ISG - Rapport Projet

Max Maiche, 21312637
Léa Movsessian, 28624266
Paul-Tiberiu Iordache, 28706827

Sorbonne Université

2025

Sommaire

1 Introduction

1.1 Projet

Le jeu SPY est un jeu sérieux permettant à des élèves de primaire ou de collège d'apprendre les bases de la programmation.

Pour ce faire, un langage à base de blocs est utilisé pour programmer un ou plusieurs robots dont le but est d'atteindre un objectif en s'aidant de différentes actions.

Ces actions peuvent être effectuées simplement, avec un appel simple, mais également avec des boucles ou selon des conditions.

Le but de ce projet est d'implémenter de nouvelles mécaniques et d'exploiter les traces d'interaction du jeu.

Nous avons donc décidé d'implémenter le concept de fonction et de créer un tableau de bord destiné à l'enseignant.

1.2 Sources

Le code est disponible sur le github : MaxMaiche/SPY. Ce `README` est également présent dans l'annexe A.4. Les détails des fichiers modifiés se trouvent dans le fichier `README_MURATET.md`.
Le scénario créé se trouve dans le fichier Assets/StreamingAssets/Scenario/Desamorceur.xml et les niveaux correspondants dans le dossier Assets/StreamingAssets/Levels/Desamorceur.
Le script du tableau de bord est dans le fichier `dashboard_dash_plotly.py`.

2 Nouvelle Mécanique : Les Fonctions

2.1 Objectifs

Pour définir nos objectifs en matière d'implémentation du concept de fonctions dans le jeu, nous nous sommes basés sur ce qui définit une fonction :

- Une fonction est un bloc de code qui effectue une tâche spécifique.
- Elle permet de réutiliser du code en l'appelant plusieurs fois dans un programme.
- Elle aide à organiser le code en le divisant en sections plus petites et plus faciles à gérer, ce qui rend le programme plus clair et plus structuré.
- Elle peut recevoir des données, les traiter, puis retourner un résultat.

Nous nous sommes concentrés sur les trois premiers points, à savoir la création d'un bloc « Fonction » nommable permettant d'effectuer une tâche spécifique, utilisable - et réutilisable - dans le script des robots et contribuant ainsi à rendre le script plus léger et structuré.

Notre objectif est d'amener le concept de manière simple et ludique, tout comme cela a pu être fait pour les boucles ou les conditions.

2.2 Implémentation

L'implémentation des fonctions a été faite de manière à permettre la création de niveaux dans le jeu. Nous avons donc deux aspects à cette implémentation : le jeu en lui-même et l'éditeur de missions.

2.2.1 Jeu

(a) Prefab du bloc `fonction` (b) Left Panel

Figure 1: Apparence du bloc « Fonction »

Premièrement, nous avons dû définir l'apparence du bloc utilisé pour la fonction. Nous avons décidé de rester dans la logique d'apparence du jeu, avec des catégories séparées par couleur et des blocs de forme plutôt carrée. On peut par exemple voir sur la figure 1b que les capteurs sont tous de couleur bleue. Nous avons opté pour la couleur rose, comme le montre dans la figure 1a, car il s'agit d'une couleur non utilisée. Concernant la forme du bloc, nous avons décidé de rester sur une forme entièrement carrée, comme celle des actions, car leur fonctionnements sont similaires (il est possible d'utiliser les fonctions dans les boucles et hors des boucles mais pas en condition de `if`, par exemple).

Quant à la notation `f(x)` et au nom, nous avons choisi de privilégier une approche explicite, préférant nous concentrer sur les fonctionnalités du bloc, plutôt que sur une tentative d'intégrer de manière subtile le concept de fonctions au jeu.

