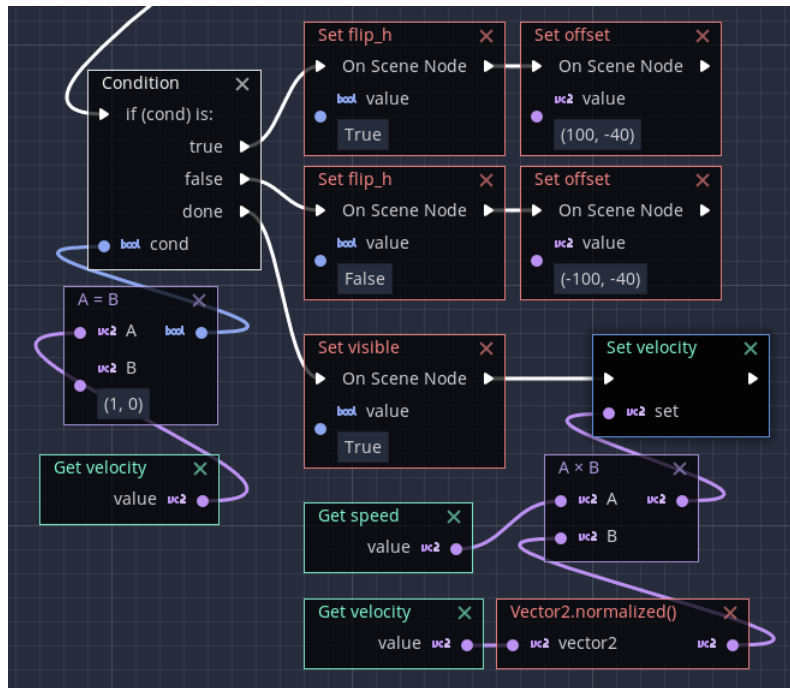


Fig. 186 – Fin de fonction *init* de *Fireball.vs*; selon la variable *velocity*, la fireball est orientée vers la gauche ou vers la droite; son *offset* initial est également dépendant de sa direction.

— Définir les fonctions *_physics_process* et *_on_VisibilityNotifier2D_screen_exited* comme présenté en Fig. 187

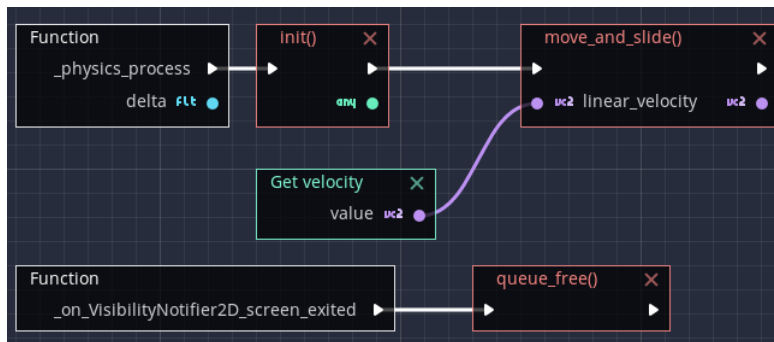


Fig. 187 – Déplacement et suppression de la *fireball*.

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

- Ajouter les *assets*, le *visual script* `Main.vs` et les fichiers `.tscn` pour les noeuds `MarioOnYoshi`, `Tilemap` et `ParallaxBackground` de la section 2.10
- Dans le noeud `Main` de type *Node2D*, instancier les noeuds `MarioOnYoshi`, `Tilemap`, `ParallaxBackground` et ajouter un noeud `Camera2D`
- Cocher le champ *Current* de la caméra
- Définir la *tilemap* pour obtenir la plateforme de la Fig. 179
- Ajouter dans les paramètres du projet, la prise en compte des actions associées à `a` et `z` comme dans la section 2.10
- Ajouter la mise à jour de la variable `dragon_velocity` dans la fonction `get_input` comme présenté en Fig. 188
[On modifie la fonction présentée en Fig. 91 et 92 présentées à la section 2.10]
- Ajouter l'appel de la fonction `dragon` à la fin de la fonction `get_input` comme présenté en Fig. 189
[On modifie la fonction présentée en Fig. 92 à la section 2.10]



Fig. 188 – Mise à jour de `Dragon_velocity` dans `Main.vs`.

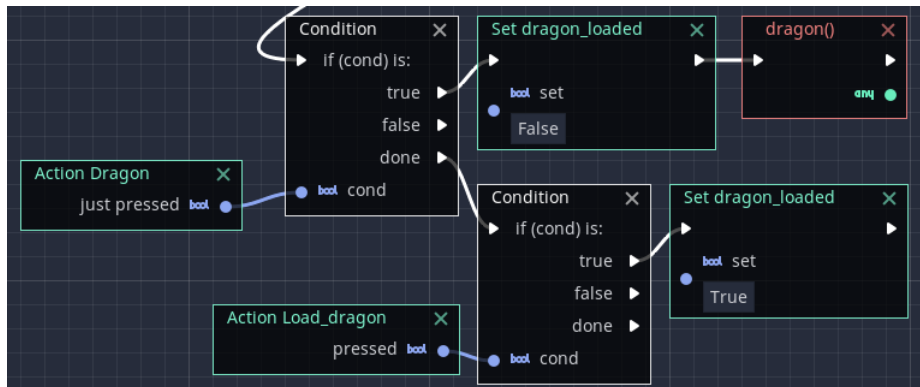


Fig. 189 – Appel de la fonction *dragon* dans *Main.vs*.

