Figure 1 – Slyness

Epita

Rendu de Soutenance 1

Slyness

Leyre Arnaut
Lecomte Paul
Giraud Lise
Aubert Tom

A rendre pour le 13 mars 2020

Table des matières

1 Introduction

Dans ce rendu de soutenance nous allons vous présenter notre avancement dans notre projet de 2ème semestre. Durant cette permière partie du projet nous avons mis en place un gameplay basique mais fonctionel pour notre jeu de puzzle. Ce sont bases qui seront par la suite enrichie et embellies par divers rajouts. Notre première expérience dans le dévelloppement de jeu vidéo a été très amusante et nous sommes toujours très motivés pour la suite de ce projet

Pour rappel voici les objectifs que nous devions atteindre pour la 1ère soutenance :

Tâches	Lise	Arnaut	Paul	Tom
Création mode Créatif	x		X	
Création ennemis/joueur	x			X
Création mode Histoire		X	x	
Création menu		x	X	
Sound Design		x		X
Graphisme		X	x	
Conception des niveaux	X			x
Création du site internet	X			x

Table 1 – Répartition des tâches prévues par le cahier des charges

Tâche/Soutenance	1^{ere} soutenance	2^{eme} soutenance	3^{eme} soutenance
Mode créatif entièrement fonctionnel	X	X	X
Premiers niveaux (tour seulement)	X		
Nouveaux ennemis		X	X
Nouveaux niveaux (dont tutoriel)		X	
Ajouts des derniers ennemis, et du 1er boss			X

Table 2 – Planning tâche/soutenances

2 Avancement du projet à la 1ère soutenance

2.1 Paul

Rappel des objectif de soutenances :

— Création d'une scène créative :
  1. Positionnement Murs
  2. Positionnement Tours
  3. Positionnement Début Fin

— Gestion de sauvegarde et de chargement :
  1. Ancienne Sauvegarde
  2. Nouvelle Sauvegarde

— Changement de scène

Mes objectifs de départ étaient de créer un mode créatif et de créer des niveaux à partir de ce mode de jeu. J'ai donc commencé par créer une scène créative et la rendre fonctionnelle. Par la suite, j'ai cherché à sauvegarder et charger l'emplacement des éléments présents sur un plateau de jeu, tels que les murs, les Tours, etc...
Je suis parvenu à charger une scène "Test" pour pouvoir tester les niveaux réalisés sur la scène créative. Et enfin il me fallait réussir à passer du mode "Test" au mode "Créatif" et inversement de manière simple et efficace.

2.1.1 Créatif :

Ma scène de créatif est composée, d'un dallage de 8 par 6 dalles dédiées au créatif, toutes séparées d'un mur lui aussi dédié au créatif. Les murs extérieurs, qui ne sont pas modifiables font exception et n'apparaissent donc pas.

Tous les murs présents sur ma scène sont abaissés de base et non activés. Si l'on passe la souris par-dessus, ils se montent et apparaissent en transparent. Si l'on clique dessus, il s'active et reste levé et opaque. Sa position est enregistrée dans une liste générale de (double, double, double) qui contient sa position en x, en z et sa rotation en y, respectivement (elle me servira à la sauvegarde de fichier).
Au contraire si le mur est déjà activé, et que l'on clique dessus à nouveau, le mur redescend et on le supprime de la liste générale. Ainsi la liste de positionnement des murs est mise à jour à chaque activation et désactivation de mur.
De plus lorsque je charge la scène, je regarde dans la liste de murs, et pour chaque mur présent, je pré-active ce dernier (cela me sert lorsque je passe de la scène Test à la scène Créatif).

Il y a ensuite les dalles créatives : elles sont vides de base et sont de couleur grise. Grâce au menu du créatif (voir Arnaut : Menus Créatif) il est possible de sélectionner un mode de remplissage de dalles qui est de base sur le mode Tour. Le changement de mode se gère grâce a une variable int qui varie selon la sélection, variant de 1 à 3 avec respectivement le mode Tour, Début et End.