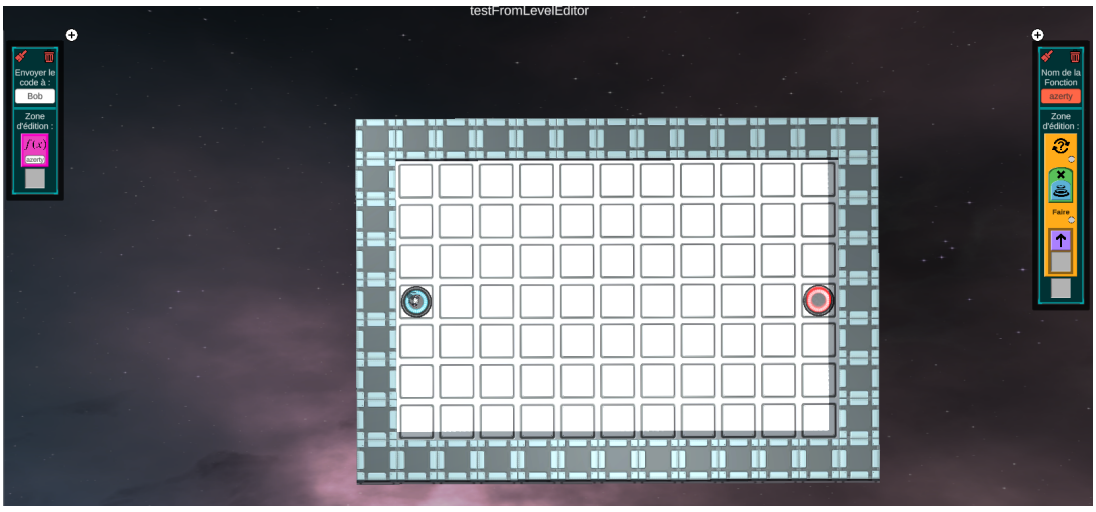


Figure 2: Niveau utilisant une fonction

La figure 2 montre comment les fonctions sont utilisées et apparaissent lorsque le joueur joue une mission. Le *container* du script associé au robot, dans lequel se trouve un bloc « Fonction », se trouve à gauche. Le *container* contenant le code correspondant à cette fonction se trouve à droite. Les deux containers sont plus précisément illustrés respectivement sur les figures 3a et 3b. Comme nous avons décidé d'utiliser le même container pour les scripts et les fonctions afin de préserver une cohérence visuelle, nous avons choisi de placer celui des fonctions à droite de l'écran afin de le différencier clairement du script associé au robot.

(a) Container du script du robot

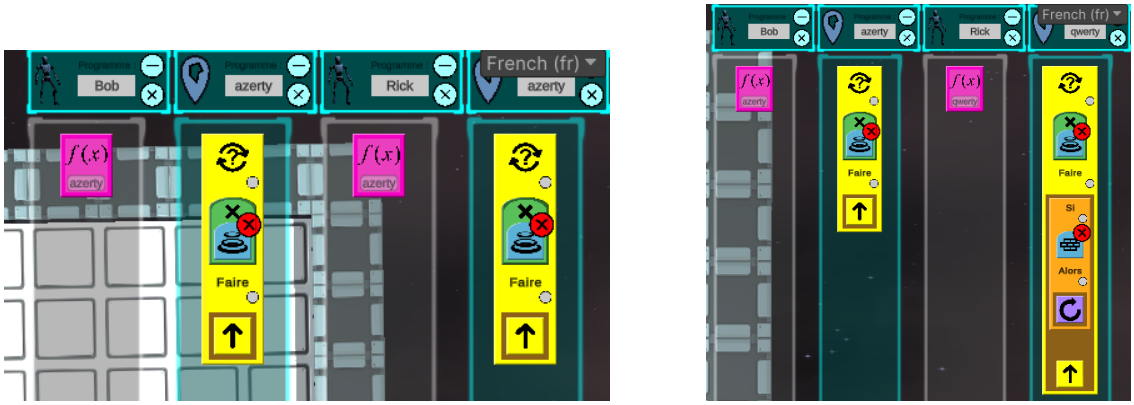
(b) Container de la fonction associée

Figure 3: Containers pendant le jeu

Comme on peut le voir sur la figure 3a, un nom est présent dans le bloc « Fonction ». Ce nom représente le nom d'appel de la fonction, qui doit être le même que dans la fonction définie à droite, ici `azerty` dans le « Nom de la fonction » comme on peut le voir sur la figure 3b. La zone d'édition du bloc « Fonction » est identique à celle du script. L'ajout d'une fonction se fait également de la même façon que l'ajout d'un script, grâce au « + » présent en haut à gauche du container. Grâce à cette gestion des noms, il est possible d'initialiser et d'appeler plusieurs fonctions : il suffit d'écrire le bon nom au bon endroit.

Figure 4: Exécution d'un niveau avec un seul robot

Pendant l'exécution, nous avons repris l'affichage de l'exécution du code du robot et nous avons ajouté l'affichage de la fonction, comme on peut le voir sur la figure 4, à droite. L'affichage de la fonction ne s'active qu'à l'entrée d'une fonction et disparaît à la sortie, ce qui évite d'avoir un nombre de container d'affichage exponentiel à mesure que l'on crée de nouvelles fonctions. Cela permet également de conserver la distinction entre le script de base et la fonction.