- Définir la fonction *dragon* comme présenté en Fig. 190
- [Charger la scène *Fireball.tscn* par glisser-déposer dans *Main.vs*]
- [Ajouter l'appel à *PackedScene.instance* en sélectionnant l'objet *Fireball.tscn*]
- [Placer le noeud créé dans l'arborescence avec *add_child*]
- [Placer le noeud créé dans la scène avec *Set position*]
- [Dans les deux cas, on utilise l'objet retourné par *PackedScene.instance*]
- [Le *Get position* correspond à la position de *Mario* sur le *Yoshi*]
- [Le décalage de la *fireball* par rapport au *Mario* sur le *Yoshi* est défini avec un *offset* qui varie selon la direction de *Mario* présenté en Fig. 188]

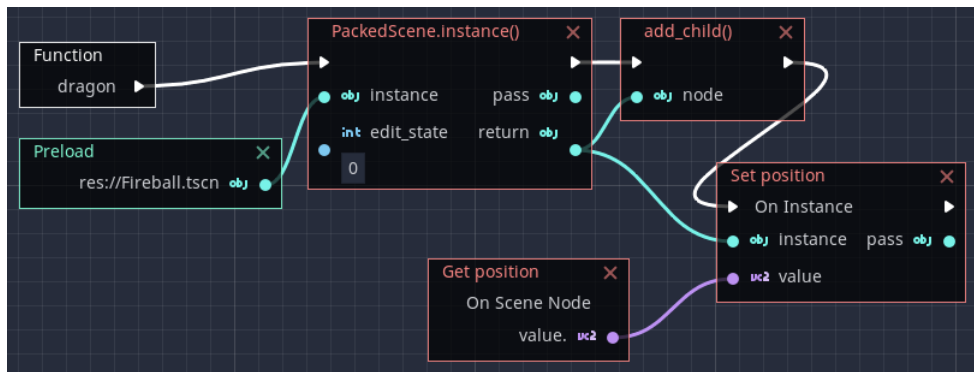


Fig. 190 – Fonction *dragon* dans *Main.vs*.

- Exécuter et lancer des *fireball* dans les deux directions comme présenté en Fig. 191



Fig. 191 – *Mario* sur un *Yoshi* lance des *fireball*.

4.8 Utiliser les *layer* et les *mask de collision*

On place *Mario* dans une *tilemap* avec une grille comme le présente la Fig. 192; *Mario* se déplace sur la grille dans toutes les directions; les bordures rouges de la grille l'empêche de descendre de la grille; une *ZigZag-Lava-Bubble* apparait toutes les 2 secondes en haut à droite avec un angle initial de déplacement vers la gauche; les *ZigZag-Lava-Bubble* entrent en collision avec les murs et rebondissent dessus; quand une *ZigZag-Lava-Bubble* entre en collision avec *Mario*, *Mario* clignote pendant quelques secondes.

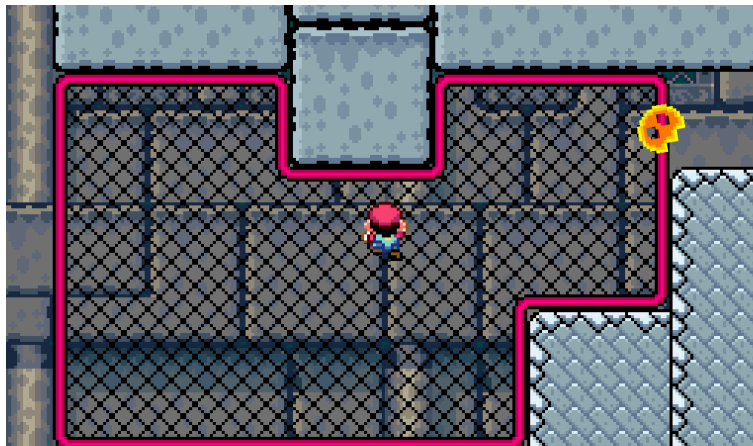


Fig. 192 – *Mario* sur une grille avec une *ZigZag-Lava-Bubble* à sa droite.

Fig. 193 présente l'arborescence de la scène principale; le *Timer* crée des *ZigZag-Lava-Bubble* toutes les 2 secondes; la Fig. 194 présente les fonctions et les variables de la scène principale correspondant au déplacement de la caméra selon les déplacements de *Mario* sur la grille.



Fig. 193 – Arborescence.

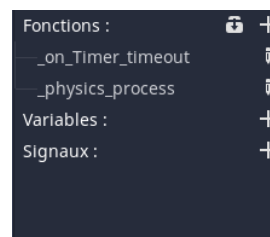


Fig. 194 – Variables et fonctions de *Main*. vs.

