# *AnimationControllerFx*

## Table des matières

## I – Définition

**AnimationControllerFx** est un module permettant de contrôler de façon dynamique les différentes animations que possède un objet. Son rôle principale est de permettre à l'utilisateur de pourvoir interagir dynamiquement entre le module et l'objet en question.
**NB :** Ce module est compatible à un jeu 2D, 3D et n'est pas sauvegardable. Notez que ce module ne supporte pas le noeud *AnimationTreePlayer*.

## II – Les propriétés disponibles

+ **NodePath Animator :** Contient la référence d'un animateur. Les types supportés sont : *AnimationTree* et *AnimationPlayer*.

+ **NodePath Target :** Contient la référence du noeud sur lequel le module fera ses traitements. Le type que supporte cette propriété est le *KinematicBody*.

+ **bool RootMotion = false :** Voulez-vous déplacer le noeud ciblé à partir des transformations récupérées sur le squelette d'animation ?

+ **float Gravity = 9.8 :** Quel est l'effet de pésenteur qui sera appliquer au noeud pris pour cible ?

## III – Les méthodes disponibles

+ **float get_total_state_length () :** Renvoie le temps total des animations prises en charge par le développeur.

+ **float get_max_state_length () :** Renvoie le temps de l'animation de plus grand parmit les animations chargées.

+ **float get_min_state_length () :** Renvoie le temps de l'animation de plus petit parmit les animations chargées.

+ **PoolStringArray get_not_running_states_names () :** Renvoie les noms de toutes les animations qui ne sont pas en cours de lecture.

+ **int get_progress_pourcentage () :** Renvoie en pourcentage, la progression en temps réel de l'animation en cours de lecture. L'ensemble des valeurs qui seront renvoyées appartiendra à l'intervalle [0 ; 100]

+ **float get_normalized_progress () :** Renvoie une valeur normalizée de la progression en temps réel de l'animation en cours de lecture. L'ensemble des valeurs qui seront renvoyées appartiendra à l'intervalle [0.0 ; 1.0].

+ **void play_state (name, queued, priority, count, speed, blend, seek, delay = 0.0) :** Joue une animation donnée en fonction des configurations prévues à son égard. Notez que cette méthode n'agit que si l'animateur est de type *AnimationPlayer*.
  » **String name :** Contient le nom de l'animation à joué.
  » **bool queued :** Voulez-vous mettre l'animation à jouée dans une file d'attente ? Par défaut, cet paramètre est inactif.
  » **int count :** Combien de fois l'animation sera jouer ? Par défaut, l'animation sera jouée une seule fois seulement. Notez qu'une valeur négative engendrera une lecture infinie.
  » **bool priority :** Met en priorité l'animation à jouée. La lecture d'une animation en priorité peut affecté ou non une animation déjà en cours de lecture. Par défaut, aucune priorité n'est attribuée à l'animation en question.
  » **float speed :** Contient la vitesse de lecture de l'animation ? Par défaut, la valeur de ce paramètre est sur *1.0*.
  » **float blend :** Voulez-vous mettre une transition entre l'animation actuellement en cours de lecture et la nouvelle animation à lire ? Notez que cet paramètre peut faire office de transition simple lorsque la lecture de l'animation est faite en utilisant le paramètre *seek*. Par défaut, la valeur de ce paramètre est sur *-1.0*.
  » **float seek :** Souhatez-vous, vous déplacer dans l'animation à travers la position du lecteur ? Par défaut, la valeur de ce paramètre est sur *-1.0*.
  » **float delay :** Quel est le temps mort avant le lancement de l'animation ?

+ **void queue_deletion (id, update = true, delay = 0.0) :** Supprime un ou plusieurs file(s) d'attente(s) de la mémoire. Notez que cette méthode n'agit que si l'animateur est de type *AnimationPlayer*.
  » **String | PoolStringArray id :** Contient le(s) nom(s) de(s) animation(s) en file d'éttente.
  » **bool update :** Voulez-vous que les modifications affectent l'animation en cours de lecture ?
  » **float delay :** Quel est le temps mort avant le lancement de l'animation ?

+ **void clear_queue (update = true, delay = 0.0) :** Supprime toutes les files d'attentes de la mémoire. Notez que cette méthode n'agit que si l'animateur est de type *AnimationPlayer*.
  » **bool update :** Voulez-vous que les modifications affectent l'animation en cours de lecture ?
  » **float delay :** Quel est le temps mort avant le lancement de l'animation ?