(a) Utilisant la même fonction (b) Utilisant des fonctions différentes

Figure 5: Exectution d'un niveau avec deux robots

Comme l'indique la figure 5, il est également possible de suivre l'exécution d'une ou de plusieurs fonctions appelées par plusieurs robots en même temps. Chaque robot dispose donc d'un *container* d'affichage de fonction associé, qui n'est actif que lorsque l'on rentre dans une fonction et qui est indépendant des autres. Une même fonction peut donc être affichée plusieurs fois sur l'écran si elle est appelée en même temps par plusieurs robots.

2.2.2 Création de missions

Il est tout autant possible de créer et d'éditer des niveaux contenant des fonctions à l'aide de fichiers XML qu'à l'aide de l'éditeur de missions présent dans le jeu. En effet, nous avons implémenté la gestion des fonctions dans la lecture et l'écriture de fichiers XML, ainsi que l'ajout du bloc « Fonction » dans l'éditeur de missions.

Changement fichier XML Dans le fichier XML, nous avons ajouté trois éléments. Premièrement le type de bloc « Fonction » et la limite d'utilisation, dans la partie `blockLimits` :

```
<blockLimit blockType="Function" limit="0" />
```

Cela permet de gérer le nombre de blocs fonction autorisé, à l'instar de tous les autres blocs utilisables.

Nous avons également ajouté le bloc de fonction dans le `script` tel que :

```
<script outputLine="Bob" editMode="0" type="3">
        <function name="avancer"/>
</script>
```

Dans ce script en particulier, on trouve uniquement un bloc « Fonction » dont le nom n'a pas été initialisé.

Finalement, on a également ajouté un bloc entier `function`, dans lequel est défini le code de la fonction. On se base ici sur le format du script :

```
<function outputLine="avancer" editMode="0" type="3">
        <while>
                <condition>
                        <not>
                                <captor type="Exit"/>
                        </not>
                </condition>
                <container>
                        <action type="Forward"/>
                </container>
        </while>
</function>
```

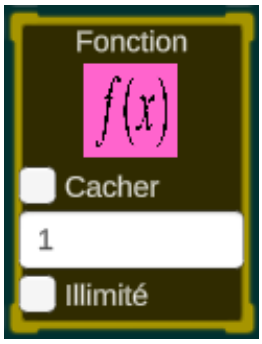
Dans cet exemple, nous avons donc une fonction qui, tant que le robot n'est pas sur la sortie, avance tout droit.

Editeur de missions Des modifications similaires ont également été réalisées dans l'éditeur de missions.

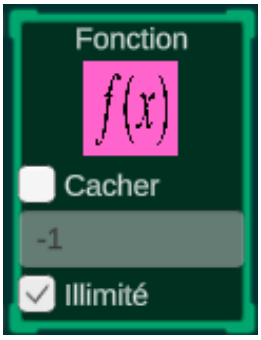
Figure 6: Paramètres globaux

Comme le montre la figure 6, les blocs de fonction sont disponibles.

(a) Indisponible



(b) Limité



(c) Illimité

Figure 7: Détail des paramètres pour les fonctions

Comme on peut le voir sur la figure 7, le bloc « Fonction » dans les paramètres globaux a les mêmes modes de disponibilités que les autres blocs : indisponible, disponible en quantité limitée ou disponible en quantité illimitée.

Figure 8: Container de fonction dans l'éditeur de missions

Le container de fonction repose, comme dans la partie jeu, sur le container de script. Il est donc possible de modifier le nom de la fonction qui peut ensuite être rendue éditable ou non par le joueur.

2.2.3 Sujets d'amélioration

Quelques points d'amélioration mériteraient d'être pris en compte. Il s'agit principalement de petits bugs ou d'optimisations de l'implémentation qui n'entravent pas significativement l'expérience de jeu :

- Le verrouillage du code des fonctions ne peut se faire qu'après la première exécution, ce qui empêche de bloquer l'utilisation des blocs de la fonction dans le code principal dès le départ.
- Il n'est pas possible d'utiliser des fonctions dans les Drones (ennemis).
- Il est impossible d'inclure des appels de fonction à l'intérieur d'autres fonctions. Le concept de récursivité est donc absent du jeu.

- Les affichages d'exécution se superposent parfois, surtout lorsqu'il y a des drones.
- Le Drag&Drop des blocs est parfois impossible au-dessus des appels de fonctions.
- Lorsque l'on saisit le nom d'une fonction, les raccourcis de la caméra sont activés.