La création de *ZigZag-Lava-Bubble* par la fonction `_On_Timer_timeout` implique la définition de la scène *Zig-Zag-Lava-Bubble.tscn* dont la Fig. 195 présente l'arborescence et la Fig. 196 présente les fonctions et les variables.

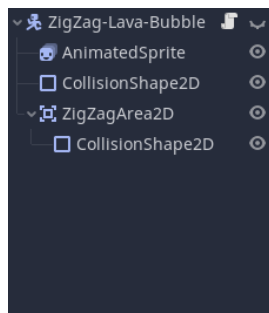


Fig. 195 – Arborescence.

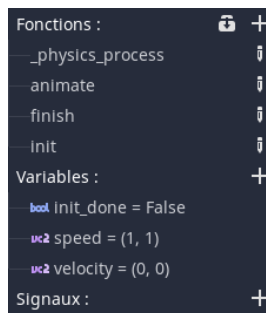


Fig. 196 – Variables et fonctions.

Pour réaliser la scène *ZigZag-Lava-Bubble.tscn*, suivre la procédure ci-dessous :

- Dans le noeud *ZigZag-Lava-Bubble* de type *Node2D*, ajouter les noeuds *AnimatedSprite* et *CollisionShape2D*
- Créer un nouveau *SpriteFrame* dans le champ *Frames* de *AnimatedSprite* [Définir l'animation *Idle* comme présenté en Fig. 197]

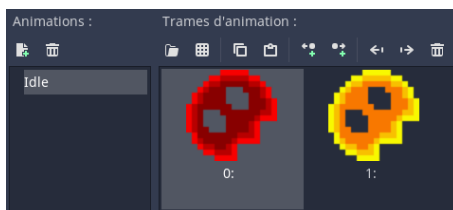


Fig. 197 – Animation Idle de *ZigZag-Lava-Bubble*.

- Définir une *CircleShape2D* dans le champ *Shape* de *CollisionShape2D* [Définir sur l'interface une forme correspondant au volume englobant des images de l'animation *Idle* comme présenté en Fig. 198] [Après l'alignement des formes sur l'interface, on désélectionne le champ *Visible* du noeud *ZigZag-Lava-Bubble* et les formes disparaissent de l'interface] [La *ZigZag-Lava-Bubble* est invisible tant qu'elle n'est pas à sa position initiale]

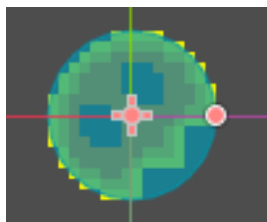


Fig. 198 – *CircleShape2D* de *ZigZag-Lava-Bubble*.

- Ajouter les noeuds ZigZagArea2D de type *Area2D* et *CollisionShape2D*
- Définir une nouvelle *CircleShape2D* dans le champ *Shape* de *CollisionShape2D*
[En toute logique, cette forme est similaire à la précédente]
- Définir les variables *init_done*, *speed* et *velocity* comme présenté en Fig. 196
- Associer au noeud ZigZag-Lava-Bubble un *visual script* ZigZag-Lava-Bubble.vs
- Définir la fonction *init* comme présentée en Fig. 199
[La position (400, -100) place les *ZigZag-Lava-bubble* en haut à droite]
[La *ZigZag-Lava-bubble* est invisible avant d'être à sa position initiale]
[Initialisé à (1, 1), *velocity* a un angle de 45 degrés; en appliquant une rotation entre 0.78 et 3.92 radians, on obtient un angle initial entre 90 et 270 degrés]

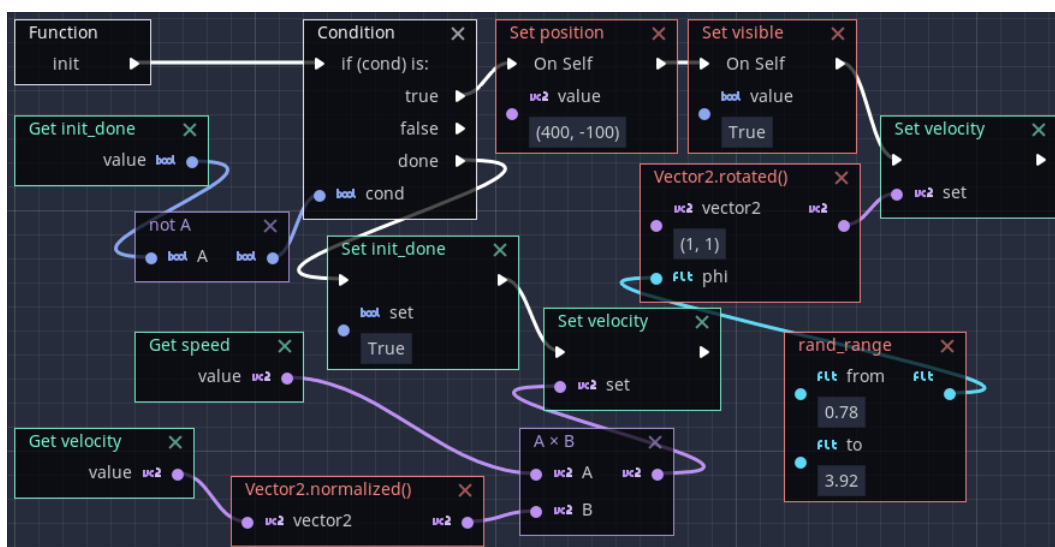


Fig. 199 – Fonction *init* de ZigZag-Lava-Bubble.vs.

