Table des matières

I – Définition

InventoryFx est un module conçut pour la gestion des inventaires dans un jeu vidéo. Le module intègre un support brute ainsi qu'un support graphique permettant de gérer en temps réel l'évolution de l'inventaire.
NB : Ce module est compatible à un jeu 2D, 3D et est sauvegardable.

II – Les propriétés disponibles

+ int Mode = 0 : Quel mode de l'inventaire voulez-vous utiliser ? Les valeurs possibles sont :
-> **InventoryFx.Model.HYBRID ou 0** : Mode sans interface graphique (hybride ou brute).
-> **InventoryFx.Model.UI ou 1** : Mode avec interface graphique.

+ NodePath ListenModule : Contient l'instance d'un module du même trampe de ce module. Exemple : *AnimatorFx, MenuFx, AudioTrackFx* etc... En d'autres termes, contient la référence d'un module possèdant également un curseur. Le but de cette fonctionnalité est de syncnrôniser la valeur du curseur du module référé à celle de ce module.

+ int PocketCount = 0 : Contient le nombre total de place à allouée par l'inventaire en mémoire. Cette propriété ne prend son utilitée que si l'inventaire passe en mode hybride. La valeur de cette propriété est dans l'intervalle [0 ; ++].

+ Texture EmptySymbol : Quelle texture devrons nous utilisée si une poche de l'inventaire se retrouve vidée ? Cette propriété ne prend son utilitée que si l'inventaire passe en mode interface utilisateur.

+ bool AllowDragAndDrop = false : Voulez-vous permettre les opérations du glisser-déposser sur l'inventaire ? Cette propriété ne prend son utilitée que si l'inventaire passe en mode interface utilisateur.

+ NodePath DragDropStyle : Contient l'instance d'un noeud de type *Sprite* ou *TextureRect*. Ce champ utilise les configurations éffectuées sur le noeud récupéré pour mettre à jour le style qu'utilisera le module pour éffectuer l'opération. Cette propriété ne prend son utilitée que si l'inventaire passe en mode interface utilisateur.

+ Array Varieties : Tableau de dictionnaires contenant les différentes configurations de chaque variété d'objet prise en charge par le développeur. Les clés que supportent les dictionnaires sont :

» **String name** : Contient le nom de la variété en question. L'utilisation de cette clé est obligatoire.

» **String symbol** : Contient le chemin pointant vers l'image représentant la variété. L'utilisation de cette clé est obligatoire. Notez que cette clé ne s'utilise que lorsque l'inventaire est en mode interface graphique.

» **Vector2 size = Vector2 (50, 50)** : Contrôle la résolution du symbol de la variété. Notez que cette clé ne s'utilise que lorsque l'inventaire est en mode interface graphique.

» **int quality = 2** : Contrôle la qualité du symbol de la variété. Notez que cette clé ne s'utilise que lorsque l'inventaire est en mode interface graphique. Les valeurs possibles sont celles définient au sein de classe *Image* de Godot.

» **bool uncastable = false** : Devons nous rendre injetable cette variété ?

» **bool maskstock = false** : Devons cacher le stock à affiché dans l'inventaire graphique ? Notez que cette clé ne s'utilise que lorsque l'inventaire est en mode interface graphique.

» **PoolStringArray combination** : Quelles sont les noms des variétés qu'il faut combiner pour obtenir cette variété ? La taille de cette tableau ne doit pas dépasser deux éléments. Notez que les variétés que vous donnerai doit déjà être définit dans l'inventaire.

» **Array levels** : Contient les différentes valeurs que doit vérifiée la valeur contenu dans la clé *stock*. Les dictionnaires issus de ce tableau suppotent les clés suivantes :
  » **int value** : Contient une des valeurs de la clé *stock*.
  » **Array actions** : Que se passera t-il lorsque la valeur du stock de cette variété atteint celle indiquée ? Cette clé contient tous les configurations relatives à un ou plusieurs flux d'exécution(s). L'utilisation de cette clé est déjà décrite au niveau des bases du framework. Précisement le sujet portant sur l'utilisation de la propriété *EventsBindings* (la section des actions d'un événement).

» **int stock = 0** : Quel est le stock maximale de cette variété ?