Si l'on est sur le mode Tour, et que l'on passe la souris sur une dalle, elle devient bleue mais redevient de la couleur initiale si l'on quitte la case. De plus si une tour est déjà posée, la case sous cette dernière devient rouge lorsque l'on passe la souris dessus. C'est une manière simple et intuitive de visualiser sur quelle case on est et facilite l'interaction entre le joueur et le jeu.
Si l'on clique sur la dalle, j'Instantiate une préfab de Tour dédié au créatif (elle ne possède pas de hitbox pour faciliter la gestion d'elle-même). Tout comme les murs, j'ajoute donc la position de la tour dans une liste, elle aussi composée de (double, double, double) prenant les même paramètres que celle des murs. Si une tour est déjà posée sur la dalle, on ne peut pas en rajouter, et si l'on fait un clic droit sur la dalle, la tour se supprime, et on la retire de la liste des positions.

En plus de cela, on a la possibilité de faire tourner la tour. De base, lorsqu'on la pose, sa rotation est de 0, et est donc dirigée vers le Nord. Si on a la souris sur la dalle, qu'une tour est déjà posée et que l'on appuie sur la touche Q, on fait tourner la tour de 90° vers la gauche, quelle que soit sa rotation actuelle. Je supprime à ce moment la Tour de la liste de position, pour la remplacer avec sa nouvelle position (je ne change que la rotation de la Tour).
Le processus est le même lorsqu'on l'on appuie sur la touche D, a l'exception de la rotation qui se fait sur la droite. Enfin, lorsque je charge la scène, je vérifie sur chaque dalle (grâce à la liste de Tour), si une tour est située sur cette dalle. Si c'est le cas j'Instantiate une tour directement et j'active la dalle.

Si on est sur le Mode Start, lorsqu'on clique sur une case, j'ajoute la position du Start dans une liste de (double, double, double) qui contient toujours la même chose (Ce n'est pas très optimisé, mais obligatoire pour mon système de sauvegarde actuel). De plus un booléen général (sobrement appelé _startposé) devient true. Ainsi lorsque _startposé est true, il est impossible de poser un autre Start et donc d'ajouter un Start a la liste.
Lorsqu'un Start est posé, la dalle le contenant devient verte. Lorsque l'on réappuie sur la case contenant le Start, tout comme les murs, on supprime la position du Start dans la liste, _startposé devient false, et la dalle redevient normale. Comme le reste, lorsque je charge la scène Créatif, je récupère l'emplacement du Start dans sa liste et j'active la dalle à cette position (Je m'assure tout d'abord que le Start existe grâce à _startposé).

Si on est sur le Mode End, le system agit de la même manière que pour Start, à l'exception de la couleur de la dalle qui est rouge. (J'utilise également une liste pour la position de l'End, ainsi qu'un booléen _endposé)

De plus il y a quelques sécurités :

- Ce script est rattaché au canvas nommé HUD. Il sert juste à initialiser la barre de leurres à chaque lancement de niveau.
- Enfin, lorsque que le Menu Echap est activé (voir Arnaut : Menu Echap), on ne peut ni posé de mur, ni posé quelque chose sur les dalles.

2.1.2 Sauvegarde :

L'étape d'après était celle de la sauvegarde : Ma première idée a été d'enregistrer de manière un peu brutale, via un fichier de Sauvegarde texte. En ouvrant le fichier texte (qui était défini selon un chemin non relatif - la bonne idée) , la sauvegarde s'effectuait ligne par ligne :

— Le script ouvre le fichier ; Le script lisait tout d'abord la liste de position des murs, tuple par tuple et l'écrivait sous la forme *x ; z ; rot* (sans les astérisques) dans le fichier texte

— Lorsqu'il a fini avec les murs, le script sépare les murs des tours à l'aide d'un " # "

— Le script lis alors la liste de positions, de la même manière que la liste de murs puis le script referme le fichier.

— Ensuite, le chargement des données s'effectuait en sens inverse, en utilisant un line.Split(), pour séparer les données contenues dans chaque ligne du fichier.

A cette période, je n'enregistrais pas encore le Start et l'End. Mais cette technique a vite été abandonnée pour de nombreuses raisons :

— Premièrement, le chemin du fichier n'étant pas relatif, la sauvegarde n'était fiable que temporairement.

— De plus un fichier texte est beaucoup trop facilement modifiable, or on ne veut pas que le joueur puisse s'amuser dans les fichiers du jeu.

— Enfin cette sauvegarde était bancale car plus on rajoutait d'éléments, plus le script de lecture du fichier devenait lourd.