- Définir la fonction *animate* comme présentée en Fig. 200
[La fonction *Vector2.angle* retourne une valeur en radians]
[Les radians multipliés par 180 et divisés par π deviennent des degrés]
[Ajouter 135 permet d'obtenir une *ZigZag-Lava-bubble* orientée vers la droite]

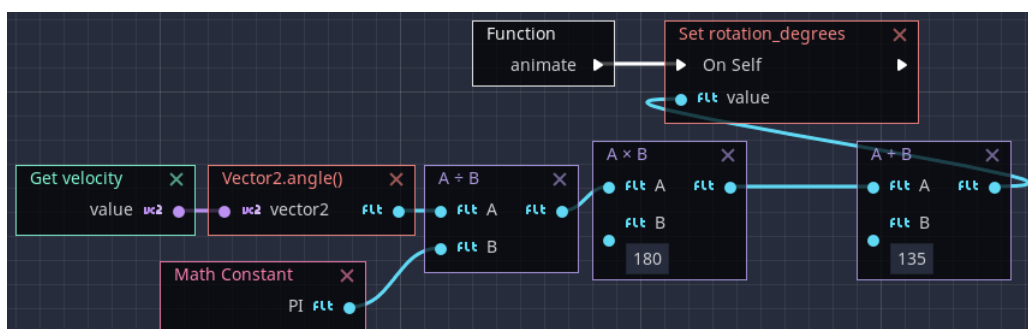


Fig. 200 – Fonction *animate* de ZigZag-Lava-Bubble.vs.

- Définir la fonction *finish* comme présentée en Fig. 201
[Quand une *ZigZag-Lava-Bubble* est trop loin de la coordonnées (0,0), elle est supprimée de la scène; la distance associée est fixée à 1000]

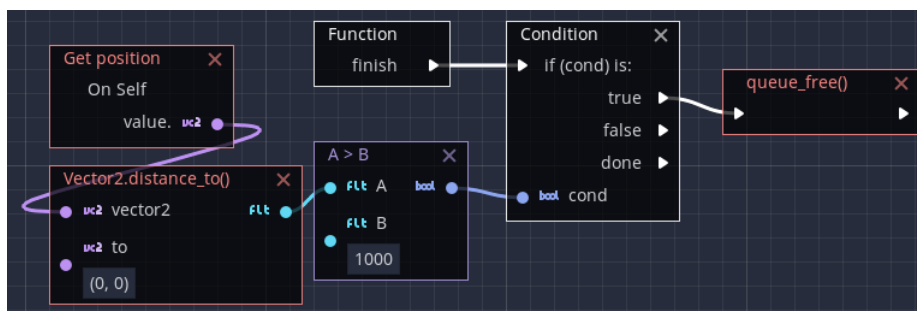


Fig. 201 – Fonction *finish* de *ZigZag-Lava-Bubble*.vs.

- Définir la fonction *_physics_process* comme présentée en Fig. 202
[En cas de collision, la fonction *move_and_collide* retourne l'objet en collision et on applique à *velocity* une nouvelle valeur correspondant au rebond selon la normale du point de collision]
[En l'absence de collision, *move_and_collide* retourne null]
[Avec ou sans collision, *_physics_process* se termine par *animate* et *finish*]