+ Array Pockets : Tableau de dictionnaires contenant les différentes configurations de chaque poche disponible sur l'inventaire. Cette propriété ne prend son utilitée que si l'inventaire passe en mode interface utilisateur. Les clés que supportent les dictionnaires sont :

» **String | NodePath pocket** : Contient l'instance d'un noeud de type *Sprite* ou *TextureRect*. Cette clé permettra au développeur de pouvoir spécifier les poches graphiques de son inventaire. Dans ce cas, l'inventaire est une interface utilisateur préconstruite par le développeur. L'utilisation de cette clé est obligatoire.

» **String | NodePath label** : Contient l'instance d'un noeud de type *Label*. Cette clé permettra au développeur de pouvoir spécifier les textes graphiques de son inventaire. Dans ce cas, l'inventaire est une interface utilisateur préconstruite par le développeur. L'utilisation de cette clé est obligatoire si le libelé spécifié n'est pas un descendant directe de l'instance de la poche renseignée.

» **bool disabled = false** : Devons nous considérer cette poche comme étant une espace disponible au sein de l'inventaire ? Notez que les poches désactivées ne seront jamais prises en par le module.

NB : Les répétitions au niveau des noms des variétés ne seront pas tolérées. Gardez à l'esprit que ce module se sert d'un curseur pour sélectionner les poches définient par le développeur. Ce curseur n'est rien d'autre que l'index de position de chaque configuration d'élément. Par défaut, sa valeur est **0** lorsqu'il y a un ou plusieurs élément(s) configuré(s) sur le module et **-1** si aucun(s) élément n'est disponible sur le module.

III – Les méthodes disponibles

+ **bool is_empty () :** L'inventaire est-il vide ?

+ **bool is_filled () :** L'inventaire est-il remplit ?

+ **void add_objects (data, reversed = false, delay = 0.0) :** Ajoute des objets dans l'inventaire.
  » **Array data :** Tableau de dictionaire contenant les différentes variétés ainsi que leur stock. Les dictionnaires supportent les clés suivantes :
    » **String | int id :** Contient le nom ou l'index de position de la variété à ciblée.
    » **int stock = 1 :** Quelle est la valeur à ajoutée au stock de cette variété ?
  » **bool reversed :** Doit-on renversé l'ordre d'ajout des objets dans l'inventaire ?
  » **float delay :** Quel est le temps mort avant le(s) ajout(s) ?

+ **void remove_objects (data, reversed = false, delay = 0.0) :** Supprime des objets de l'inventaire.
  » **Array data :** Tableau de dictionaire contenant les différentes variétés ainsi que leur stock. Les dictionnaires supportent les clés suivantes :
    » **String | int id :** Contient le nom ou l'index de position de la variété à ciblée.
    » **int stock = 1 :** Quelle est la valeur à enlevée au stock de cette variété ?
  » **bool reversed :** Doit-on renversé l'ordre de suppression des objets dans l'inventaire ?
  » **float delay :** Quel est le temps mort avant le(s) suppression(s) ?

+ **void clear (id = null, delay = 0.0) :** Détruit complètement une variété de l'inventaire. Si aucune variété n'a été précisé, L'inventaire complet sera nétoyé.
  » **String | PoolStringArray | PoolIntArray | int id :** Quelle(s) variété(s) voulez-vous détruire ?
  » **float delay :** Quel est le temps mort avant le(s) destruction(s) ?

+ **int get_current_stock (id = null, pocket_index = -1, on_config = false) :** Renvoie le stock actuel d'une variété. Notez que si aucune paramètre n'est précisée, le stock total de l'inventaire sera renvoyé. Si la variété a été précisée, le stock total de cette dernière sera renvoyé. Dans le cas contraire, le stock contenu dans la poche précisée sera renvoyé.
  » **String | int id :** Contient le nom ou l'index de position de la variété en question.
  » **String | int pocket_index :** Contient l'index de position d'une poche quelconque.

» **bool on_config** : Doit-on utilisé les configurations éffectuées au niveau du module pour donner le résultat ? Par défaut, le module utilise les valeurs issues de la progression de l'inventaire pour donner le résultat final.

+ **void refresh (delay = 0.0) :** Réactualise l'inventaire.
» **float delay** : Quel est le temps mort avant le rafraîchissement ?

