IUT Bordeaux

Département Informatique

PROJET COMMUN POO – COO : Hex Light Bot

GENTY Laurent - DEVRIÉSÈRE Aymeric

QUESTIONS

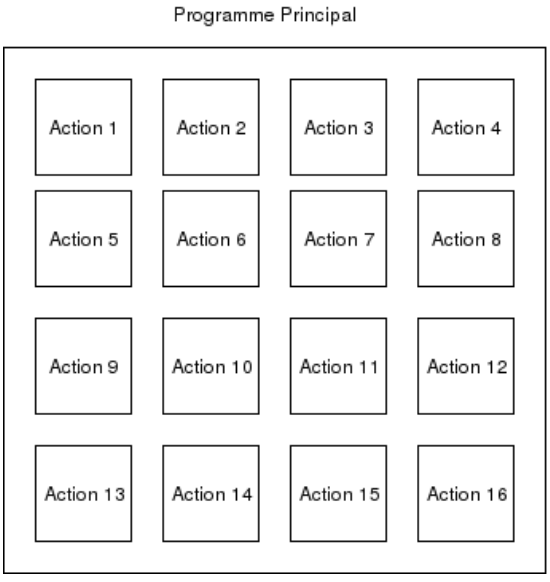
1) Liste des actions élémentaires proposées au « joueur / programmeur »

Dans notre projet, nous donnons différentes possibilités pour le joueur / programmeur. Ces actions sont essentielles au fonctionnement du jeu et sont :

- choisir son programme : jouer ou éditer
- choisir son niveau
- choisir des instructions, les enlever ou les ajouter aux programmes de jeu
- lancer son niveau et déplacer son robot
- stopper l'avancement du niveau
- quitter le jeu

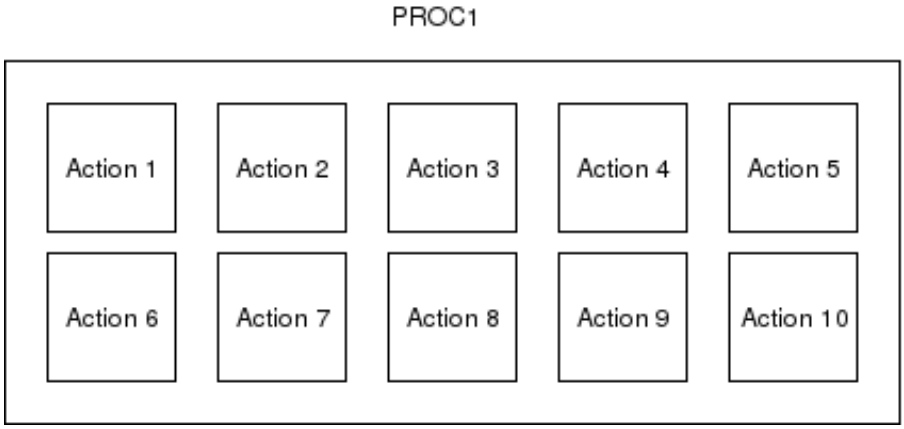
2) Taille maximale du programme MAIN

Le programme principal MAIN contiendra une liste d'actions et l'utilisateur pourra rajouter ou supprimer des actions de ce programme. Ce programme décrira donc le comportement du robot dans l'avancée du niveau. Ce MAIN sera donc un rectangle 4x4 pour un total de 16 actions comme celui ci : (les actions s'effectueront donc à la suite et l'utilisateur pourra voir clairement de quoi est composé son programme)



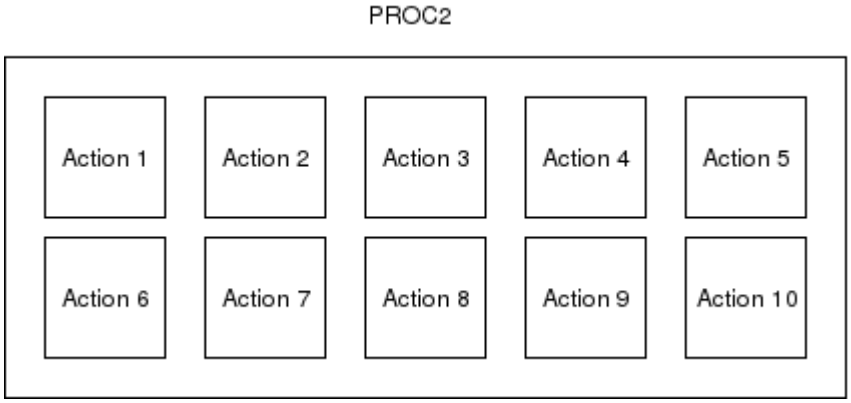
3) Taille maximale du programme PROC1

A l'instar du MAIN, le PROC1 est un programme composé d'une suite d'actions mais contrairement au MAIN, celui-ci peut-être appelé dans le main en tant que sous-programme. Il ressemblera à un rectangle d'une taille de 5x2 actions pour un total de 10 actions :



