

# Structure du jeu SPY

## Listes des systèmes par scène dans l'ordre du MainLoop

### Scène « TitleScreen »

**TitleScreenSystem** : Gère le menu principal pour : charger les niveaux et scénarios ; afficher et débloquent les niveaux d'une campagne ; synchroniser les données du joueur (progression) ; recharger le contexte sur retour d'un niveau (campagne, éditeur de scénario, éditeur de niveau).

**ParamCompetenceSystem** : Gère les différents référentiels de compétence et l'éditeur de scénario.

**PopupManager** : Gère l'affichage des fenêtres de dialogues pour demander à l'utilisateur de valider/annuler ces choix.

**SettingsManager** : Gère la fenêtre de gestion des paramètres.

**VirtualKeyboardManager** : Gère le clavier virtuel en mode mobile/tablette.

**SendStatements** : Gère l'envoi des statements au LRS en dépillant les composants **SendUserData**.

**SyncLocalization** : Gère le changement de langues.

### Scène « MainScene »

#### Exécution en FixedUpdate

**RandomRotationSystem** : Gère la rotation du décors en arrière plan.

#### Exécution en Update

**SettingsManager** : Gère la fenêtre de gestion des paramètres.

**LevelGenerator** : Lit le XML d'un niveau et génère son contenu. Si le niveau contient un script pré rempli, génère le composant **ScriptToLoad**.

**ScriptGenerator** : Décode le XML des script (exploitation des composants **ScriptToLoad**) pour remplir les zones de scripts pré remplies dans la description du niveau.

**MoveSystem** : Maintient la position et l'orientation des agents en fonction de leurs composants **Position** et **Direction** et gère le déclenchement des animations.

**CoinManager** : Gère les collisions entre les robots contrôlés par le joueur et les pièces ramassables dans le jeu.

#### UISystem :

- Gère les boutons Play/Pause/Stop...
- Gère l'affichage des bonnes interfaces entre le mode édition (exploitation du composant **EditMode** généré par le **ModeManager**) et le mode exécution (exploitation du composant **PlayMode** généré par le **ModeManager**)
- Transfère les scripts des panneaux éditions vers les panneaux exécutions
- Maintient visible l'action exécutée en mode exécution

**HistoryManager** : Accumule les tentatives successives du joueur pour résoudre un niveau afin de lui présenter le récapitulatif de ses tentatives en fin ou en cas d'échec.

**DialogSystem** : Gère l’affichage des dialogues du briefing au début du niveau.

**CameraSystem** : Gère les déplacements et les rotations de la caméra.

**DragDropSystem** :

- Gère le déplacement d’une action de l’inventaire dans le script du joueur
- Gère le déplacement d’une action à l’intérieur du script du joueur
- Gère le double-clic sur les actions de l’inventaire joueur
- Gère la suppression d’une action du script du joueur (clic-droit ou déplacement d’un bloc depuis la zone d’édition vers l’inventaire)
- (Dés)Active les zones de drop
- Génère des composants **Dropped** et **AddOne**

**BlocLimitationManager** : Maintient à jour le nombre de blocs disponibles dans l’inventaire du joueur en fonction de la génération dynamique des composants **Dropped** (consommation d’une action de l’inventaire, voir **DragDropSystem** et **HistoryManager**) et **AddOne** (restitution d’une action dans l’inventaire en cas de suppression, voir **DragDropSystem**).

**StepSystem** : Contrôle la génération et la suppression du composant **NewStep** à intervalle de temps régulier.

**CurrentActionManager** : Gère le composant **CurrentAction**

- Initialise pour chaque nouvelle exécution l’action active de chaque script de chaque agent (ajout d’un composant **CurrentAction** sur la première action de chaque agent).
- A chaque pas de simulation (écoute du composant **NewStep** – voir **StepSystem**), calcule pour chaque script de chaque agent la nouvelle action active (suppression du composant **CurrentAction** de l’action précédente et ajout du composant **CurrentAction** sur la nouvelle)

**GameStateManager** : Serialize le contenu du jeu pour pouvoir être rechargé par exemple lorsque le joueur revient à l’état du jeu avant sa dernière exécution.