+ **int combine (first, second, delay = 0.0) :** Effectue la combinaison de deux vériétés différentes pour donner une autre variété. Cette méthode utilise la clé *combination* du champ *Varieties* pour mener à bien son travail.
» **String | int first** : Contient le nom ou l'index de position de la première variété à combinée.
» **String | int second** : Contient le nom ou l'index de position de la deuxième variété à combinée.
» **float delay** : Quel est le temps mort avant la combinaison ?

+ **Dictionary get_data (json = true) :** Renvoie les données sur la gestion des variétés de l'inventaire.
» **bool json** : Voulez-vous renvoyer le résultat sous le format json ?

+ **int | PoolIntArray get_varieties_index (id) :** Renvoie le(s) index de position d'une ou de plusieurs variété(s) en fonction de(s) identifiant d'une ou de plusieurs poche(s).
» **int | PoolStringArray | PoolIntArray | String id** : Contient le(s) index de position d'une ou de plusieurs poche(s) quelconque(s). Notez que si vous mettez le(s) nom(s) d'une ou de plusieurs variété(s) définit dans le module, son/leur index de position sera renvoyé.

+ **void cancel_operation (delay = 0.0) :** Annule une opération en cours d'exécution. Notez que l'annulation écrase également les progressions éffectuées par l'opération en question avant son arrêt. Cette méthode ne travail que si l'inventaire passe en mode interface utilisateur.
» **float delay** : Quel est le temps mort avant l'annulation ?

+ **void confirm_operation (delay = 0.0) :** Valide une opération en cours d'exécution. Notez que la validation savegarde également les progressions éffectuées par l'opération en question avant son arrêt. Cette méthode ne travail que si l'inventaire passe en mode interface utilisateur.
» **float delay** : Quel est le temps mort avant la validation ?

+ **int rotate_pocket (index, angle = 90.0, delay = 0.0) :** Tourne une poche graphique de l'inventaire. Cette méthode ne travail que si l'inventaire passe en mode interface utilisateur.
» **int index** : Contient l'index de position de la poche à ciblée.
» **float angle** : Quel est l'angle de rotation de la poche ?
» **float delay** : Quel est le temps mort avant la rotation ?

+ **int | PoolIntArray get_pockets_index (variety_id) :** Renvoie le(s) index de position d'une ou plusieurs poche(s) en fonction de l'identifiant d'une variété.
  » **int | String | PoolStringArray | PoolIntArray variety_id** : Contient le(s) nom(s) ou le(s) index de position d'une ou de plusieurs variété(s) quelconque(s).

+ **int get_cursor_position () :** Renvoie la position du sélecteur de l'inventaire. Les valeurs des positions du curseur sera égale à celles des poches disponibles au sein de l'inventaire. Cette méthode ne travail que si l'inventaire passe en mode interface utilisateur.

+ **void set_cursor_position (new_value, delay = 0.0) :** Change la position du sélecteur de l'inventaire. Les valeurs doivent être dans l'intervalle [0 ; (NombreTotalDePocheDisponible - 1)]. Cette méthode ne travail que si l'inventaire passe en mode interface utilisateur.
  » **int | String new_value** : Contient la nouvelle valeur du sélecteur.
  » **float delay** : Quel est le temps mort avant le changement de valeur ?

+ **int | PoolIntArray get_empty_pockets_index (variety_id) :** Renvoie le(s) index de position d'une ou plusieurs poche(s) vide(s).

+ **int | PoolIntArray get_busy_pockets_index (variety_id) :** Renvoie le(s) index de position d'une ou plusieurs poche(s) occupée(s).

+ **int | PoolIntArray get_uncastable_varieties_index (on_config = false) :** Renvoie le(s) index de position de la ou des variété(s) injetable(s).
  » **bool on_config** : Doit-on utilisé les configurations éffectuées au niveau du module pour donner le résultat ? Par défaut, le module utilise les valeurs issues de la progression de l'inventaire pour donner le résultat final.

+ **int | PoolIntArray get_combinable_varieties_index (on_config = false) :** Renvoie le(s) index de position de la ou des variété(s) combinable(s).
  » **bool on_config** : Doit-on utilisé les configurations éffectuées au niveau du module pour donner le résultat ? Par défaut, le module utilise les valeurs issues de la progression de l'inventaire pour donner le résultat final.