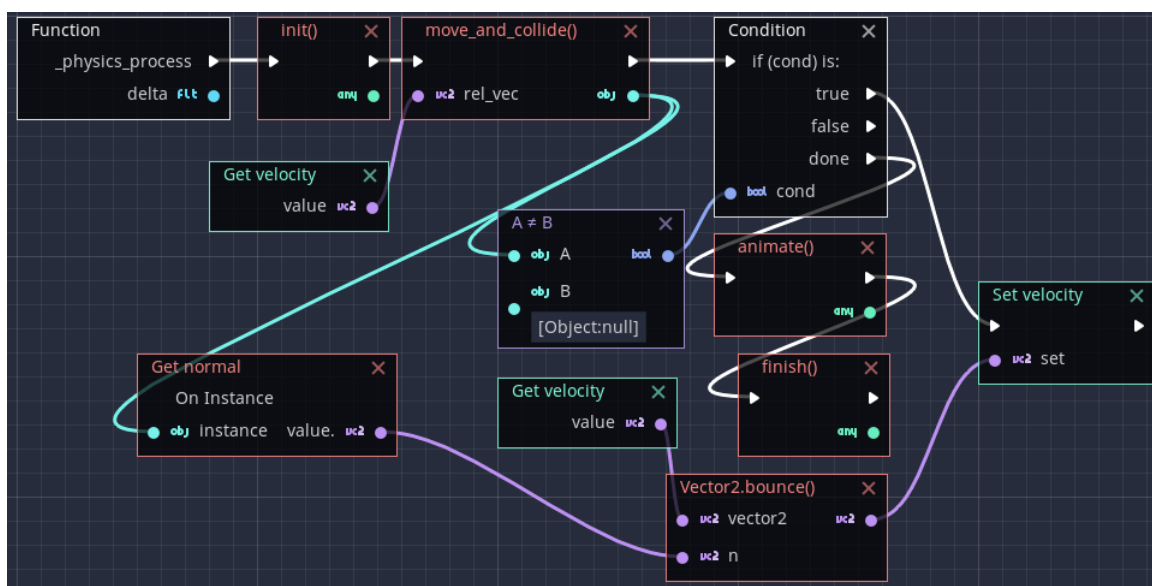


Fig. 202 – Fonction *_physics_process* de *ZigZag-Lava-Bubble*.vs.

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

- Ajouter les *assets*, le *visual script* `Main.vs` et le noeud `Camera2D`
- Cocher le champ *Current* de la caméra
- Définir une scène `ParallaxBackground.tscn`
 - Ajouter les noeuds `ParallaxBackground` et `ParallaxLayer`
 - [Dans le champ *Mirroring*, fixer les valeurs de *x* et *y* à 2048 et 1168]
 - [Dans le champ *Scale*, fixer la valeur à 0.5]
 - Ajouter un noeud `Sprite` en lui associant l'image présentée en Fig. 203
- Ajouter un noeud `ParallaxBackground` dans la scène principale
- Définir une scène `Grid.tscn`
 - Ajouter un noeud `Grid` de type *Tilemap*
 - [On définit les tuiles présentées en Fig. 204]
 - [On définit avec ces tuiles la *tilemap* présentée en Fig. 205]

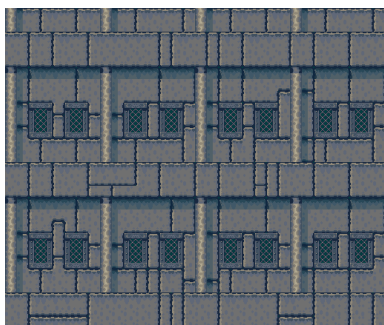


Fig. 203 – Background.

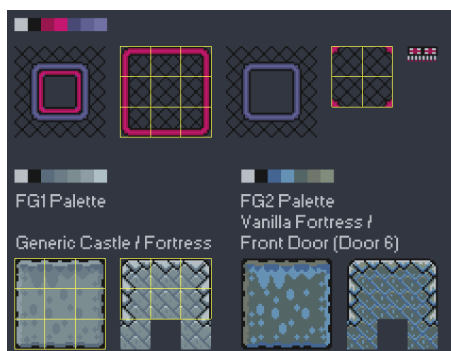


Fig. 204 – Tuiles utilisées.

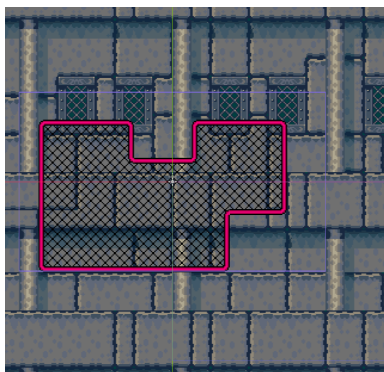


Fig. 205 – Tilemap de la grille.

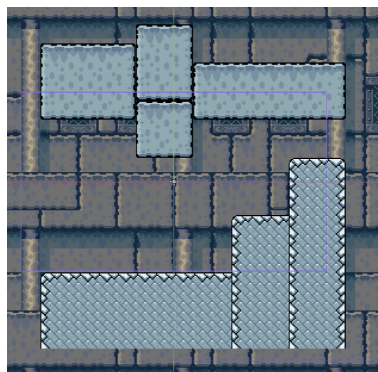


Fig. 206 – Tilemap des murs.

- Ajouter un noeud `Grid` dans la scène principale
- Définir une scène `Walls.tscn`
 - Ajouter un noeud `Walls` de type *Tilemap*
 - [On réutilise les tuiles présentées en Fig. 204 avec lesquelles on définit la *tilemap* présentée en Fig. 206]
- Ajouter un noeud `Walls` dans la scène principale

Fig. 207 présente l'arborescence de la scène correspondant à *Mario* quand il est sur une grille; *Mario* se déplace ici dans toutes les directions; la Fig. 208 présente les fonctions et les variables de cette scène.



Fig. 207 – Arborescence.

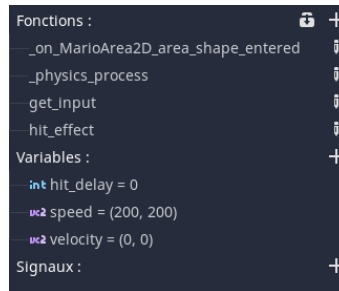


Fig. 208 – Variables et fonctions.