4) Taille maximale du programme PROC2

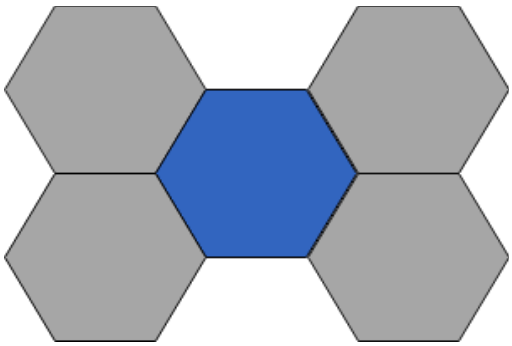
Tout comme le PROC1, le PROC2 est un sous-programme pouvant être appelé depuis le MAIN et est aussi composé d'une suite d'actions d'une taille de 5x2, pour un total de 10 actions :



Cependant nous avons fait le choix pouvoir appeler le PROC1 depuis le PROC2 et inversement. Ce qui veut dire que les combinaisons et le nombre d'actions disponibles pour l'utilisateur devient beaucoup plus grand.

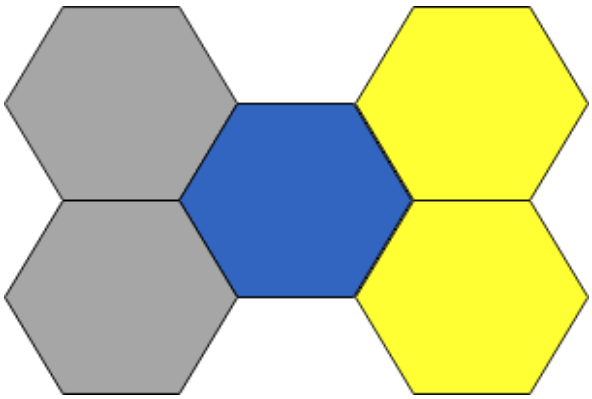
5) Identification du passage sur hexagones (1)

L'utilisateur verra sur l'interface, une série de cases. Les cases dites « spéciales » sur lesquelles il devra passer pourront donc être reconnaissables car elles seront BLEUES et donc, seront différenciables des cases dites « normales » : (cases normales en gris avec une case spéciale en bleue)



6) Identification du passage sur hexagones (2)

L'utilisateur pourra savoir quand il a allumé une case lorsqu'une case précédemment bleue est passée en JAUNE (ci-dessous – on peut voir deux cases sur lesquelles le robot est passé) :



7) Fonctions et contrôles du jeu

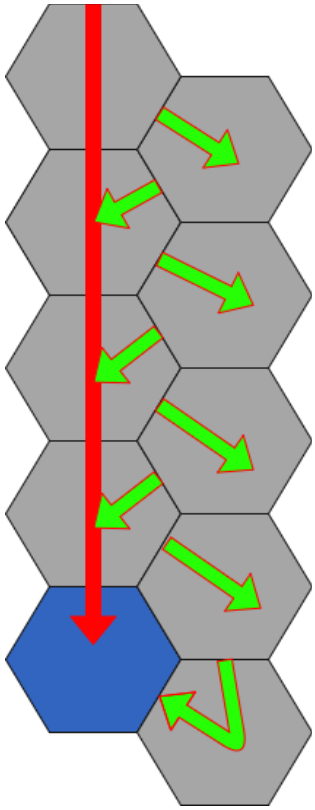
L'utilisateur aura un panel de possibilités dans le jeu et pourra accéder à différentes fonctionnalités du jeu comme celle suivantes :

- ✗ Reboot : repartir au tout premier niveau
- ✗ Reset : « reset » (recommencer) le niveau actuel (remettre le robot à sa position initiale)
- ✗ Quit : revenir au menu principal
- ✗ HARD Reset : remettre le niveau à 0 (remettre le robot à sa position initiale + effacer les programmes)
- ✗ Clean : supprimer les programmes
- ✗ Save : sauvegarder l'avancée des niveaux (nombre de niveaux atteints)
- ✗ Menu : retourner au menu des niveaux
- ✗ Play : lancer le jeu (lancer les programmes)
- ✗ Choose Level : choisir son niveau
- ✗ Actions : modifier les programmes (ajouter / supprimer / changer ordre actions)
- ✗ Help : expliquer les règles et contrôles
- ✗ Change Height : changer hauteur case
- ✗ Change Position : changer position Robot (en mode édition)
- ✗ Change Type : changer type d'une case (case spéciale)
- ✗ Save Level : sauvegarder un niveau édité (dans un fichier texte)

8) Algorithme du calcul de score

Notre algorithme permettant de calculer le score se présente sous cette manière : l'utilisateur devra emprunter « le plus court chemin ». C'est à dire : comparaison du nombre de cases optimal afin d'atteindre l'objectif au nombre de cases parcourues par l'utilisateur :

Par exemple, dans l'exemple ci contre, si l'utilisateur prends le chemin vert, certes, il aura beau avoir fait le programme le moins coûteux en terme d'actions, il emprunt un nombre de cases trop important par rapport au passage optimal, et donc perdra.



