Table des matières

I – Définition

SaveLoadFx est un module conçut pour la sauvegarde et le chargement de données dans un jeu. Ce dernier permettra aux développeurs de sauvegarder et de charger facilement leurs données. Cependant, les données du jeu sont déposées dans un fichier configuré par le développeur lui- même.
NB : La sauvegarde et le chargement des données peuvent être sécurisées. Notez également que ce module est de nature indestructible, est compatible à un jeu 2D, 3D et n'est pas sauvegardable.

II – Les propriétés disponibles

+ int TargetPath = 0 : Contient les différents chemins que prend en charge ce module. Ces chemins représentent les endroits possibles où l'on peut déposé le fichier de sauvegarde du jeu. Les valeurs possibles sont celles définient au sein de la méthode *get_os_dir ()* de la classe **MegaAssets**.

+ String Source : Contient le chemin pointant vers le fichier de sauvegarde définit ou à créé par le module au cours des sauvegardes.

+ String ActiveCheckpoint : Contient le nom du point de sauvegarde à ciblé pour la gestion des données du jeu sur le fichier de sauvegarde. Elle représente également le point de sauvegarde actif. Cette option met le focus sur le point de sauvegarde donné parmi plusieurs autres points de sauvegarde dans lequel les sauvegardes et les chargements de données seront éffectuées. Notez que le fichier de sauvegarde peut avoir un ou plusieurs point(s) de sauvegarde. Pour pouvoir sauvegarder et/ou charger les données du jeu, il faudra préciser le point de sauvegarde à ciblé pour effectuer l'opération demandée. Pour tous traitement à faire sur les données du jeu, on solicitera par défaut, le point de sauvegarde actif au niveau du fichier de sauvegarde.

+ int Security = 7 : Contient les différents modes de sécurité possibles que l'on peut utilisé au cours de la sauvegarde des données du jeu. Les valeurs possibles sont celles définient au sein de la méthode *serialize ()* de la classe **MegaAssets**.

+ int Level = 1 : Contient les différents niveaux de sécurité possibles que l'on peut utilisé au cours de la sauvegarde des données du jeu. Les valeurs possibles sont celles définient au sein de la méthode *serialize ()* de la classe **MegaAssets**.

+ int FileChecksum = 1 : Quelle méthode de chiffrement voulez-vous utiliser pour générer le checksum du fichier de sauvegarde ? Les valeurs possibles sont celles définient au sein de la méthode *serialize ()* de la classe **MegaAssets**.

+ String Key : Contient le mot de passe à utilisé pour sécuriser les données du jeu. Le développeur a la possibilité de générer automatiquement un mot de passe via le booléen *GeneratePassword*.

+ bool ScreenCapture = false : Voulez-vous éffectuer une capture d'écran à chaque sauvegarde éffectuée ?

+ **Vector2 Size = Vector2 (64, 64) :** Contrôle la résolution des captures d'écran à générées lors des sauvegardes.

+ **int Quality = 2 :** Contrôle la qualité des captures d'écran à générées lors des sauvegardes. Les valeurs possibles sont celles définient au sein de la classe *Image* de Godot.

+ **int CompressMode = 2 :** Quel mode de compression voulez-vous adopter pour compresser les captures d'écran générées lors des sauvegardes ? Les valeurs possibles sont celles définient au sein de la méthode *get_screen_shot ()* de la classe **MegaAssets**.

+ **int CompressSource = 2 :** A partir de quelle source de compression les compressions des captures d'écran générées lors des sauvegardes seront éffectuées ? Les valeurs possibles sont celles définient au sein de la classe *Image* de Godot.

+ **float CompressRatio = 1000.0 :** Quel taux de compression appliqué aux captures d'écran générées lors des sauvegardes ?

+ **int Format = 6 :** Quel sera le format des captures d'écran à générées lors des sauvegardes ? Les valeurs possibles sont celles définient au sein de la méthode *get_screen_shot ()* de la classe **MegaAssets**.

+ **bool LoadAllData = true :** Voulez-vous charger automatiquement le fichier de sauvegarde à l'initialisation du jeu ?