**EditableContainerSystem** : Ce système gère les interactions avec les containers servant à éditer les robots :

- Ajout/Suppression d’une colonne d’édition
- Nettoyage d’une colonne d’édition
- Synchronisation des noms entre un robot et sa colonne d’édition

**ScrollSystem** : Gère le scroll automatique sur les panneaux d’édition.

**VirtualKeyboardManager** : Gère le clavier virtuel en mode mobile/tablette.

**SendStatements** : Gère l’envoi des statements au LRS en dépilant les composants **ActionPerformedForLRS**.

**TooltipKeyboardSystem** : Modifie le comportement des tooltip en cas d’utilisation du clavier virtuel.

[Exécution en LateUpdate](#)

**HighLightSystem** : Met en surbrillance les objets du jeu au survol de la souris ou pour mettre en évidence l’action en cours d’exécution.

**CurrentActionExecutor** : Gère l’exécution des actions définies comme actives (contenant un composant **CurrentAction**, voir **CurrentActionManager**). Modifie par exemple les composants

**Position** et **Direction** (voir **MoveSystem**), consomme/produit les composants **TurnedOn** (voir **DoorManager**).

**EndGameSystem** : Vérifie si le joueur a atteint l'arrivée après la dernière exécution de son script. Gère les composants **NewEnd** générés par exemple par le **DetectorManager**.

**DoorAndConsoleManager** : Gère l'ouverture/fermeture des portes en fonction des actions réalisées sur les consoles de commande (exploitation des composants **TurnedOn** voir **CurrentActionExecutor**).

**DetectorManager** :

- Maintient à jour à chaque pas de simulation (écoute du composant **NewStep**, voir **StepManager**) la position des zones de détection en fonction de la position des agents
- Gère la détection des collisions entre le robot du joueur et les zones de détection (génère un composant **NewEnd**)

**Mode Manager** : Génère les composants **PlayMode** et **EditMode** pour notifier aux autres systèmes du changement de contexte.

[Scene « EditorScene »](#)

**EditorEscMenu** : Gère l'affichage du menu principal de l'éditeur.

**OpenFileSystem** : Gère l'ouverture d'un niveau.

**EditorGridSystem** : Lit le XML d'un niveau et charge la grille du niveau.

**TilePopupSystem** : Gère les fenêtres de configuration d'un objet pour pouvoir le configurer.

**EditorLevelDataSystem** : Lit le XML d'un niveau et charge les métas données du niveau.

**DragDropSystem** :

- Gère le déplacement d'une action de l'inventaire dans le script du joueur
- Gère le déplacement d'une action à l'intérieur du script du joueur
- Gère le double-clic sur les actions de l'inventaire joueur
- Gère la suppression d'une action du script du joueur (clic-droit ou déplacement d'un bloc depuis la zone d'édition vers l'inventaire)
- (Dés)Active les zones de drop
- Génère des composants **Dropped** et **AddOne**

**ScrollSystem** : Gère le scroll automatique sur les panneaux d'édition.

**SaveFileSystem** : Gère la sauvegarde d'un niveau.

**EditorCameraSystem** : Gère les mouvements de la camera dans l'éditeur de niveau.

**EditableContainerSystem** : Ce système gère les interactions avec les containers servant à éditer les robots :

- Ajout/Suppression d'une colonne d'édition
- Nettoyage d'une colonne d'édition

**BlocLimitationManager** : Maintient à jour le nombre de blocs disponibles dans l'inventaire du joueur en fonction de la génération dynamique des composants **Dropped** (consommation d'une action de

l'inventaire, voir **DragDropSystem** et **HistoryManager**) et **AddOne** (restitution d'une action dans l'inventaire en cas de suppression, voir **DragDropSystem**).

**ScriptGenerator** : Décode le XML des script (exploitation des composants **ScriptToLoad**) pour remplir les zones de scripts pré remplies dans la description du niveau.

**PopupManager** : Gère l'affichage des fenêtres de dialogues pour demander à l'utilisateur de valider/annuler ces choix.

## Classe utilitaire

**Utility** : Cette classe statique fournit une bibliothèque de fonctions partagée entre plusieurs systèmes.

**UtilityEditor** : Cette classe statique fournit une bibliothèque de fonctions partagée entre plusieurs systèmes de l'éditeur de niveau.

## Description des composants

**ActionPerformedForLRS** : CameraSystem, DialogSystem, DragDropSystem, EditableContainerSystem, EndGameManager, LevelGenerator, ModeManager, SendStatements, StepSystem, UISystem

Contient les données à utiliser par le système SendStatements pour envoyer des statements sur le LRS

**Activable** : SaveContent, TurnedOn, CurrentActionExecutor, CurrentActionManager, DoorAndConsoleManager, GameStateManager, LevelGenerator

Contribue à décrire le fonctionnement d'une console. Indique les slots commandés par la console.