9) Niveaux implémentés

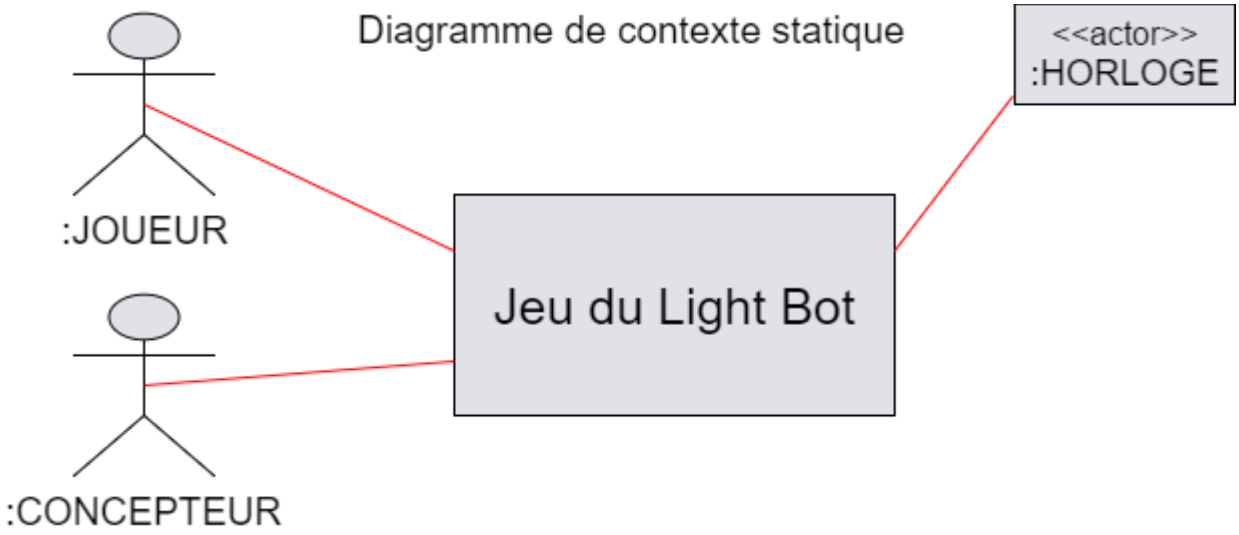
Nous allons implémenter différents niveaux ayant chacun une utilité pour l'utilisateur lui permettant de s'accommoder avec les différents contrôles et gestion des programmes et sous-programmes. Nous allons adopter une difficulté croissante lui permettant de comprendre les mécanismes tout en réfléchissant :

Un total de 7 niveaux de base :

1. Découverte du jeu et des mécanismes de base : avancer, allumer la lumière sur une ligne droite
2. Découverte mécanismes pour tourner le robot
3. Niveau de test pour le joueur sur les mécanismes appris (difficulté accrue)

4. Découverte mécanismes du saut (niveau plus compliqué)
5. Niveau test plus compliqué
6. Découverte du mécanisme sous-programme PROC1 a réutiliser dans le MAIN
7. Découverte sous-sous-programme avec PROC1 ET PROC2

10) Diagramme de contexte statique



11) Liste des événements externes

- ✗ ARR_BoutonJouer
- ✗ ARR_BoutonEdition
- ✗ ARR_LancerProgramme
- ✗ ARR_ModifierProgramme (cela peut être ajouter / supprimer / déplacer une case)
- ✗ ARR_BoutonEnJeu
- ✗ ARR_ModifierPositionRobot
- ✗ ARR_ModifierCase (l'allumer / éteindre / changer sa hauteur / ajouter / supprimer une case)
- ✗ ARR_ResetProgramme
- ✗ ARR_HardReset
- ✗ ARR_ResetNiveauEdition
- ✗ ARR_EnregistrerNiveau
- ✗ ARR_EnregistrerAvancée