J'ai donc décidé de changer drastiquement le système de sauvegarde (même s'il reste encore un peu bancal). J'introduis l'attribut Serializable dans le script de Sauvegarde afin de pouvoir stocker les données de manière plus simple. Nous allons donc créer un fichier dans la mémoire du terminal grâce à Application.persistentDataPath.
Cette propriété en lecture seule nous donne un chemin sur le terminal dédié à l'application. Je crée donc une liste de listes de (double, double, double) contenant toutes les données de position. Pour sérialiser nos données, nous allons passer par un BinaryFormatter. Grâce a cela on sérialise l'objet via le BinaryFormatter directement dans le fichier.
Nos données étant binarisées dans le fichier, il est prêt à être rechargé plus tard. Ce fichier ne sera pas supprimé lorsque le joueur quittera l'application, mais le risque comme abordé plus haut est que le joueur s'amuse à modifier ce fichier et le corrompe tout de même. Et pour charger il nous suffit d'ouvrir le fichier et de dé-sérialiser afin de récupérer toutes les données de positions.

2.1.3 Passage entre les modes :

Enfin, pour pouvoir jouer au niveau que le joueur crée, il me fallait faire un passage, entre la scène créative et la scène de jeu de Tom.

Pour charger une scène j'utilise directement le SceneManager de Unity qui me permet de Load une scène en l'appelant par son nom. Mais pour passer d'une scène à l'autre en conservant les positions de chaque élément, il nous faut donc utiliser le script de Sauvegarde et de Chargement pour pouvoir charger tous les éléments.

Tout d'abord pour passer du mode créatif au mode test, il nous faut vérifier qu'un Start et un End sont présents sur le terrain, puis ensuite sauvegarder l'entièreté des listes de positions. Enfin on charge la scène de Test. Lorsque que la scène s'initialise, cette dernière initialise toutes les listes a 0, puis charge l'ensemble des listes et change leur valeur. Le GameManager (voir Tom : GameManager) initialise la position initiale du joueur à l'emplacement du Start. Ensuite j'Instantie les Murs dédiés au test, ainsi que les préfabs de Tour, créés par Tom. Enfin on réinitialise les leurres (voir Lise : Bouton Restart) afin d'éviter les problèmes de liste liés aux leurres.

Pour passer du mode Test au mode créatif, on se contente de charger la scène Créatif, et lorsque cette dernière s'initialise, elle remet ses listes à 0, charge la Sauvegarde, et initialise le mode créatif au mode Tour. Les dalles et les murs s'initialisent tout seuls en tant qu'activés ou non comme vu plus haut.

2.2 Tom

Pour cette soutenance, je devais avoir codé tous les objets basiques nécessaires au fonctionnement du jeu : du personnage contrôlé par l'utilisateur, aux tours qui bloquent la progression du joueur.

Résumé des objectifs avant cette soutenance :
— faire un player avec colliders
— faire la camera
— faire la tour
— faire le système de leurre

2.2.1 Player

Le principal objet à implémenter est le Player (ou joueur), une petite sphère rouge. Nous voulions que ce dernier puisse se déplacer librement dans l'espace (sauf en hauteur), et qu'il ne puisse pas traverser les murs ou les tours.

Après plusieurs essais afin d'obtenir un mouvement fluide, j'ai choisi d'utiliser Input.getKeyDown et Input.getKeyUp pour activer 4 booléens. Chacun d'eux représentent une direction (up, down, left, right), et sont mis sur « true » lorsque la touche liée à leur déplacement est pressée (KetKeyDown), et inversement. Par exemple, appuyer sur la touche « z » mettra MoveUp sur true et ainsi de suite.
Dans le cas d'un mouvement en diagonale (lorsque deux booléens de mouvement sont sur true en même temps), la vitesse est divisée par la racine de 2 afin de conserver l'homogénéité de la vitesse.
Cette technique permet notamment d'éviter d'activer 2 mouvements contraires en même temps (lorsqu'un mouvement est activé, on peut mettre false sur le booléen du mouvement opposé), et de gérer plus facilement la rotation de l'objet.

Une fois les mouvements réalisés, il faut ajouter un système de détection, pour ne pas que le joueur rentre dans les murs/tours. J'ai choisi de le réaliser à l'aide de 3 raycasts, qui sont lancés dans la direction des touches pressées ces derniers permettent d'annuler le mouvement dans la direction ciblée.