+ **void run_state (source, value, reversed, fade, time, type, easing, delay = 0.0) :** Modifie la valeur d'une propriété à partir de la source donnée. Cette méthode n'agit que si l'animateur est de type *AnimationPlayer*.
  » **String source :** Quel est le chemin pointant vers la propriété à changée. Notez que vous n'avez plus bésoin de mettre le préfix *"parameters/"* avant de préciser l'élément que vous voulez ciblé. Exemple : *"TimeScale/scale"* au lieu de *"parameters/ TimeScale/scale"*.
  » **Variant value :** Quelle est la nouvelle valeur de la propriété pris pour cible ?
  » **bool reversed :** Voulez-vous modifier les valeurs dans l'ordre inverse où elles sont venue ? Par défaut, la modification des valeurs s'éffectue de façon normale.
  » **bool fade :** Aimeriez-vous effectuer une transition lorsque la valeur de la propriété ciblé changera ? Par défaut, aucune transition n'est appliquée.
  » **float time :** La transition de valeur s'effectuera en combien de temps ? Par défaut, la valeur de ce paramètre est sur *0.0*.
  » **int type :** Quel type de transition souhaitez-vous utiliser ? Les constantes issues de ce paramètre sont celles définient aux sein de Godot. Par défaut, la valeur de ce paramètre est sur *0*.
  » **int easing :** Quel assouplissement désirez-vous utiliser pour faire la transition. Les constantes issues de ce paramètre sont celles définient aux sein de Godot. Par défaut, la valeur de ce paramètre est sur *2*.
  » **float delay :** Quel est le temps mort avant le changement de valeur ?

+ **void combo (states, ref, start, level, source = '', blend = -1.0, delay = 0.0) :** Gère les différents appuies répétés conçut pour déclencher des actions ou états.
  » **String | PoolStringArray states :** Quels sont/est le(s) état(s) qui sera/seront exécuté(s) au fure et à mesure que le joueur progressera dans les appuies répétés ?
  » **String | PoolStringArray ref :** De quelle(s) animation(s), les appuies répétés devront commencés ?
  » **String start :** Sur quelle animation revenir lorsque tous les états auront été exécutés ou lorsque l'animation en cours d'exécution ne correspond à l'état de référence ?
  » **int level :** Quel est le niveau d'appuie de la touche prise dans une série d'appuies répétés ?
  » **String source :** Quel est le chemin pointant vers la propriété à changée. Notez que vous n'avez plus bésoin de mettre le préfix *"parameters/"* avant de préciser l'élément que vous voulez ciblé. Exemple : *"TimeScale/scale"* au lieu de *"parameters/ TimeScale/scale"*. Le noeud doit être uniquement de type *AnimationNodeTransition*. C'est obligatoire et important de le faire ainsi. La valeur de ce paramètre est inutile lorsque l'animateur est de type *AnimationPlayer*.
  » **float blend :** Voulez-vous mettre une transition entre l'animation actuellement en cours de lecture et la nouvelle animation à lire ? Cette option est à utilisée uniquement lorsque l'animateur est de type *AnimationPlayer*.
  » **float delay :** Quel est le temps mort avant l'exécution des états prévu ?

+ **Dictionary get_queue_data (json = true) :** Renvoie les données de toutes les animations misent dans une file d'attente.
  » **bool json :** Voulez-vous renvoyer le résultat sous le format json ?

## IV – Les événements disponibles

+ **queue_changed (node) :** Signal déclenché lorsque des modifications sont éffectuées dans la file d'attente.
  » **Node node** : Contient le noeud où cet signal a été émit.

+ **state_started (data) :** Signal déclenché au démarrage de la lecture d'un état. Cet événement renvoie un dictionaire contenant les clés suivantes :
  » **Node node** : Contient le noeud où cet signal a été émit.
  » **String name** : Contient le nom de l'animation en question.

+ **state_finished (data) :** Signal déclenché à la fin de la lecture d'un état. Cet événement renvoie un dictionaire contenant les clés suivantes :
  » **Node node** : Contient le noeud où cet signal a été émit.
  » **String name** : Contient le nom de l'animation en question.

+ **state_changed (data) :** Signal déclenché lorsqu'un état a été changé par un autre. Cet événement renvoie un dictionaire contenant les clés suivantes :
  » **Node node** : Contient le noeud où cet signal a été émit.
  » **String old** : Contient le nom de l'ancienne l'animation .
  » **String new** : Contient le nom de la nouvelle animation en question.