Pour réaliser la scène `MarioOnGrid.tscn`, suivre la procédure ci-dessous :

- Dans le noeud `MarioOnGrid` de type *Node2D*, ajouter les noeuds `AnimatedSprite` et `CollisionShape2D`
- Créer un nouveau *SpriteFrame* dans le champ *Frames* de `AnimatedSprite` [Définir l'animation *Climb* comme présenté en Fig. 209]
[Fixer le champ *Vitesse* à 10 IPS pour augmenter la vitesse d'enchainement des images dans l'animation]

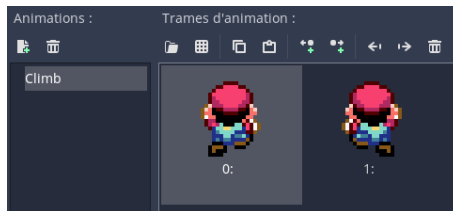


Fig. 209 – Animation Climb de *Mario*.

- Définir un *RectangleShape2D* dans le champ *Shape* de `CollisionShape2D`
- Ajouter un noeud `MarioArea2D` de type *Area2D*
- Ajouter un `CollisionShape2D` dans `MarioArea2D` avec une forme *RectangleShape2D* dans son champ *Shape*
- Ajouter les variables `hit_delay`, `speed` et `velocity` comme présenté en Fig. 208
[Quand `hit_delay` est supérieure stricte à 0, *Mario* clignote pour indiquer qu'il a été touché par un objet; l'effet de clignotement est créé par alternance du champs *visible* à `True` et `False`; quand `hit_delay` est égale à 0, *Mario* est visible et ne clignote pas]
- Déclarer les quatre fonctions présentées en Fig. 208

- Définir la fonction `_physics_process` comme présentée en Fig. 212
`[get_input` définit `velocity` en fonction des touches clavier pressées]
`[move_and_slide` déplace `Mario` en fonction de `velocity`]
`[hit_effect` ajoute un effet de clignotement en fonction des collisions]

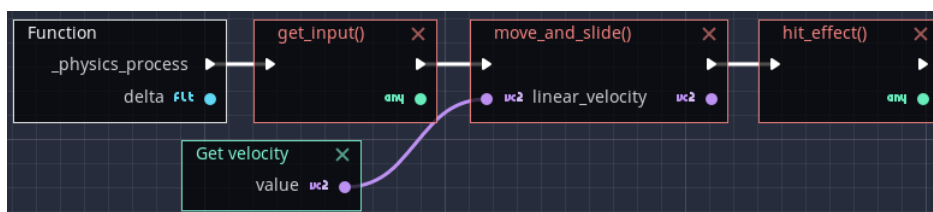


Fig. 212 – Fonction `_physics_process` de `MarioOnGrid.tscn`.

- Définir la fonction `get_input` comme le présente les Fig. 213, 214 et 215
`[Si aucune des touches ←, →, ↓, ↑ n'est pressée, on désactive l'animation Cimb; si une de ces touches est pressée, l'animation est active]`

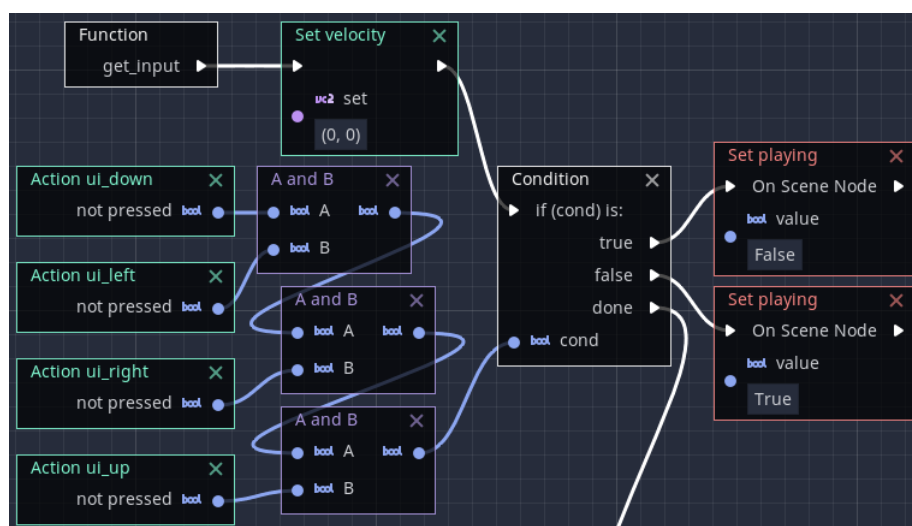


Fig. 213 – Début de la fonction `get_input`.