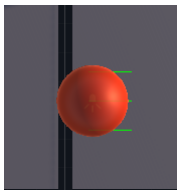

FIGURE 2 – test de collision engendré par un mouvement à droite

Le mouvement s'opère donc de la manière suivante :
Pour les 4 booléens de directions :
si le booléen est sur true, et que les raycasts dans la direction du booléen ne détectent rien : on déplace le joueur dans la direction ciblée avec une vitesse = (vitesse de base / réduction de la vitesse)
La rotation du joueur s'effectue par rapport aux touches pressées, et s'adapte a la rotation de la caméra (voir plus bas).

2.2.2 Camera

Le deuxième objet à implémenter est la Caméra, qui doit pouvoir tourner sur elle-même, rester au milieu du terrain (mode statique) ou suivre le joueur si l'utilisateur le désire (mode follow).

La camera est enfant d'un gameObject "GameCamera", et c'est ce gameObject, centré par rapport au terrain qui se déplace et tourne ; la camera se déplace donc par rapport à lui, mais c'est surtout l'aspect "centre de rotation" qui est extrêmement intéressant.

Il faut noter que lorsque le GameCamera tourne, les déplacements du joueur dans l'espace changent en fonction des touches pressées.Par exemple, tourner la camera de 90 degrés fait que la touche S avance le joueur sur l'axe x et non plus z.

C'est pourquoi les Vecteurs de déplacements, ainsi que les angles de rotations du joueur sont stockés dans des listes. La rotation de la caméra augmente ou diminue un index présent dans le code du joueur. Cet index cycle de 0 à 3 compris. Grace à ces listes et l'index, je peux récupérer les mouvements et les rotations adaptés à la caméra. Autre point : la vitesse de la Camera est augmentée lorsque le joueur meurt, ou lorsque l'on passe du mode statique au mode follow, pour rejoindre le joueur plus rapidement.

2.2.3 Tours

Nous avons notre joueur et notre camera, le prochain objet à créer est donc la tour. Celle-ci doit voir si un leurre est dans son champ de détection, et se retourner dans cette direction pendant quelques secondes. Elle doit surtout détecter si le joueur n'est pas dans son champ de vision, et dans le cas contraire le tuer.

Pour détecter les leurres, la tour lance un raycast dans chaque direction (devant, derrière, droite, gauche) si l'un d'entre eux détecte un leurre, la tour le détruit puis se retourne dans la direction du leurre. Elle revient à sa position initiale au bout de 2-3 secondes.

Pour détecter les leurres, un seul raycast suffit amplement car ces derniers sont forcément situés au milieu d'une case (nous verrons cela juste en dessous), mais ce n'est pas le cas du joueur, qui peut se déplacer librement dans la scène.

J'utilise donc 3 raycasts (voir annexe n° blablabla) qui vont couvrir une bonne partie du champ de vision. On peut remarquer par ailleurs que la hitbox du joueur est plus grosse que son corps, ce qui est nécessaire pour que le joueur soit bien détecté s'il fait l'erreur de trop se rapprocher du champ de vision.

Dans le cas où le joueur est repéré, il est bloqué pendant 2 secondes, la tour lance une animation (pour bien faire comprendre à l'utilisateur la raison de son échec), puis le joueur revient à sa position d'origine (stockée dans le script GameManager).

2.2.4 Leurres

Le dernier objet à réaliser était le LeurreManager, afin de gérer les 3 leurres d'un coup. Ce dernier se sert d'un tableau (int, float, float)[3] (le int est le niveau d'activation, les deux floats représentent respectivement la position sur l'axe x et la position sur l'axe z).
On a donc 3 slots de leurres disponibles.
Le niveau d'activation est sur 0 pour une absence de leurre, sur 1 pour un leurre posé non actif, sur 2 pour un leurre actif .

Poser un leurre :
Lorsque la touche « espace » est pressée, un raycast est lancé sous le joueur. Celui-ci va détecter la case sur lequel le joueur se situe. Dans le cas où aucun leurre ne se situe sur la case (vérifié grâce aux floats du tableau) et que au moins 1 des trois éléments du tableau possède un int à 0, un leurre non actif est posé au milieu, et le slot libre du tableau prend en valeur (1, position sur l'axe x du leurre, position en z du leurre).

Activer un leurre :
La touche J active le leurre du premier élément du tableau, et il en va de même avec la touche k qui active le 2ème élément, ainsi que la touche L qui active le 3ème élément. Lors de l'activation, le leurre non actif est Détruit, et un leurre actif est Instancié aux mêmes positions. Son Niveau d'activation passe alors à 2 dans le tableau. A noter que le leurre Actif meurt automatiquement au bout de 3 secondes, s'il n'est pas détruit avant par une tour. Le leurre actif est détecté par la tour car il possède le tag « piège ».