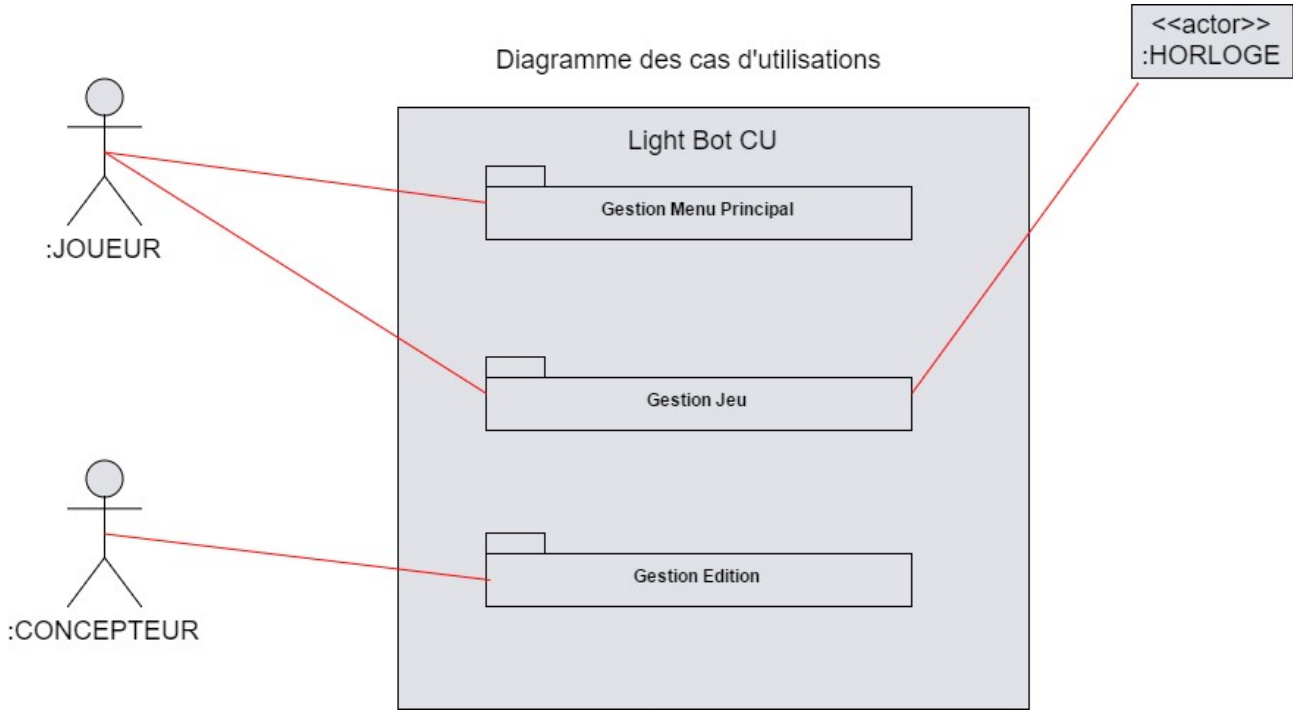
12) Liste des événements temporels

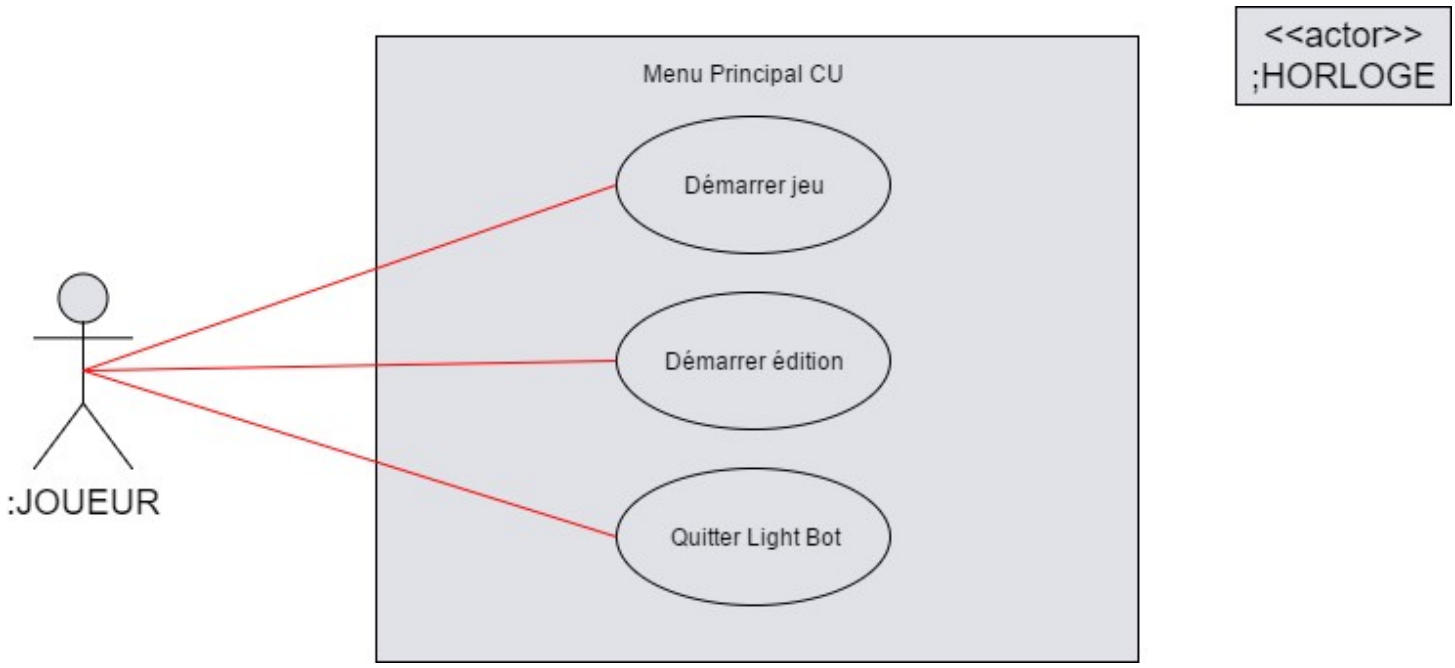
- ✗ ENV_Timer

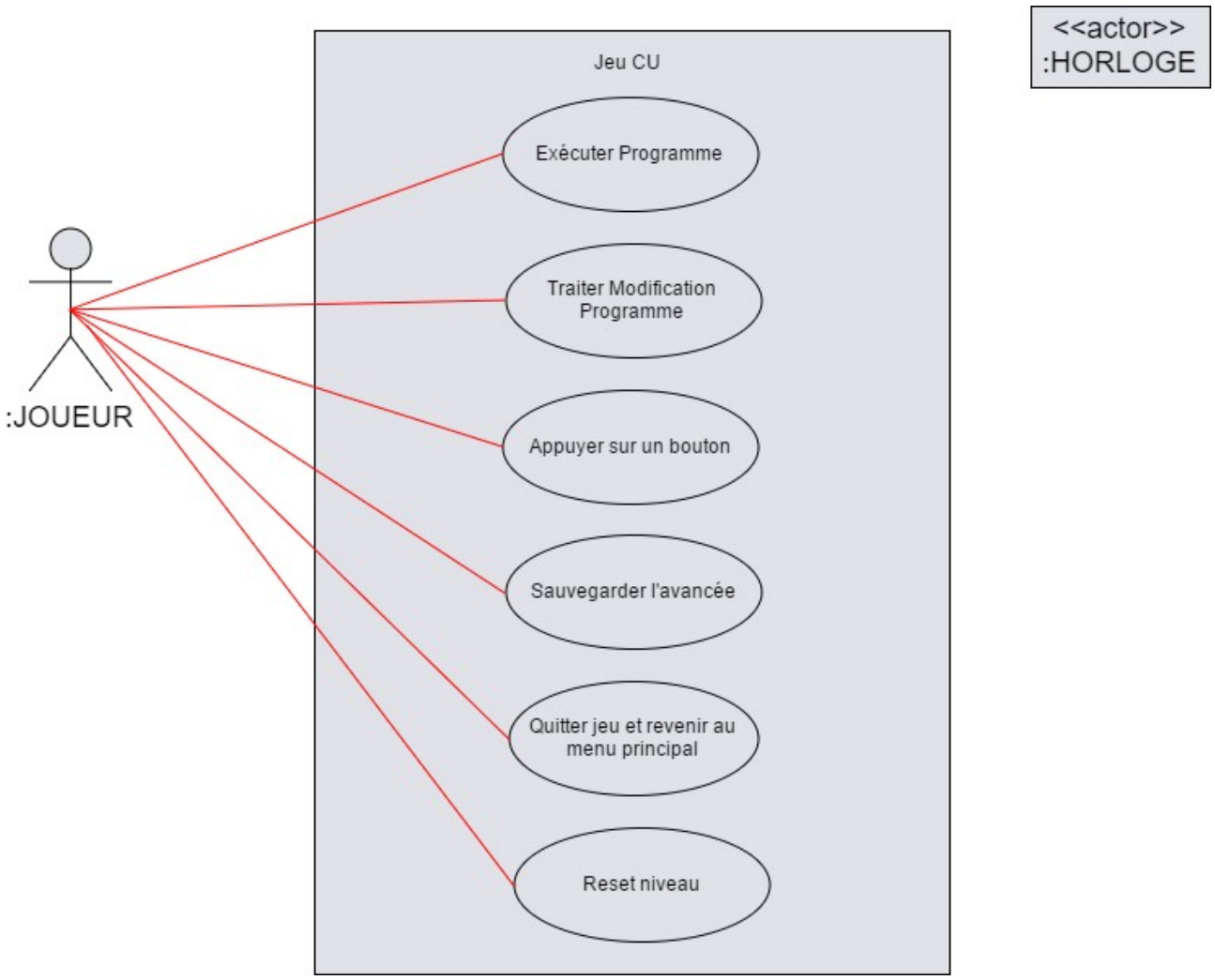
13) Liste des événements résultats