[On ajuste velocity en fonction des touches pressées]

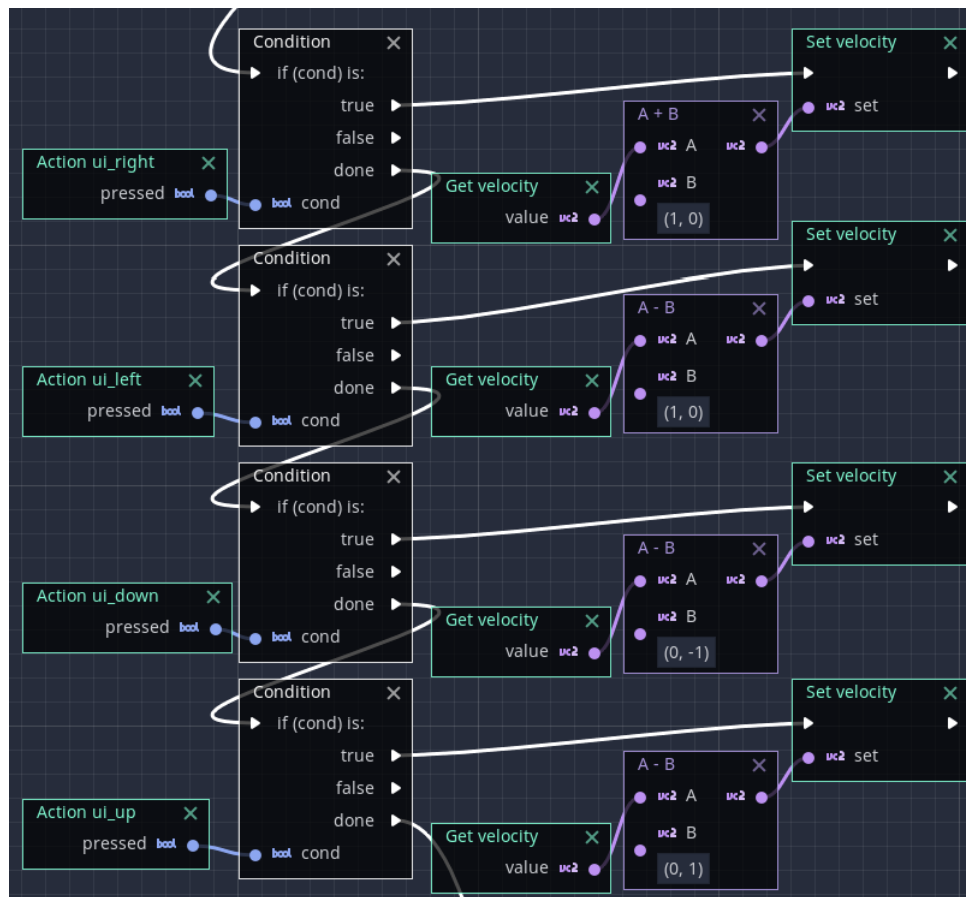


Fig. 214 – Suite de la fonction *get_input*.

[On normalise le vecteur velocity]

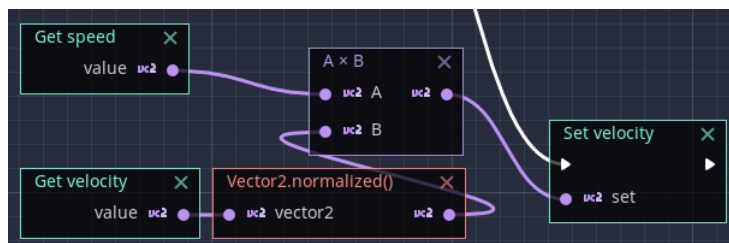


Fig. 215 – Fin de la fonction *get_input*.

Pour terminer de réaliser la scène principale de cette interface, suivre la procédure ci-dessous :

- Ajouter un noeud Mario de type *MarioOnGrid*
- Ajouter un noeud Timer de type *Timer*
[Cocher *Autostart* dans l'inspecteur]
[Fixer le champ *Wait Time* à 2]
- Définir les fonctions *_physics_process* et *_on_Timer_timeout* comme le présente la Fig. 216
[*_physics_process* ajuste la position de la caméra sur la position de *Mario*]
[*_on_Timer_timeout* instancie une *ZigZag_Lava_Bubble* toutes les 2 secondes]

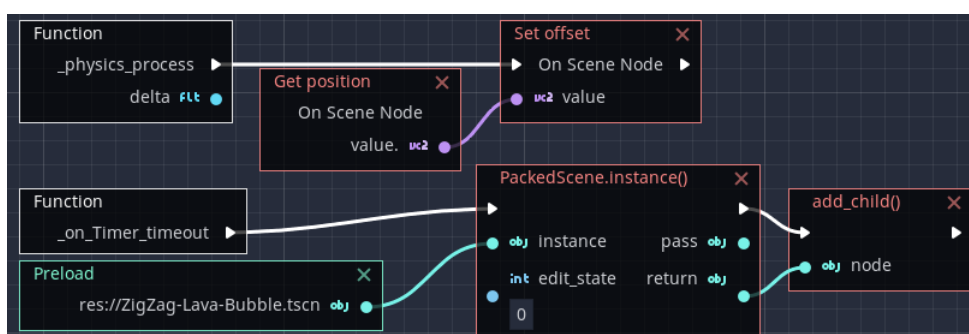


Fig. 216 – Fonctions *_Physics_process* et *_on_Timer_timeout*.