Mort d'un leurre Actif :
Une fois mort, le niveau d'activation du slot est mis a 0, tout comme les positions en x en z. Un nouveau leurre pourra alors être posé.

2.2.5 Difficultés rencontées

Outres les nombreux essaies infructueux, il y a eu enormement de debuggage a effectuer, et d'ameliorations apporté aux codes/technique pour réaliser ces objets. Il reste néanmoins certains bug a corriger, et tres surment de nombreux bugs que nous n'avons pas decouverts ;

2.2.6 Bonus :

J'ai aussi réalisé l'animation de tir de la tour, ainsi que les lévitations qui fonctionnent avec la fonction sinus.

2.3 Arnaut

Ma tâche était de créer en coopération avec Lise les menus et interfaces du jeu. Le but était d'obtenir les menus permettant de se déplacer entre les différent modes de jeu, rapidement afin de faciliter la compilation des différentes scènes. Tout ces Menus ont été créés à partir de l'outil Canvas de Unity, ce dernier permet de positionner un objet en 2D devant une camera en 3D en adaptant tous les fils du Canvas en fonction de la taille d'écran de l'utilisateur.

2.3.1 Main menu

Le Menu principal comporte les fondations des futures options qui seront disponibles dans le jeu. Ce Menu comporte 4 boutons, j'ai tenté de recoder les boutons pour éviter d'utiliser ceux fourni par Unity cependant il s'avère que les objets dans un Canvas sont incapables de détecter la position de la souris par rapport à la leur. J'ai donc utilisé les UI Button de Unity. Le bouton « Quit » est un simple « Application.Quit() ». Les boutons Campain, Creative et Settings se servent de la fonction SetActive(Bool b) pour changer entre les différents panel contenant les sous-menu (Voir MainMenuManager.cs).

FIGURE 3 – MainMenu

C'est avec les sous-menus que le code commence à être intéressant les codes LvlButton.cs et SavedLvlButton.cs servent respectivement dans le menu campain et le menu Creative, contiennent une variable publique lvl qui peut être modifiée dans l'inspector de Unity. Cela permet de définir un Lvl depuis Unity et d'adapter tout le code à partir de ça. Ainsi lors du lancement de la scène main menu les boutons affichent leur lvl et lors de leur activation ils chargent la sauvegarde dans le chemin : appdata + le mode + lvl (voir Paul save).

2.3.2 Creative menu

Je me suis également occupé des interfaces du mode créatif, ce qui comprend un menu pause, un bouton permettant de passer au mode test et un menu contenant les différentes structures positionnable sur le terrain.

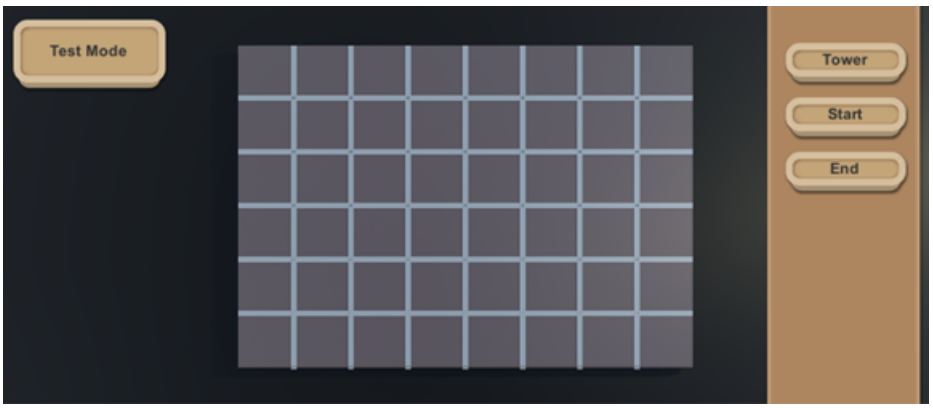

FIGURE 4 – Creative interface

Le bouton « test mode » vérifie la présence d'un début et d'une fin de niveau, sauvegarde le niveau en court et charge la scène Histoire. Pour l'instant il ignore le nom du niveau et charge uniquement la sauvegarde unique sur laquelle le jeu tourne. Cependant Paul et moi avons déjà travaillé et commencé à modifier les fonctions pour accueillir un jeu multi-sauvegarde. Le menu défilant bien que cette fonctionnalité soit pour l'instant peut utile, est le script qui m'a posé le plus de problèmes. En effet le fait que transforme.Position soit en 3 dimensions m'a laisser penser que le vect3 obtenu était la position dans le monde. J'ai donc tenté 2 alternative pour palier a ce problèmes.