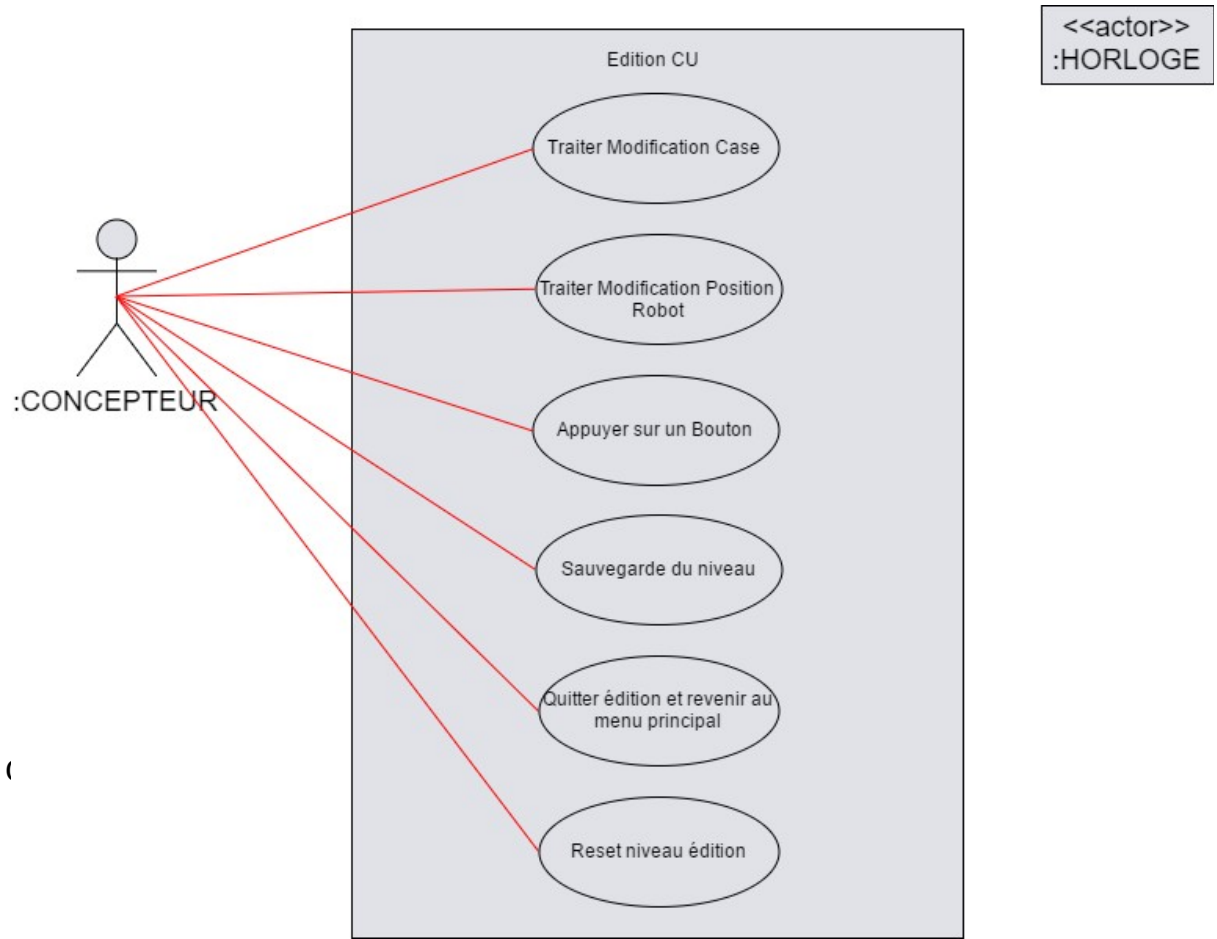
- ✗ ENV_TraiterActions (quand le programme est lancé)
- ✗ ENV_TraiterPositionRobot (changer position du robot)
- ✗ ENV_TraiterModifierProgramme (cela peut être ajouter / supprimer / déplacer une case)
- ✗ ENV_TraiterModifierCase (l'allumer / éteindre / changer sa hauteur / ajouter / supprimer une case)
- ✗ ENV_RobotSeDéplace
- ✗ ENV_MAJAffichage
- ✗ ENV_EnregistrementNiveauFichier
- ✗ ENV_MiseEnPause
- ✗ ENV_SuppressionNiveauEdition