+ **PoolStringArray TargetScenes :** Contient tous les noms en chaîne de caractères des scènes à ciblées. Cette option, permettra au module de charger ou de recharger le fichier de sauvegarde ciblé par le développeur lorsque l'une des scènes spécifiées par ce dernier sera charger et active.
NB : La répétition du nom d'une ou de plusieurs scène(s) n'est pas tolérée.

+ **bool GeneratePassword = false :** Souhaitez-vous générer automatiquement un mot de passe ? A ce niveau, la valeur du champ *Key* sera mise à jour à chaque génération.

III – Les méthodes disponibles

+ **void generate_password (delay = 0.0) :** Génère au hazard, un mot de passe en fonction des configurations présentent à son niveau. La valeur du champ *Key* sera mise à jour à chaque génération. **Notez que vous n'avez pas la possibilité de générer un mot de passe lorsque le jeu est en cours d'exécution**.
  » **float delay :** Quel est le temps mort avant la génération ?

+ **void save_game_data (checkpoints = null, ignore = null, active_camera = null, delay = 0.0) :**
Enregistre le dictionnaire des données du jeu dans le fichier de sauvegarde.
» **String | PoolStringArray checkpoints** : Quel(s) est/ sont le(s) point(s) de sauvegarde à modifié(s) avant la sauvegarde complète des données du jeu ? La mise à jour que subira les différents points de sauvegarde ne sera rien d'autre que l'insertion des clés spéciales suivantes : *__last_save_date__*, *__last_save_time__*, *__last_load_date__*, *__last_load_time__*, *__game_time__* et *__game_screenshot__* dont se base certaines fonctionnalités du module pour éffectuer leurs traitements. **Notez que si aucun point de sauvegarde n'est précisé, celui actif ne sera pas solicité**.
» **Variant ignore** : Contient la liste des clés à ignorées au cours de la sauvegarde des données. Notez que vous avez la possibilité d'ignorer un point de sauvegarde tout entier. Dans ce cas, toutes ces clés subiront le même sort.
» **Camera active_camera** : Voulez-vous faire une capture d'écran à partir d'une autre caméra avant la sauvegarde générale des données du jeu ?
» **float delay** : Quel est le temps mort avant la sauvegarde des données du jeu ?

+ **void load_game_data (checkpoints = null, ignore = null, delay = 0.0) :** Charge les données du jeu à partir du fichier de sauvegarde créé par la méthode *save_game_data ()*.
» **String | PoolStringArray checkpoints** : Quel(s) est/ sont le(s) point(s) de sauvegarde à modifié(s) après le chargement complet des données du jeu ? Notez que si vous précisez des points de sauvegarde dans ce paramètre, Seule ces derniers seront redéfinit dans le gestionnaire de données. Les autres conserveront leurs données. **Par défaut, toutes les données du gestionnaire sont écrasées par celles chargées à partir du fichier de sauvegarde**.
» **String | PoolStringArray ignore** : Contient la liste des clés à ignorées au cours du chargement des données. Notez que vous avez la possibilité d'ignorer un point de sauvegarde tout entier. Dans ce cas, toutes ces clés subiront le même sort.
» **float delay** : Quel est le temps mort avant le chargement des données du jeu ?

+ **bool is_progress () :** Détermine s'il y a une quelconque progression éffectuée dans le jeu.

+ **String get_root_folders () :** Renvoie les dossiers parents du fichier de sauvegarde.

+ **String get_full_path () :** Renvoie le chemin complet pointant vers le fichier de sauvegarde.

+ **Dictionary get_last_save_date (checkpoint = '') :** Renvoie la date de la dernière sauvegarde éffectuée pour un point de sauvegarde donné. Cette méthode exploite la clé *__last_save_date__* pour accomplir sa tâche. Le dictionaire renvoyé par cette méthode contient les clés suivantes : *year, day, month, weekday* et *dst* (heure d'été).
» **String checkpoint** : Contient le nom du point de sauvegarde à ciblé. Par défaut, le point de sauvegarde actif est utilisé pour faire le traitement demandé.

+ Dictionary **get_last_save_time (checkpoint = '')** : Renvoie le temps de la dernière sauvegarde éffectuée pour un point de sauvegarde donné. Cette méthode exploite la clé *__last_save_time__* pour accomplir sa tâche. Le dictionaire renvoyé par cette méthode contient les clés suivantes : *hour, minute* et *second*.
  » String **checkpoint** : Contient le nom du point de sauvegarde à ciblé. Par défaut, le point de sauvegarde actif est utilisé pour faire le traitement demandé.