2.3 Scénario

2.3.1 Description

Pour intégrer les fonctions dans le jeu, nous avons créé un scénario intitulé `Desamorceur`. Nous supposons que le joueur maîtrise déjà les autres mécanismes du jeu.

Le scénario est disponible en français et en anglais.

Nous nous sommes inspirés du scenario `Infiltration` pour créer une histoire ludique. Le joueur doit donc, avec le robot Bob, désamorcer une bombe se situant dans un Datry, un magasin rempli de robots. L'objectif est donc pour le robot de sauver sa famille et pour le joueur de l'aider.

Le scenario est structuré autour de l'apprentissage progressif des fonctions. Au début, le joueur est initié à des concepts simples comme l'écriture d'un script, qu'il connaît déjà et l'appel de fonctions déjà existantes (niveaux 1 à 3). Ensuite, l'objectif devient de créer et d'organiser des fonctions au sein d'un même script (pour un même robot), en introduisant des tâches de plus en plus complexes, telles que l'utilisation de plusieurs fonctions dans un même script ou la réutilisation d'une fonction dans plusieurs parties du programme (niveaux 4 à 8). L'idée est de renforcer la compréhension des mécanismes de fonctions, d'éviter la redondance de code et de découvrir l'importance de la modularité.

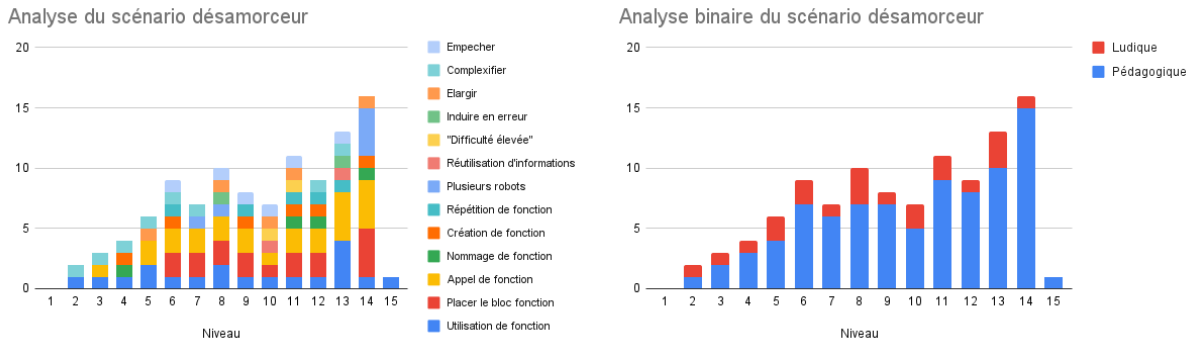
La difficulté augmente à partir du niveau 9 où le joueur est confronté à des défis qui nécessitent une réflexion plus poussée sur l'organisation des blocs. Un mécanisme (artificiel) semblable à l'importation de fichiers ou de fonctions est notamment introduit (niveau 10), de même que des niveaux plus complexes demandant d'utiliser plusieurs robots, fonctions, etc. Un moment clé du scénario se situe au niveau 13, où le joueur est confronté à une situation qui lui demande de reproduire un code qu'il a observé visuellement (compter le nombre de pièces) dans le niveau précédent. Ce niveau implique l'utilisation de quatre fonctions non modifiables, que le joueur doit organiser dans le bon ordre pour résoudre le problème. Le brouillard activé[1] empêche le joueur de deviner l'ordre d'exécution des fonctions, et il doit donc se fier à l'indice laissé dans le niveau précédent pour réussir. Cette étape renforce l'aspect ludique et interactif de l'apprentissage. Le scénario se termine par un niveau simple pour féliciter le joueur de sa réussite.

Le détail des scénarios est disponible dans l'annexe A.1.

2.3.2 Analyse de la complexité

Les descripteurs permettant d'analyser la complexité du scénario `Desamorceur` sont détaillés et expliqués dans l'annexe A.2.

[1] En créant ce niveau, nous avons remarqué que l'on voit les ennemis alors que le fog est activé, ce qui ne devrait pas être possible.



(a) Analyse complète (b) Analyse binaire

Figure 9: Analyse de la complexité du scénario `Desamorceur`

Comme le montre la figure 9, la progression de la complexité est correcte et s'accroît à mesure que les niveaux augmentent. Le dernier niveau n'ayant pas de but pédagogique, sa complexité est faible. On observe une baisse de complexité notable pour les niveaux 7 et 10, car des éléments importants y sont ajoutés : la possibilité d'utiliser la même fonction pour deux robots différents (niveau 7) et la notion d'import (niveau 10). On remarque ce même phénomène pour le niveau 12, qui s'explique par le fait que le joueur doit également voir un code (le nombre de pièces) à l'écran, code qui n'est pas une mécanique présente dans le jeu SPY.