**ActivationSlot** : CurrentActionManager, DoorAndConsoleManager, LevelGenerator

Contribue à décrire le fonctionnement d'une porte. Indique le slot commandant la porte.

**AddOne** : BlocLimitationManager, DragDropSystem

Indique si un item de l'inventaire doit être crédité de 1.

**AddSpecificContainer** : EditableContainerSystem, LevelGenerator

Indique qu'une nouvelle zone d'édition doit être créée. Contient les données utiles à cette création (nom, état, script initial).

**AgentColor** : HighlightSystem, LevelGenerator, EditingUtility

Indique les différentes couleurs utilisées pour mettre en évidence les couleurs des actions exécutées.

**AgentEdit** : TooltipContent, EditableContainerSystem, HistoryManager, LevelGenerator, UISystem

Contient le nom de l'agent à utiliser pour pouvoir le programmer.

**AskToRefreshCompetencies** : ParamCompetenceSystem, TitleScreenSystem

Composant flag utilisé pour demander au système ParamCompetenceSystem de recalculer les compétences en jeu dans un niveau.

**AskToSaveHistory** : HistoryManager, StepSystem

Indique que les scripts actuels doivent être enregistrés dans l'historique.

**AskToTestLevel** : ParamCompetenceSystem, TitleScreenSystem

Exploité par **ParamCompetenceSystem** pour tester un niveau.

**BaseCaptor** : BlocLimitationManager, EditingUtility

Indique qu'un bloc de programmation est un capteur.

**BaseCondition** : DragDropSystem, EditableContainerSystem, HighLightSystem, HistoryManager, EditingUtility

Indique qu'un bloc de programmation est un élément expression (capteur ou opérateur logique).

**BaseElement** : DragDropSystem, EditableContainerSystem, HighLightSystem, HistoryManager, LevelGenerator, StepSystem, UISystem

Indique qu'un bloc de programmation est une instruction (action ou structure de contrôle).

**BaseOperator** : BlocLimitationManager, EditingUtility

Indique qu'un bloc de programmation est un opérateur logique.

**BasicAction** : BlocLimitationManager, CurrentActionExecutor, CurrentActionManager, EndGameManager, UISystem

Indique qu'un bloc de programmation est une action qui peut être exécutée.

**CellBrush** : EditorGridSystem

Contient les propriétés du pinceau sélectionné.

**Competency** : ParamCompetenceSystem, TitleScreenSystem

Contient les propriétés d'une compétence

**ControlElement** : EditingUtility

Indique qu'un bloc de programmation est une structure de contrôle.

**CurrentAction** : SaveContent, CurrentActionExecutor, CurrentActionManager, EndGameManager, GameStateManager, HighLightSystem, LevelGenerator, StepSystem, UISystem, EditingUtility

Indique l'action en cours d'exécution.

**DataLevelBehaviour** : ParamcompetenceSystem

Contient les données d'identification d'un niveau

**Detector** : CurrentActionManager, DetectorManager, GameStateManager

Permet d'associer une zone de détection à une sentinelle.

**DetectRange** : DetectorManager, LevelGenerator

Permet de définir les propriétés de la zone de détection.

**Direction** : SaveContent, CurrentActionExecutor, CurrentActionManager, DetectorManager, GameStateManager, LevelGenerator, MoveSystem

Indique la direction dans laquelle l'élément de jeu doit s'orienter.

**DoorPath** : DoorAndConsoleManager, LevelGenerator

Indique à quel slot se réfère cette partie de chemin.

**Dragging** : DragDropSystem, HighLightSystem, ScrollSystem, UISystem, VirtualKeyboardManager

Indique qu'un bloc est en train d'être déplacé.

**Dropped** : BlocLimitationManager, DragDropSystem, HistoryManager

Indique qu'un élément de programmation vient d'être lâché dans une zone d'édition.

**DropZone** : DragDropSystem, EditingUtility

Indique que ce GameObject est une zone de drop qui peut être utilisée pour lâcher un nouveau bloc.

**EditableCanvacComponent** : EditableContainerSystem

Indique combien de zones programmables sont définies dans le panneau dédié à cet effet.

**EditMode** : CoinManager, CurrentActionManager, DetectorManager, DragDropSystem, ModeManager, StepSystem, UISystem

Indique si le jeu doit basculer en mode édition.

**EditorBlockData** : EditorLevelDataSystem, SaveFileSystem

Identifie un GameObject contenant les méta données d'un bloc.