+ int **get_game_time (checkpoint = '') :** Renvoie le temps total écoulé toutes les fois où le jeu a été ouvert pour un point de sauvegarde donné. **Notez qu'en cas d'échec, le temps écoulé depuis le démarrage du jeu sera renvoyé**. Cette méthode exploite la clé *__game_time__* pour accomplir sa tâche.
  » String **checkpoint** : Contient le nom du point de sauvegarde à ciblé. Par défaut, le point de sauvegarde actif est utilisé pour faire le traitement demandé.

+ int **get_global_game_time (checkpoint = '') :** Renvoie le temps global écoulé toutes les fois où le jeu a été ouvert pour un point de sauvegarde donné. **Notez qu'en cas d'échec, le temps écoulé depuis le démarrage du jeu sera renvoyé**. Cette méthode exploite la clé *__game_time__* pour accomplir sa tâche.
  » String **checkpoint** : Contient le nom du point de sauvegarde à ciblé. Par défaut, le point de sauvegarde actif est utilisé pour faire le traitement demandé.

+ Dictionary **get_last_load_date (checkpoint = '') :** Renvoie la date du dernier chargement éffectué pour un point de sauvegarde donné. Cette méthode exploite la clé *__last_load_date__* pour accomplir sa tâche. **Notez qu'en cas d'échec, la date du dernier chargement éffectué au niveau du fichier de sauvegarde sera renvoyé**. Le dictionaire renvoyé par cette méthode contient les clés suivantes : *year, day, month, weekday* et *dst* (heure d'été).
  » String **checkpoint** : Contient le nom du point de sauvegarde à ciblé. Par défaut, le point de sauvegarde actif est utilisé pour faire le traitement demandé.

+ Dictionary **get_last_load_time (checkpoint = '')** : Renvoie le temps du dernier chargement éffectué pour un point de sauvegarde donné. Cette méthode exploite la clé *__last_load_time__* pour accomplir sa tâche. **Notez qu'en cas d'échec, le temps du dernier chargement éffectué au niveau du fichier de sauvegarde sera renvoyé**. Le dictionaire renvoyé par cette méthode contient les clés suivantes : *hour, minute* et *second*.
  » String **checkpoint** : Contient le nom du point de sauvegarde à ciblé. Par défaut, le point de sauvegarde actif est utilisé pour faire le traitement demandé.

+ ImageTexture **get_screen_capture (checkpoint = '') :** Renvoie la capture d'écran générée pour un point de sauvegarde donné lors d'une sauvegarde. Cette méthode exploite la clé *__game_screenshot__* pour accomplir sa tâche.

» String **checkpoint** : Contient le nom du point de sauvegarde à ciblé. Par défaut, le point de sauvegarde actif est utilisé pour faire le traitement demandé.

+ void **set_data** **(key, value, checkpoint = ”, delay = 0.0) :** Mets à jour le gestionnaire de données.
  » String **key** : Contient le nom de la clé qui va servir d'identification à la valeur à insérée. **Evitez de mettre des espaces, si vous tenez à récupéré valeur contenu dans votre clé**. Si votre clé n'existe pas, elle sera automatique créé.
  » Variant **value** : Contient la valeur de la clé à ajoutée ou à modifiée.
  » String **checkpoint** : Contient le nom du point de sauvegarde à ciblé. Si le point de sauvegarde n'est pas définit dans le gestionnaire de données, il sera automatiquement créé. Par défaut, le point de sauvegarde actif est pris pour cible.
  » float **delay** : Quel est le temps mort avant la mise à jour du gestionnaire de données ?

+ Variant **get_data** **(key, default = null, checkpoint = ”) :** Renvoie la valeur correspondante à *key*.
  » String **key** : Contient le nom de la clé qui va servir d'identification à la valeur à récupérée.
  » Variant **default** : Quelle valeur va t-on retournée, lorsque le nom de la clé ou celui du point de sauvegarde n'est pas définit dans le gestionnaire de données.
  » String **checkpoint** : Contient le nom du point de sauvegarde à ciblé. Par défaut, le point de sauvegarde actif est pris pour cible.