Le nommage et la création de fonctions sont moins exploités que l'appel de fonctions. C'est parce qu'il est difficile de concevoir des niveaux intéressants où le joueur est obligé de créer des fonctions celles-ci pouvant généralement être remplacées par des boucles `while` ou `for` si le joueur y a accès. C'est pourquoi nous avons limité cela dans le scénario `Desamorceur`.

3 Analyse de traces : Tableau de bord

3.1 Objectifs

Pour définir nos objectifs concernant la création du tableau de bord, nous avons adopté le point de vue de l'enseignant. Nous nous sommes posé les questions suivantes :

- Comment évaluer le niveau d'un élève ou d'un groupe d'élèves ?
- Quels éléments seraient utiles pour vérifier l'acquisition de compétences liées à un niveau ?
- Comment représenter les données de manière simple et intuitive ?

3.2 Données

Pour répondre à nos problématiques, nous avons dû décider des éléments à afficher à l'enseignant. Nous nous sommes arrêtés sur trois données en particulier :

- Le temps mis pour accomplir un niveau.
- Le score obtenu lors de l'accomplissement d'un niveau.

- Le nombre d'essais nécessaires pour réussir un niveau.

Nous avons choisi ces données en particulier car elles offrent une vision assez complète et pertinente de la performance des élèves. Ces indicateurs permettent de mesurer non seulement l'efficacité et la rapidité d'apprentissage (temps et score), mais aussi la persévérance et les difficultés rencontrées (nombre d'essais). Ces éléments répondent directement aux besoins identifiés dans nos objectifs : évaluer le niveau des élèves et vérifier l'acquisition des compétences.

Pour garantir une présentation claire et facilement interprétable des données, nous avons opté pour trois histogrammes et un boxplot.

Le premier histogramme montre le score maximal et minimal obtenu pour chaque niveau, en le divisant par le score maximal atteignable. L'enseignant peut ainsi facilement comparer les performances des élèves par rapport à la difficulté de chaque niveau.

Le deuxième histogramme affiche les temps d'exécution (maximum, minimum et moyen) par niveau, offrant des informations sur la rapidité des élèves et sur les éventuelles difficultés rencontrées.

Le troisième histogramme illustre le nombre d'essais nécessaires pour chaque niveau, ce qui permet d'évaluer la persévérance des élèves et de repérer les niveaux où ils rencontrent des obstacles.

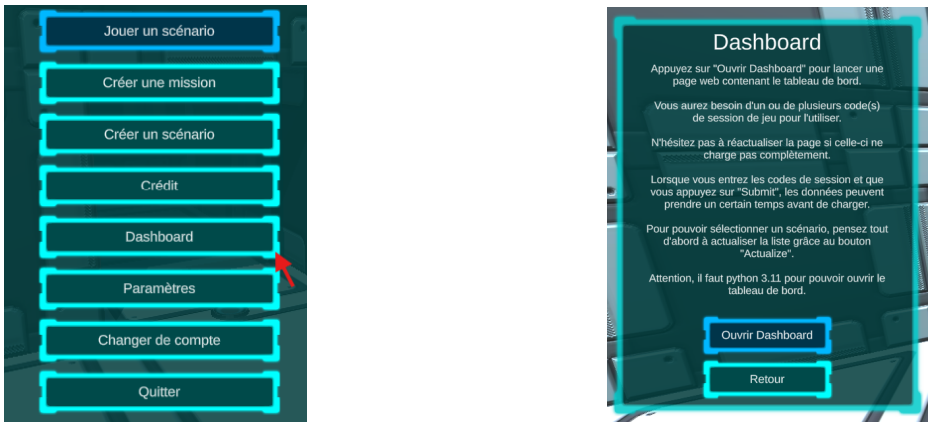
Enfin, le boxplot permet de visualiser la distribution des scores obtenus, divisés par le score maximal possible. Ce graphique met en évidence la variation des performances au sein de chaque niveau et permet à l'enseignant d'identifier les élèves qui se situent en dehors de la norme.

Ces choix visent à rendre les données accessibles, synthétiques et faciles à interpréter, et à permettre d'identifier les points forts et les axes d'amélioration de chaque élève.

L'utilité de ces graphiques sera présentée plus en détail dans l'étude de cas 3.4.