**ElementToDrag** : BlocLimitationManager, HighLightSystem, LevelGenerator, UISystem, EditingUtility

Indique si un élément de jeu est en cours de drag ou pas.

**ExecutablePanelReady** : CurrentActionManager, UISystem

Indique que les panneaux d'exécution sont remplis.

**FocusCamOn** : CameraSystem, DialogSystem

Indique une position à atteindre à la caméra.

**ForceMoveAnimation** : CurrentActionExecutor, MoveSystem

Force le déclenchement d'une animation.

**ForceRemoveContainer** : EditableContainerSystem, HistoryManager, EditorLevelDataSystem

Force la suppression d'une zone d'édition.

**ForControl** : SaveContent, BlocLimitationManager, CurrentActionManager, GameStateManager, HighLightSystem, HistoryManager, LevelGenerator, UISystem, EditingUtility

Indique qu'un élément de programmation est une boucle de type « for ».

**ForeverControl** : BlocLimitationManager, CurrentActionManager, HighLightSystem, EditingUtility

Indique qu'un élément de programmation est une boucle de type « forever ».

**GameData** : BlocLimitationManager, CoinManager, DetectorManager, DialogSystem, DoorAndConsoleManager, DragDropSystem, EditableContainerSystem, EndGameManager, HistoryManager, LevelGenerator, MoveSystem, StepSystem, TitleScreenSystem, UISystem

Contient différentes données du jeu.

**GameLoaded** : BlocLimitationManager, DetectorManager, DoorAndConsoleManager, EditableContainerSystem, LevelGenerator

Indique que le jeu est chargé.

**Highlightable** : HighLightSystem

Indique qu'un objet du jeu peut être mis en évidence.

**IfControl** : BlocLimitationManager, CurrentActionManager, EditingUtility

Indique qu'un élément de programmation est une structure de contrôle du type « if then ».

**IfElseControl** : BlocLimitationManager, CurrentActionManager, EditingUtility

Indique qu'un élément de programmation est une structure de contrôle du type « if then else ».

**ItemSelector** : SyncLocalization

Permet d'identifier tous les boutons permettant de changer la langue.

**LevelData** : OpenFileSystem, SaveFileSystem

Contient les informations du fichier décrivant un niveau donné.

**LibraryItemRef** : BlocLimitationManager, CurrentActionManager, DragDropSystem, HighLightSystem, UISystem, EditingUtility

Indique à quel item de la bibliothèque un bloc intégré dans un script fait référence.

**LinkedWith** : LevelGenerator

Indique quel agent est relié un panneau d'exécution.

**LocalizationLoaded** : ParamCompetenceSystem, SyncLocalization, TitleScreenSystem

Permet de notifier différents systèmes lorsque les données de localization sont chargées.

**MessageForUser** : EditorEscMenuSystem, SaveFileSystem, ParamCompetenceSystem, PopupManager, TitleScreenSystem

Permet d'informer le **PopupManager** qu'une fenêtre de dialogue doit être présentée à l'utilisateur.

**Moved** : CurrentActionExecutor, DetectorManager

Indique qu'un drone est en mouvement.

**NeedRefreshHierarchy** : DragDropSystem, EditableContainerSystem

Indique que les zones d'édérations doivent être rafraîchies.

**NeedRefreshPlayButton** : DragDropSystem, EditableContainerSystem, UISystem

Indique que l'état du bouton Play doit être rafraîchi.

**NeedToDelete** : DragDropSystem, EditableContainerSystem

Indique qu'un bloc de la zone d'édition doit être supprimé.

**NewEnd** : CurrentActionManager, DetectorManager, EndGameManager, HistoryManager, StepSystem, UISystem, EditingUtility

Indique qu'une fin de niveau a été déclenchée.

**NewLevelToLoad** : EditorGridSystem, EditorLevelDataSystem, OpenFileSystem

Indique à différents système qu'un nouveau niveau doit être chargé.

**NewStep** : CurrentActionManager, StepSystem

Indique qu'un nouveau pas de simulation doit être exécuté.

**PaintableGrid** : EditorGridSystem, SaveFileSystem, TilePopupSystem

Contient les informations de la grille et l'emplacement des objets de la scène.

**PlayMode** : CoinManager, CurrentActionManager, DetectorManager, DragDropSystem, EndGameManager, GameStateManagement, ModeManager, StepSystem, UISystem

Indique si le jeu doit basculer en mode exécution.

**Popup** : EditorEscMenuSystem, TilePopupSystem

Permet d'identifier les GameObject servant à afficher les propriétés des objets de la scène dans l'éditeur.