+ Array **get_checkpoints_list** **() :** Renvoie la liste de tous le(s) point(s) de sauvegarde défini(en)t au sein du gestionnaire de données.

+ bool **has_checkpoints** **(checkpoints) :** Le(s) point(s) de sauvegarde référé(s) est/sont-il(s) défini(en)t au sein du gestionaire de données ?
  » String **checkpoint** : Contient le(s) nom(s) de(s) point(s) de sauvegarde à ciblé(s).

+ void **destroy_keys** **(keys = null, checkpoints = null, delay = 0.0) :** Supprime un ou plusieurs identifiant(s) dans un ou plusieurs point(s) de sauvegarde. Notez que la valeur de(s) identifiant(s) sera également supprimée.
  » String | PoolStringArray **keys** : Contient le(s) identifiant(s) à supprimé(s). **Si aucune clé n'a été référée, on assistera à un nétoyage complet de toutes les données au sein du point de sauvegarde en question**.
  » String | PoolStringArray **checkpoints** : Contient le(s) nom(s) de(s) point(s) de sauvegarde dans lesquel(s) le(s) identifiant(s) précisé(s) sera(ont) supprimé(s). Par défaut, on ciblera le point de sauvegarde actif pour faire le traitement.
  » float **delay** : Quel est le temps mort avant la mise à jour du gestionnaire de données ?

+ bool has_keys **(keys, checkpoints = null) :** Détermine si un ou plusieurs identifiant(s) sont bel et bien définient dans un ou plusieurs point(s) de sauvegarde.
  » String | PoolStringArray **keys** : Contient tous identifiants à cherchés.
  » String | PoolStringArray **checkpoints** : Quel(s) est/ sont le(s) point(s) de sauvegarde qui sera(ont) ciblé(s) ? Par défaut, le point de sauvegarde actif sera pris pour cible.

+ Dictionary | String get_game_data **(json = true) :** Renvoie toutes les informations sur le gestionnaire des données du jeu.
  » bool **json** : Voulez-vous renvoyer le résultat sous le format json ?

+ Dictionary | String get_checkpoint_data **(checkpoint = ", json = true) :** Renvoie toutes les données présentes au sein d'un point de sauvegarde donné.
  » String **checkpoint** : Quel point de sauvegarde voulez- vous ciblé ? Par défaut, le point de sauvegarde actif sera utiliser pour faire le traitement.
  » bool **json** : Voulez-vous renvoyer le résultat sous le format json ?

+ void destroy_game_data **(on_disc = false, delay = 0.0) :** Supprime toutes les données du jeu au sein du gestionnaire. Faites très attention lorsque vous utilisez cette fonction, car il n'y a pas de retour arrière après une telle opération.
  » bool **on_disc** : Voulez-vous supprimer physiquement le fichier de sauvegarde du disque dure ?
  » float **delay** : Quel est le temps mort avant le nétoyage ?

+ void destroy_checkpoints **(checkpoints = null, delay = 0.0) :** Supprime un ou plusieurs point(s) de sauvegarde.
  » String | PoolStringArray **checkpoints** : Quel(s) est/ sont le(s) nom(s) de(s) point(s) de sauvegarde à détruire ? Par défaut, le point de sauvegarde actif est pris pour cible.
  » float **delay** : Quel est le temps mort avant le nétoyage ?

+ void set_checkpoint_data **(data, checkpoint = ", delay = 0.0) :** Mets à jour les données au sein d'un point de sauvegarde donné.
  » Dictionary **data** : Quelles sont différentes données qui constituent le point de sauvegarde en question. **Evitez de mettre des espaces dans vos clés si vous tenez à les récupérées**.
  » String **checkpoint** : Quel est le nom du point de sauvegarde à modifié ? Notez que si ce dernier n'est pas définit dans le gestionnaire, il sera automatiquement créé. Par défaut, le point de sauvegarde actif sera pris pour cible.
  » float **delay** : Quel est le temps mort avant la mise à jour du gestionnaire des données ?

+ int get_total_data_count **() :** Renvoie le nombre total de données définient au sein du gestionnaire des données du jeu.

IV – Les événements disponibles

+ **before_save :** Signal appelé avant la sauvegarde des données du jeu.

+ **before_load :** Signal appelé avant le chargement des données du jeu.