3.3 Implémentation

Pour créer le tableau de bord, nous avons décidé de le faire en dehors du jeu, en `python`. Pour cela, nous avons utilisé la bibliothèque `plotly dash` pour afficher les données.



(a) Bouton "Dashboard" du menu SPY (b) Menu Dashboard

Figure 10: Accès au tableau de bord depuis l'interface de SPY

Nous avons ajouté un bouton « Dashboard » au menu SPY (voir la figure 10a), qui ouvre un menu (voir la figure 10b) dans lequel on peut lire les indications pour utiliser le tableau de bord[2]. Il suffit ensuite de cliquer sur le bouton « Ouvrir Dashboard » pour que le script python[3] se lance.

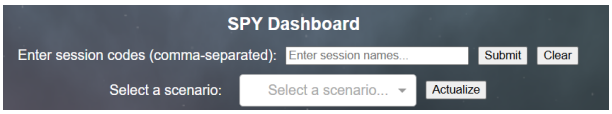


Figure 11: Entrée des données

Le tableau de bord se compose de deux zones : une entrée de données et un affichage de graphiques. Comme on peut le voir sur la figure 11, deux entrées sont présentes :

- Une zone de texte pour saisir les codes des sessions à analyser.
- Une liste déroulante avec les scénarios joués par les joueurs.

Lorsque le bouton « Submit » est appuyé, le script récupère les données LRS des codes insérés et les enregistre dans la mémoire. Si plusieurs codes sont saisis dans la zone d'entrée alors le programme effectue une moyenne des données, ce qui permet notamment d'observer le niveau moyen d'une classe.

Les scénarios récupérés ne se mettent pas à jour automatiquement. En effet, les `callbacks` de `dash` fonctionnant de manière asynchrone, il y a un risque que la liste déroulante ne s'actualise pas si cette opération est automatique. Il faut donc cliquer sur le bouton « Actualize » pour que la liste déroulante se mette à jour. Il est ensuite possible de sélectionner le scénario souhaité. Les niveaux affichés ne seront que ceux du scénario sélectionné.

Un bouton « Clear » permet également de supprimer les données récupérées en LRS de la mémoire. Cela permet de pouvoir refaire des requêtes pour une même session si le joueur a fait d'autres niveaux entre-temps, et ce, sans avoir à redémarrer le tableau de bord.

[2]Le texte est disponible dans l'annexe A.3 au cas où il ne serait pas lisible directement sur l'image.

[3]Attention, comme indiqué dans le menu du tableau de bord, il est nécessaire d'avoir python 3.11 pour pouvoir exécuter le script et d'avoir `python3.11.exe` dans le `path`

3.4 Etude de cas

Nous avons demandé à un élève de troisième, qui n'a aucune connaissance en programmation de jouer au scénario `Infiltration`. Nous avons ensuite récupéré ses données grâce au tableau de bord.

Figure 12: Affichage des données pour la session 4C6ED003

Les données représentées nous permettent d'observer plusieurs éléments relatifs à l'apprentissage de l'élève :

- Le graphique du nombre d'exécutions par niveau nous montre que, globalement, l'élève a réussi tous les niveaux du scénario en 1 ou 2 exécutions, mis à part pour les niveaux 13 et 20 pour lesquels il a visiblement eu beaucoup plus de mal.
- Cette observation se confirme lorsque l'on examine les scores : il n'a pas obtenu le score maximal pour les niveaux 13 et 20.
- De plus, les graphiques permettent également de déduire que le joueur n'a pas un profil d'*achiever*, puisqu'il n'a pas obtenu le score maximum pour des niveaux qu'il a réussis facilement (niveaux 2, 5 et 17). Il aurait pu en effet recommencer pour tenter d'obtenir les trois étoiles.
- L'histogramme des scores nous indique également que l'apprenant a trouvé une solution meilleure que la solution « optimale » pour le niveau 12 étant donné qu'il a obtenu un ratio supérieur à 1.
- En ce qui concerne le temps d'exécution, on voit que l'apprenant semble être relativement rapide dans sa résolution des niveaux, mis à part pour certains d'entre eux : le 1, le 10, le 12, le 13 et le 20. Pour les niveaux 13 et 20, cela concorde avec les observations précédentes. Pour le 1, étant donné que c'est le début du scénario, on peut supposer qu'il a dû lire les explications, les comprendre et appréhender le jeu.

Il est possible de pousser l'analyse en regardant à quelles compétences correspondent les niveaux et en déduisant quelles ont été les difficultés pour l'élève.