**Position** : SaveContent, CurrentActionExecutor, CurrentActionManager, DetectorManager, DoorAndConsoleManager, EndGameManager, GameStateManager, LevelGenerator, MoveSystem, UISystem

Indique la position d'un élément sur la grille de jeu.

**RandomRotation** : RandomRotationSystem

Contient les données de rotation aléatoires.

**RaycastOnDrag** : DragDropSystem

Permet de rendre les UI au cours du drag invisibles pour l'EventSystem d'Unity.

**RefreshSizeOfEditableContainer** : DragDropSystem, EditableContainer

Indique qu'il faut recalculer la taille des zones éditables.

**ReplacementSlot** : DragDropSystem, EditableContainerSystem, LevelGenerator, EditingUtility

Indique que ce GameObject est une zone de drop qui peut être remplacée par un nouveau bloc.



**ResetBlocLimit** : BlocLimitationManager, DragDropSystem, EditingUtility

Indique qu'un bloc d'une ligne de programmation doit être restitué à l'inventaire.

**ScriptRef** : CurrentActionExecutor, CurrentActionManager, EditableContainerSystem, EndGameManager, HighLightSystem, HistoryManager, LevelGenerator, UISystem

Indique qu'un agent est lié à un panneau d'exécution.

**ScriptToLoad** : EditorLevelDataSystem, LevelGenerator, ScriptGenerator

Contient les data au format XML du script à charger.

**Selected** : DragDropSystem, EditingUtility

Indique la dernière zone de drop utilisée.

**SendUserData** : EndGameManager, SendStatements, TitleScreenSystem

Indique que des données utilisateurs doivent être envoyées au serveur.

**SyncColor** : HighLightSystem

Contient la liste des images dont la couleur doit être synchronisée.

**ToolTipContent** :

Indique le texte à afficher dans le tooltip.

**TurnedOn** : CurrentActionExecutor, DoorAndConsoleManager, LevelGenerator

Indique si une porte est ouverte ou fermée.

**UIRootContainer** : EditableContainerSystem, DragDropSystem, HistoryManager, LevelGenerator, UISystem, EditingUtility.

Indique le nom d'une zone éditables ainsi que ses propriétés

**UserData** : EndGameManager, SendStatements, TitleScreenSystem

Contient les données personnelles du joueur.

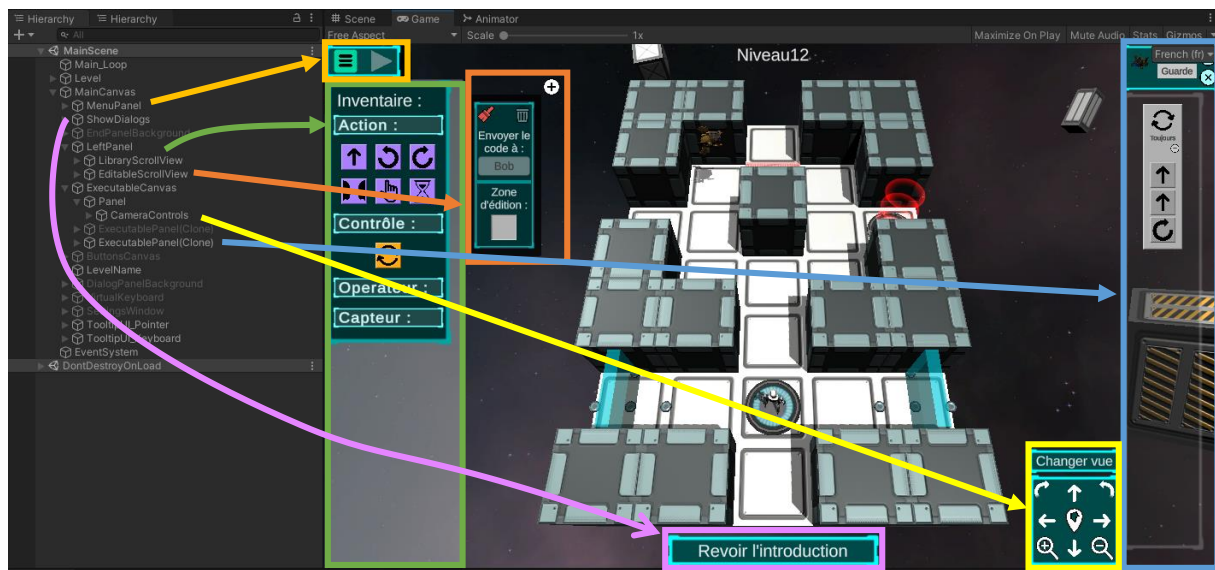
**ViewportContainer** : DragDropSystem, EditableContainerSystem, UISystem

Permet de repérer les containers des zones éditables.