- Pour différencier les collisions, on définit les *layer* et *mask* de collision présentés en Fig. 217, 218, 219 et 220
- *Mario* entre en collision avec les bords de la grille
- Les *ZigZag-Lava-Bubble* entrent en collision avec les murs
[Les Fig. 221 et 222 présentent les formes de collision de deux tuiles de la *tilemap* *Grid* et les Fig. 223 et 224 pour deux tuiles de la *tilemap* *Walls*]

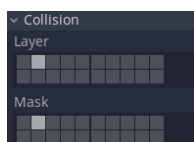


Fig. 217 – Grid.

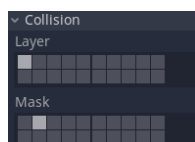


Fig. 218 – Mario.

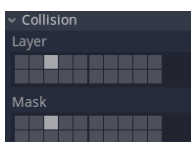


Fig. 219 – Walls.

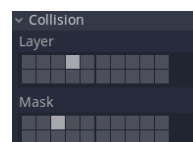


Fig. 220 – Bubble.

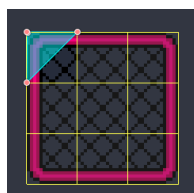


Fig. 221 – Coin.

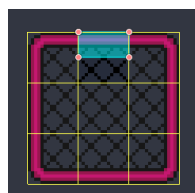


Fig. 222 – Bord.

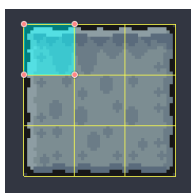


Fig. 223 – Coin.

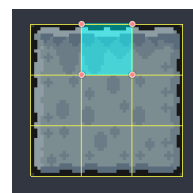


Fig. 224 – Bord.

[Fig. 225 et 226 présentent les zones de collision de l'ensemble des formes de la grille et des murs; ces zones de collisions sont présentées en transparence bleue]

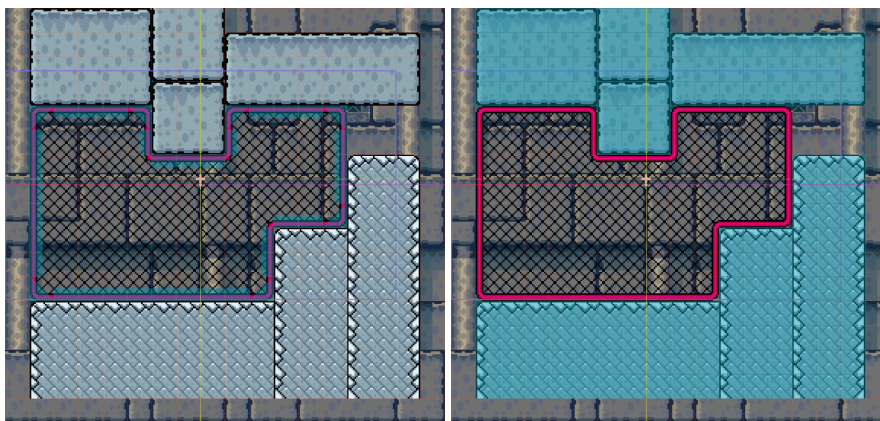


Fig. 225 – Zones de collision de Grid. Fig. 226 – Zones de collision de Walls.

- Exécuter permet d'obtenir la Fig. 227 après quelques secondes
 [Les *ZigZag-Lava-Bubble* rebondissent sur les murs]
 [Les mouvements de *Mario* sont limités par les bords rouges de la grille]
 [Quand *Mario* entre en collision avec une *ZigZag-Lava-Bubble*, il clignote quelques secondes]

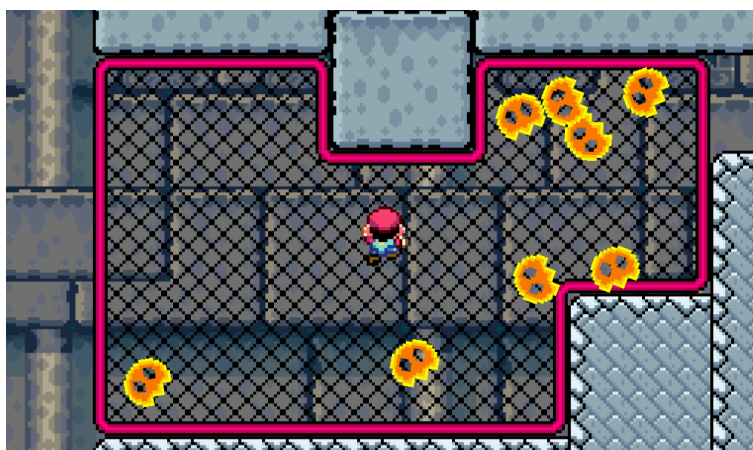


Fig. 227 – *Mario* évitant des *ZigZag-Lava-Bubble* sur une grille.