A Annexes

A.1 Détail des niveaux du scénario

- **Niveau 1 :** Le joueur commence sans fonction, et il doit simplement créer et exécuter un script assez simple. L'objectif de ce niveau est d'introduire le scénario.
- **Niveau 2 :** On introduit au joueur la notion de fonction, avec l'utilisation d'une fonction déjà existante, nommée et appelée. Le joueur doit simplement exécuter le niveau.
- **Niveau 3 :** Le joueur doit écrire le nom d'une fonction déjà créée et nommée dans le bloc d'appel de la fonction. Cela permet de présenter la notion d'appel de fonction.
- **Niveau 4 :** Le bloc d'appel de fonction est déjà prêt dans le script du robot. Le joueur doit ajouter une fonction et la coder. Ce niveau introduit la création de fonctions simples.
- **Niveau 5 :** Le script du robot est déjà créé, ainsi que deux fonctions. Les deux blocs d'appel de fonction sont placés mais pas nommés. Le joueur doit observer le niveau pour en déduire où appeler chaque fonction. Cela permet d'apprendre à utiliser plusieurs fonctions dans un même script.
- **Niveau 6 :** Le joueur doit utiliser une même fonction simple, qu'il a créée lui-même, à deux reprises dans le script, ce qui lui montre un des avantages des fonctions, à savoir l'évitement de la redondance de code.
- **Niveau 7 :** Une fonction est définie, nommée et non modifiable. Le joueur doit placer les blocs de fonctions dans deux robots différents. Cela permet de montrer à l'apprenant une autre manière d'éviter la redondance en utilisant une même fonction dans deux scripts différents.
- **Niveau 8 :** Ce niveau combine le niveau 6 et 7, obligeant le joueur à utiliser plusieurs fonctions pour des robots différents.
- **Niveau 9 :** Ce niveau est assez simple. Il doit créer une fonction simple et l'utiliser à plusieurs reprises dans le script du robot. Ce niveau est en interaction avec le niveau 10.
- **Niveau 10 :** Le joueur récupère la fonction qu'il a créé dans le niveau précédent. Ce mécanisme est artificiel. On s'est assuré que le joueur ne pouvait pas modifier le nom de la fonction et qu'il n'y avait qu'une solution disponible dans le niveau 9. Nous avons cherché ici à simuler la notion d'import sans toutefois l'expliciter, étant donné qu'elle est peut-être trop avancée pour des élèves qui viennent d'être initiés aux fonctions.
- **Niveau 11 :** Ce niveau est assez compliqué. L'apprenant est totalement autonome. Il a accès à tous les blocs, à l'exception de quelques uns, limités, afin de l'obliger à utiliser une fonction.
- **Niveau 12 :** Ce niveau est semblable au 9 en termes de difficulté. De plus, il est en interaction avec le niveau 13. L'objectif ici est de compter les pièces présentes dans la pièce par groupe, afin d'obtenir un code (3214).

- **Niveau 13 :** Ce niveau montre l'intérêt ludique de l'ajout des fonctions dans SPY. Quatre fonctions ont été définies et sont non modifiables. Quatre blocs de fonctions sont insérés dans le script du robot, et l'apprenant doit entrer les noms des fonctions dans le bon ordre. Le brouillard est activé, ce qui empêche le joueur de connaître l'ordre d'appel des fonctions s'il n'a pas fait attention au code du niveau précédent, car il ne voit pas le chemin[4].

- **Niveau 14 :** Le joueur est autonome. Il doit créer plusieurs fonctions pour plusieurs robots.

- **Niveau 15 :** Ce niveau est un niveau simple visant à célébrer la victoire du joueur. Il a un but purement ludique.

A.2 Données pour l'analyse de complexité

Descripteurs	**Description**
Pédagogiques	
Utilisation de fonction	Nombre de fonctions différentes à utiliser
Placer le bloc « Fonction »	Nombre de bloc « Fonction » à placer dans un ou plusieurs scripts de robots
Appel de fonction	Nombre de fonctions à appeler, c'est-à-dire le nombre de fois où le joueur doit entrer le nom d'une fonction dans un bloc « Fonction »
Nommage de fonction	Nombre de fonctions à nommer
Création de fonction	Nombre de fonctions à créer
Répétition de fonctions	1 si on doit utiliser plusieurs fois la même fonction, 0 sinon
Plusieurs robots	Nombre de robots supplémentaires pour lequel le script doit être rempli
Réutilisation d'informations	1 si le niveau réutilise des informations obtenues dans le niveau précédent, 0 sinon
"Difficulté élevée"	1 si le niveau demande une autonomie et serait considéré comme un niveau avec une certaine difficulté sans le concept de fonction, 0 sinon
Ludiques	
Induire en erreur	1 s'il y a un élément du niveau qui induit (au moins un peu) en erreur, 0 sinon
Elargir	1 si le niveau est plus grand et offre plus de liberté de mouvement, 0 sinon
Complexifier	1 s'il y a un nouveau élément de gameplay, dans les actions, le mécanisme de jeu ou dans l'environnement, 0 sinon
Empêcher	1 si un élément est posé pour empêcher la solution évidente du niveau, 0 sinon