**WhileControl** : BlocLimitationManager, CurrentActionManager, EditingUtility

Indique qu'un élément de programmation est une boucle de type « while ».

## Structure de l'UI



## Principe du gestionnaire de pas de simulation

Pour exécuter les actions des agents, SPY intègre une boucle de simulation spécifique. Celle-ci se structure autour des composants **PlayMode**, **ExecutablePanelReady**, **NewStep** et **CurrentAction**. Un pas de simulation SPY se répartit sur plusieurs cycles de mise à jour FYFY (et donc plusieurs pas de simulation Unity).

Lors du clic sur le bouton Play, le composant **PlayMode** est généré en T0. Ceci déclenche en T0+1 :

- Dans l'**UISystem** :
  - Le basculement d'interface en mode exécution
  - L'appel de la méthode **copyEditableScriptsToExecutablePanels** qui copie le contenu des zones d'édition dans les canvas d'exécution et génère le composant **ExecutablePanelReady**.
- Dans le **StepSystem** :
  - Son auto activation

La génération du composant **ExecutablePanelReady** (en T0+1) est récupéré par le **CurrentActionManager** en T0+2 qui ajoute les composants **CurrentAction** sur les premières actions à exécuter (voir la fonction **initFirstsActions**).

La génération des composants **CurrentAction** (en T0+2) est récupérée par le **CurrentActionExecutor** en T0+3 qui calcule le nouvel état des agents (position, rotation...).

Le **StepSystem** s'étant activé T0+1 par la génération du composant **PlayMode**, sa méthode **onProcess** produit un composant **NewStep** à intervalle de temps régulier (TX).

La génération du composant **NewStep** (généré en TX) est détecté par le **CurrentActionManager** en TX+1 et appelle la méthode **onNewStep()** : Pour chaque action courante (contenant le composant **CurrentAction**), suppression du composant **CurrentAction** de cette action et appel de la méthode **delayAddCurrentAction()** (coroutine). La coroutine **delayAddCurrentAction()** ajoute le composant **CurrentAction** à l'action courante en TX+2.

L'ajout en TX+2 des composants **CurrentAction** (cas similaire au cas T0+2) est récupéré par le **CurrentActionExecutor** en TX+3 qui lance les animations de déplacement des agents.

La fin du niveau est déterminée par la génération d'un composant **NewEnd**. Ce composant peut être généré par le **DetectorManager** si le robot est détecté par une sentinelle ; par le **CurrentActionManager** si le joueur crée une boucle infinie ou qu'aucun bloc ne peut être exécuté ; par le **EndGameManager** si le joueur a terminé le niveau ou si le joueur a épuisé toutes ses tentatives ; Par le **LevelGenerator** si le niveau à charger n'est pas connu ; par le **UISystem** ou le **ScriptGenerator** si un script envoyé au panneau d'exécution contient une condition non complète (voir **Utility.conditionToString**).

L'ajout d'un composant **NewEnd** est détecté par le **EndGameManager** pour afficher le bon panneau en fonction du type de fin.

## Exercices :

Téléchargez la dernière version du code de SPY (<https://github.com/Mocahteam/SPY>) et chargez le dans Unity (version 2020.3.19), vérifiez que le jeu se lance correctement.


Rendre actif par défaut tous les niveaux d'une campagne (aide : regardez la fonction **showLevels** du système **TitleScreenSystem**).

Ajouter un debug lorsque le joueur appui sur les touches A et E pour faire tourner la caméra

Augmenter la vitesse de rotation de la caméra sans modifier le code

Ajouter un debug à chaque fois qu'une action est ajoutée au programme depuis l'inventaire

Ajouter un debug à chaque pas de simulation (à chaque fois qu'un composant **NewStep** est généré)

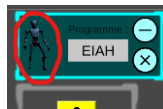
Ajouter un debug pour chaque action  exécutée

Ajouter un debug lorsque le joueur est détecté par un autre agent

Ajouter un debug lorsqu'une porte est ouverte

Ajouter un debug lorsque la souris survole un objet de l'inventaire (voir **HighLightSystem**)

Ajouter un debug lorsque la caméra se positionne sur la position d'un agent



Créer quelques niveaux avec l'éditeur de niveau et votre propre campagne avec l'éditeur de scénario