14) Diagrammes des cas d'utilisations

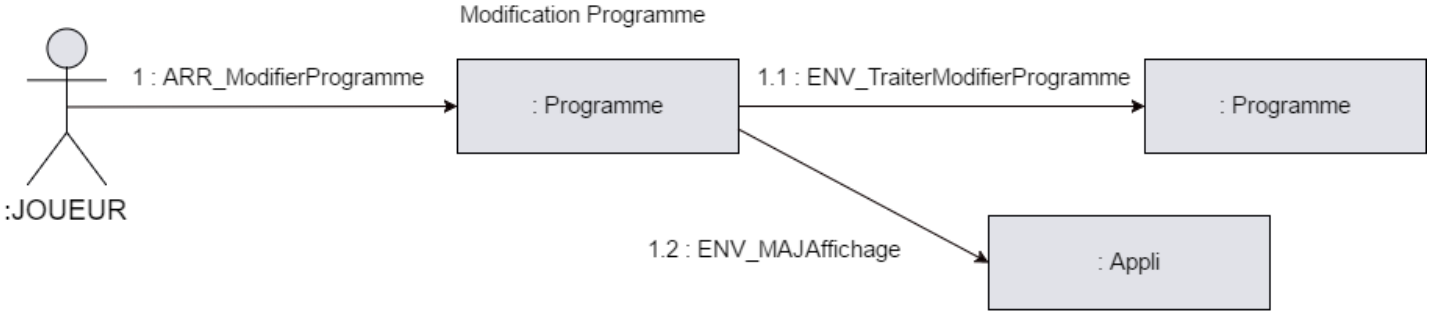




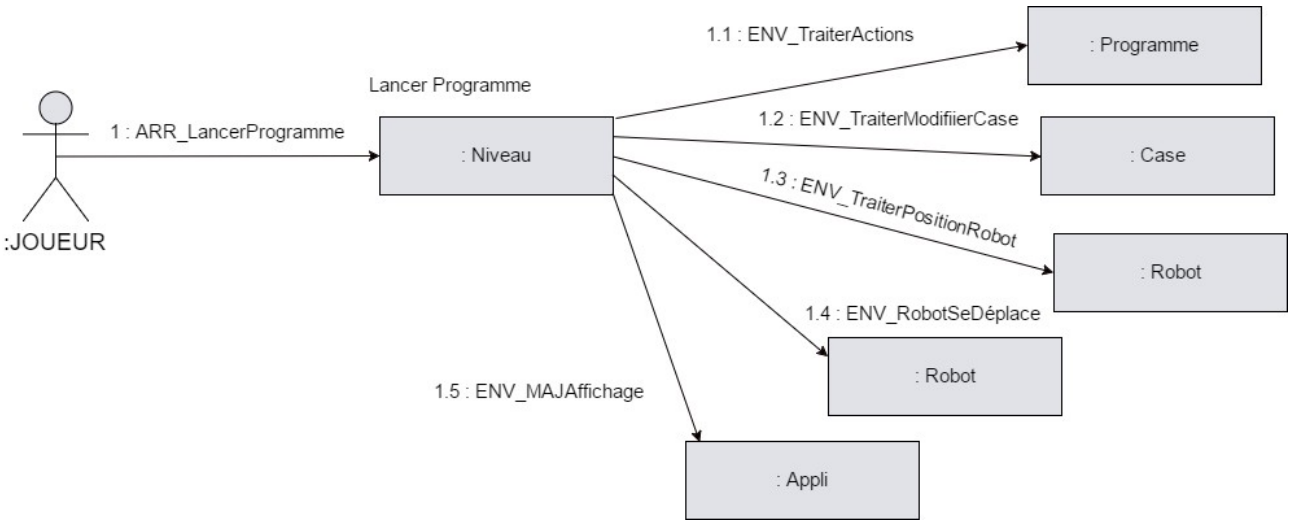

<<actor>>
:HORLOGE
Jeu CU
Exécuter Programme
Traiter Modification Programme
Appuyer sur un bouton
Sauvegarder l'avancée
Quitter jeu et revenir au menu principal
Reset niveau
:JOUEUR


<<actor>>
:HORLOGE
Edition CU
Traiter Modification Case
Traiter Modification Position Robot
Appuyer sur un Bouton
Sauvegarde du niveau
Quitter édition et revenir au menu principal
Reset niveau édition
:CONCEPTEUR

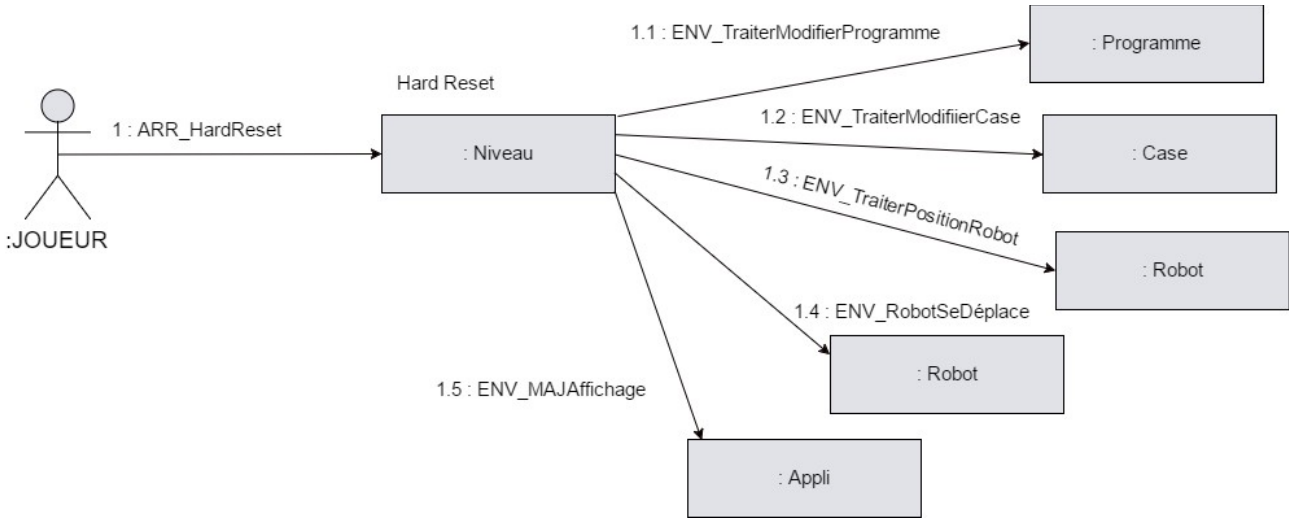
15) Diagrammes de communication pertinents



Le joueur peut ajouter, supprimer ou déplacer une action dans le programme. Dans ce diagramme de communication on peut voir le premier événement externe qui intervient qui est le ARR_ModifierProgramme : cet événement va agir sur la classe Programme (donc changer les liste d'actions) qui par la suite va produite des événements résultats. Le premier est un événement qui va traiter la demande : par exemple ajouter une action. Elle va donc vérifier si le programme peut recevoir une action supplémentaire dans sa liste d'actions (vérifier la taille du tableau d'actions, si c'est possible de l'ajouter) enfin, dans le scénario où l'action peut être ajoutée sans problème, alors on va mettre à jour l'écran pour afficher l'action en plus dans le programme sur l'écran : donc on agit sur la classe Appli possédant une fenêtre d'affichage.