Table 1: Description des descripteurs

[4]Nous avons remarqué en créant ce niveau que les ennemis sont visibles dans le fog, ce qui ne devrait pas être possible.

Niveau	Utilisation	Placer le	Appel de	Nommage	Création c	Répétition c	Plusieurs i	Réutilisati	"Difficulté	Induire	Elargir	Complexifi	Empecher
1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	1	0
3	1	0	1	0	0	0	0	0	0	0	0	1	0
4	1	0	0	1	1	0	0	0	0	0	0	1	0
5	2	0	2	0	0	0	0	0	0	0	1	1	0
6	1	2	2	0	1	1	0	0	0	0	0	1	1
7	1	2	2	0	0	0	1	0	0	0	0	1	0
8	2	2	2	0	0	0	1	0	0	1	1	0	1
9	1	2	2	0	1	1	0	0	0	0	0	0	1
10	1	1	1	0	0	0	0	1	1	0	1	0	1
11	1	2	2	1	1	1	0	0	1	0	1	0	1
12	1	2	2	1	1	1	0	0	0	0	0	1	0
13	4	0	4	0	0	1	0	1	0	1	0	1	1
14	1	4	4	1	1	0	4	0	0	0	1	0	0
15	1	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13: Détails de l'analyse par niveau

Niveau	Pédagogique	Ludique
1	0	0
2	1	1
3	2	1
4	3	1
5	4	2
6	7	2
7	6	1
8	7	3
9	7	1
10	5	2
11	9	2
12	8	1
13	10	3
14	15	1
15	1	0

Figure 14: Détails de l'analyse par niveau par groupe de descripteurs

A.3 Texte du menu Dashboard

Appuyez sur "Ouvrir Dashboard" pour lancer une page web contenant le tableau de bord.

Vous aurez besoin d'un ou de plusieurs code(s) de session de jeu pour l'utiliser.

N'hésitez pas à réactualiser la page si celle-ci ne charge pas complètement.

Lorsque vous entrez les codes de session et que vous appuyez sur "Submit", les données peuvent prendre un certain temps avant de charger.

Pour pouvoir sélectionner un scénario, pensez tout d'abord à actualiser la liste grâce au bouton "Actualize".

Attention, il faut python 3.11 pour pouvoir ouvrir le tableau de bord.

A.4 Résumé des modifications du code

A.4.1 Systèmes modifiés

- **BlocLimitationManager**: Ajout du bloc « Fonction ».
- **CurrentActionManager**:
  - Gestion d'appel de fonction.
  - Instanciation des fonctions (panel d'exécution, etc.).
  - Gestion des instructions exécutées dans les fonctions.
  - Gestion de fin de fonction.
- **DragDropSystem**: Gestion d'input de nom de fonction.
- **EndGameManager**: Ajout d'une fin si le nom de fonction n'est pas valide.
- **HighLightSystem**: Ajout de gestion dans le panel de fonction.
- **EditorLevelDataSystem**: Gestion des fonctions dans l'éditeur.
- **SaveFileSystem**: Sauvegarde des fonctions pré-implémentées.
- **LevelGenerator**: Lecture des fonctions sauvegardées pour leur génération.
- **ScriptGenerator**: Transformation d'une `"FunctionToLoad"` en `GameObject`.
- **StepSystem**: Gestion pour l'exécution dans une fonction.
- **UISystem**:
  - Création des conteneurs pour les fonctions.
  - Nettoyage à la fin d'une exécution.
- **TitleScreenSystem**: Correction du chemin pour la lecture des scénarios locaux.

A.4.2 Systèmes ajoutés

- **EditableContainerSystemFunction**: Copie du système `EditableContainerSystem` pour gérer les conteneurs de fonctions.
- **LaunchDashboardScript**: Script pour le lancement du script Python qui crée le Dashboard.

A.4.3 Composants modifiés / ajoutés

- `Function`
- `AddSpecificContainerFunction`
- `FunctionToLoad`
- `ScriptRef`
- `NewEnd`