+ **int get_big_stock (on_config = false) :** Renvoie la valeur du stock le plus grand dans l'inventaire.
  » **bool on_config** : Doit-on utilisé les configurations éffectuées au niveau du module pour donner le résultat ? Par défaut, le module utilise les valeurs issues de la progression de l'inventaire pour donner le résultat final.

+ **int get_tiny_stock (on_config = false) :** Renvoie la valeur du stock le plus petit dans l'inventaire.
  » **bool on_config** : Doit-on utilisé les configurations éffectuées au niveau du module pour donner le résultat ? Par défaut, le module utilise les valeurs issues de la progression de l'inventaire pour donner le résultat final.

+ **void run_operation (operation, delay = 0.0) :** Exécute une opération de l'inventaire. Cette méthode ne travail que si l'inventaire passe en mode interface utilisateur.
  » **int operation** : Contient l'opération à exécutée. Les valeurs possibles sont :
    -> **InventoryFx.Operation.IDLE ou 0** : Mode normal de l'inventaire.
    -> **InventoryFx.Operation.STATIC_MOVING ou 1** : Mode déplacement statique d'une variété de l'inventaire.
    -> **InventoryFx.Operation.DYNAMIC_MOVING ou 2** : Mode déplacement dynamique d'une variété de l'inventaire.
    -> **InventoryFx.COMBINATION ou 3** : Mode combinaison de variétés.
  » **float delay** : Quel est le temps mort avant l'exécution ?

+ **void bdi_run_operation (operation, delay = 0.0) :** Exécute une opération de l'inventaire en mode bidirectionelle. Cette méthode ne travail que si l'inventaire passe en mode interface utilisateur.
  » **int operation** : Contient l'opération à exécutée. Les valeurs possibles sont :
    -> **InventoryFx.Operation.IDLE ou 0** : Mode normal de l'inventaire.
    -> **InventoryFx.Operation.STATIC_MOVING ou 1** : Mode déplacement statique d'une variété de l'inventaire.
    -> **InventoryFx.Operation.DYNAMIC_MOVING ou 2** : Mode déplacement dynamique d'une variété de l'inventaire.
    -> **InventoryFx.COMBINATION ou 3** : Mode combinaison de variétés.
  » **float delay** : Quel est le temps mort avant l'exécution ?

IV – Les événements disponibles

+ **inventory_empty (node) :** Signal déclenché lorsque l'inventaire est vide.
  » **Node node** : Contient le noeud où ce signal a été émit.

+ **inventory_filled (node) :** Signal déclenché lorsque l'inventaire est remplit.
  » **Node node** : Contient le noeud où ce signal a été émit.

+ **inventory_updated (node) :** Signal déclenché lorsque l'inventaire est mise à jour.
  » **Node node** : Contient le noeud où ce signal a été émit.

+ **operation_started (data) :** Signal déclenché au démarrage d'une opération de l'inventaire. Ce événement renvoie un dictionaire contenant les clés suivantes :
  » **Node node** : Contient le noeud où ce signal a été émit.
  » **int | PoolIntArray accepted** : Contient le(s) index de position de la ou des poche(s) vérifiant les conditions que pose l'opération en question.
  » **int | PoolIntArray denied** : Contient le(s) index de position de la ou des poche(s) ne vérifiant pas les conditions que pose l'opération en question.

+ **operation_running (data) :** Signal déclenché tanqu'une opération sera en cours d'exécution. Ce événement renvoie un dictionaire contenant les clés suivantes :
  » **Node node** : Contient le noeud où ce signal a été émit.
  » **int operation** : Contient l'opération en cours d'exécution.

+ **operation_stoped (data) :** Signal déclenché lorsqu'on arrête l'exécution d'une opération de l'inventaire. Ce événement renvoie un dictionaire contenant les clés suivantes :
  » **Node node** : Contient le noeud où ce signal a été émit.
  » **int operation** : Contient l'opération qui a été arrêtée.

+ **operation_failed (data) :** Signal déclenché lorsque l'exécution de la tâche principale de l'opération en question a échoué. Ce événement renvoie un dictionaire contenant les clés suivantes :
  » **Node node** : Contient le noeud où ce signal a été émit.
  » **int operation** : Contient l'opération ayant provoquée l'erreur.