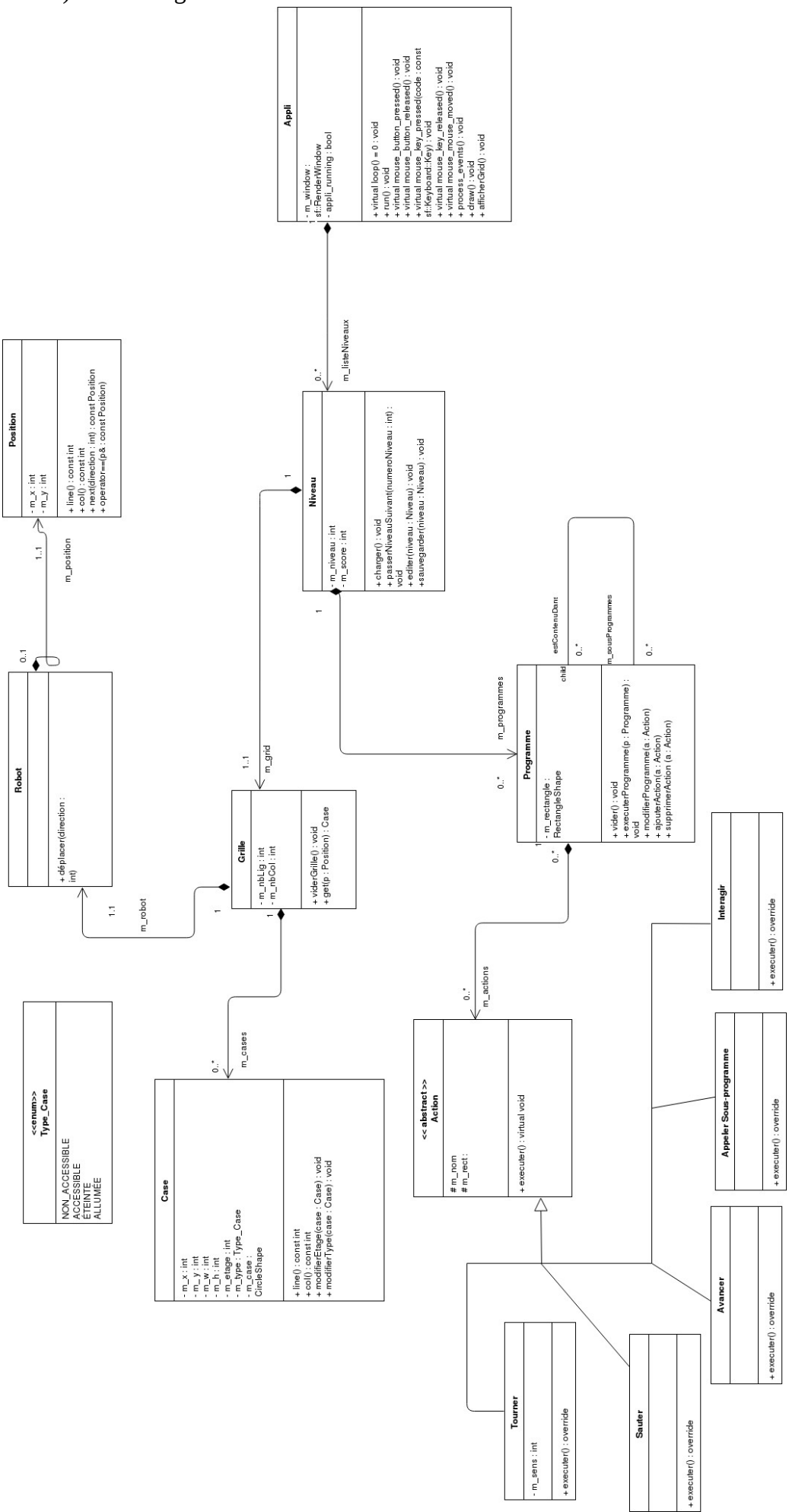


Faisons le diagramme de communication lorsque le joueur décide de lancer une suite d'actions. Tout d'abord partons du principe que tout se passe bien (les déplacements sont bons). L'événement ARR_LancerProgramme agira donc sur la classe Niveau : elle résultera 5 événements résultats. Le premier : ENV_TraiterActions qui agit sur le programme et qui regarde la suite des actions. Ensuite l'événement résultat ENV_TraiterModifierCase qui agit sur la classe Case qui modifie (ou non) le passage sur des cases spéciales. Maintenant on traite l'événement résultat ENV_TraiterPositionRobot : agit sur la classe Robot et modifie sa position, donc le Robot se déplace grâce à l'événement ENV_RobotSeDéplace et enfin on affiche tous ces changements à l'écran (classe Appli) avec l'événement résultat ENV_MAJAffichage.



Enfin, le joueur peut décider d'HARD RESET : ce qui signifie qu'il supprime ses programmes en cours et remet tout à zéro (position et cases spéciales). L'événement externe est donc ARR_HardReset qui agit sur le niveau créant ainsi plusieurs événements résultats : ENV_TraiterModifierProgramme agissant sur le programme et supprimant donc toutes les actions. Ensuite ENV_TraiterModifierCase agissant sur la classe Case qui réinitialise toutes les cases à leur état d'origine. Puis ENV_TraiterPositionRobot (agit sur Robot) qui remet à zéro la position du Robot. ENV_RobotSeDéplace et ENV_MAJAffichage permettent ensuite d'afficher tous ces changements.

16) Diagramme de classes



17) Dictionnaire des données

- ✗ next() de Position permet de connaître la position suivante selon une direction passée en paramètre (allant de 1 à 6)
- ✗ operator==() permet de savoir si le robot est sure une Position
- ✗ modifierEtage() de Case permet en mode édition de changer la hauteur d'une case
- ✗ modifierType() permet de changer le type d'une case selon que si l'on passe dessus et on l'allume alors elle change de type(grâce à l'attribut de type type_case qui est une enum)

18) État honnête d’avancement du projet

Actuellement dans le projet, en Programmation, nous en sommes dans une phase d’accommodation avec la SFML et nous faisons apparaître une grille de cases hexagonales en 2D. Notre objectif est donc de trouver une technique pour faire apparaître de la 2D isométrique.