+ **game_data_changed :** Signal appelé lorsque la structure des données définient au sein du gestionnaire a changé.

+ **before_destroy (data) :** Signal appelé avant la destruction des données du jeu.
  » **Dictionary | Array data** : Contient des informations sur les différent(s) élément(s) à détruire.

+ **after_save (datum_count) :** Signal appelé après la sauvegarde des données du jeu.
  » **int datum_count** : Contient le nombre total de donnée(s) sauvegardée(s).

+ **after_load (datum_count) :** Signal appelé après le chargement des données du jeu.
  » **int datum_count** : Contient le nombre total de donnée(s) chargée(s).

+ **after_destroy (data) :** Signal appelé après la destruction des données du jeu. Cet événement renvoie un dictionaire contenant les clés suivantes :
  » **Array checkpoints** : Contient la liste de tous les points de sauvegarde supprimés.
  » **Array keys** : Contient la liste de toutes les clés supprimées.

+ **before_update (data) :** Signal appelé avant la mise à jour du gestionnaire des données (à chaque fois qu'on ajoute ou modifie la valeur d'une clé). Cet événement renvoie un dictionaire contenant les clés suivantes :
  » **String key** : Contient le nom de la clé à modifiée ou insérée dans le gestionnaire de données.
  » **String value** : Contient la valeur de la clé à modifiée insérée dans le gestionnaire de données.
  » **String checkpoint** : Contient le nom du point de sauvegarde pris pour cible.

+ **after_update (data) :** Signal appelé après la mise à jour du gestionnaire des données. (à chaque fois qu'on ajoute ou modifie la valeur d'une clé). Cet événement renvoie un dictionaire contenant les clés suivantes :
  » **String key** : Contient le nom de la clé modifiée ou insérée dans le gestionnaire de données.
  » **String value** : Contient la valeur de la clé modifiée ou insérée dans le gestionnaire de données.
  » **String checkpoint** : Contient le nom du point de sauvegarde pris pour cible.

+ **file_cant_open (data) :** Signal déclenché lorsqu'on a du mal à ouvrir le fichier de sauvegarde ou que son accès à été refusé. Cet événement renvoie un dictionaire contenant les clés suivantes :
  » **String path** : Contient le chemin pointant vers le fichier de sauvegarde.
  » **int type** : Contient le type de l'erreur déclenché.

» **String message** : Contient le méssage renvoyé par l'erreur déclenché.

+ **file_not_found (data) :** Signal déclenché lorsque le fichier de sauvegarde n'est pas définit. Cet événement renvoie un dictionaire contenant les clés suivantes :
  » **String path** : Contient le chemin pointant vers le fichier de sauvegarde.
  » **int type** : Contient le type de l'erreur déclenché.
  » **String message** : Contient le méssage renvoyé par l'erreur déclenché.

+ **file_corrupted (data) :** Signal déclenché lorsque le fichier de sauvegarde a été corromput de l'extérieur. Cet événement renvoie un dictionaire contenant les clés suivantes :
  » **String path** : Contient le chemin pointant vers le fichier de sauvegarde.
  » **int type** : Contient le type de l'erreur déclenché.
  » **String message** : Contient le méssage renvoyé par l'erreur déclenché.

+ **file_saving (data) :** Signal déclenché pendant qu'on sauvegarde les données du jeu dans le fichier de sauvegarde. Cet événement renvoie un dictionaire contenant les clés suivantes :
  » **String path** : Contient le chemin pointant vers le fichier de sauvegarde.
  » **int progress** : Contient la progression actuelle en pourcentage de la sauvegarde.

+ **file_loading (data) :** Signal déclenché pendant que le fichier de sauvegarde est en cours de chargement. Cet événement renvoie un dictionaire contenant les clés suivantes :
  » **String path** : Contient le chemin pointant vers le fichier de sauvegarde.
  » **bool is_over** : Est-ce que toutes les données du fichier de sauvegarde ont-elles été complètement chargées ?
  » **int progress** : Contient le nombre total de donnée(s) déjà chargée(s) en mémoire.

+ **game_time_changed (time) :** Signal déclenché à chaque fois que le temps écoulé depuis le démarrage du jeu, évolue.
  » **int time** : Contient le temps actuellement écoulé en seconde.