1. Crée un nombre défini de petit mouvement ; problèmes avec impossible de corrigé le dépassement du a la distance, qui se cumulait.
2. Utiliser la position par rapport à l'écran ; problèmes ça change selon

les ecrans Par la suite je me suis rendue compte que je m'était trompé en que transforme.Position donner la position relative au Canvas. Par la suite il a suffi de modifier, à l'activation du bouton, le public int qui permettre de savoir quelle structure poser dans le code de Paul L'activation Menu pause quant à elle place un calque noir devant la caméra et stop le temp dans du jeu. Ce qui n'arrête cependant pas les détections telle que le OnMouseOver() il a donc fallut retirer manuellement la possibilité de poser des éléments en fond. Le bouton Save réutilise le script Save de Paul et Option affiche un menu qui permettra de changer les controles par la suite

2.4 Lise

La réalisation d'un HUD (Head Up Display) a été effectuée. Cet HUD n'est pas fini et évoluera au fil de l'avancement du projet. Pour le moment il comprend trois éléments : un chronomètre qui affiche combien de temps s'est écoulé depuis le lancement du niveau, un bouton restart qui réinitialise le niveau et fait recommencer le joueur depuis le début de ce niveau et une barre de leurres qui représente l'état des 3 leurres que l'on peut poser simultanément pour chaque niveau. Pour réaliser les différents composants de ce Head Up Display, l'utilisation Unity, Rider, et d'autres logiciels pour réaliser les graphismes (qui sont provisoires pour le moment) a été nécessaire.

2.4.1 Barre de leurres :

Tout d'abord, il a fallu réaliser une image contenant 3 cercles de couleurs différentes que j'ai faite sur Inkscape. Les trois couleurs de ces cercles sont le gris, le bleu et le rouge. Un cercle gris dans la barre de leurres correspond à un leurre non utilisé, un cercle bleu signifie un leurre posé mais non actif et un cercle rouge montre un leurre posé et actif.

Une fois l'image réalisée, je l'ai intégrée en tant que Sprite dans le projet Unity et j'ai utilisé le Sprite Editor de Unity pour couper l'image en trois et pouvoir ajouter chaque cercle dans la scène de façon distincte.

Dans la scène en cours de Unity, j'ai créé un canvas (nommé HUD) qui a une taille qui se cale automatiquement à la taille de l'écran utilisé avec une résolution 1920 par 1080 pixels. A ce canvas j'ai ajouté un panel sur le côté gauche. Sur ce panel j'ai mis le sprite appelé Panel pour mettre une image de fond sur ce panel. Toujours sur le panel, j'ai ajouté 3 panels avec 1 cercle gris à l'intérieur pour chaque panel qui correspondent à la barre de leurres (LeurZone dans Unity). Cette barre correspond à la limite du nombre de leurres que l'on peut poser simultanément. Dans le jeu nous avons décidé de fixer cette limite à 3 leurres simultanés.

Toutefois, le nombre de leurres par niveau est illimité : si trois leurres sont posés en même temps mais qu'on en active un, il disparait de la barre de leurres et une nouvelle place est libérée pour poser un leurre supplémentaire.

Tout ceci est géré dans le script C# appelé TrapBar.cs. Dans ce script j'utilise une public array (traps) qui contient les images de chaque leurres, trois public sprite qui correspondent à chaque image de leurre, et une 2ème array (listeleurre), publique également en static qui contient l'état de chaque piège (0, 1 ou 2) et deux floats qui correspondent aux coordonnées du leurres dans le jeu. Cette dernière n'est pas utilisée uniquement dans TrapBar.cs mais aussi dans GameManager (script réalisé par Tom Aubert).

Ensuite, ce script utilise une méthode que j'ai appelé Load et qui se contente d'assigner la bonne couleur de cercle dans la barre de leurres en fonction de l'état de celui-ci dans l'array listeleurre. Ainsi, quand un leurre est posé ou activé, le leurre correspondant dans la barre de leurre (TrapBar dans le panel) change de couleur.

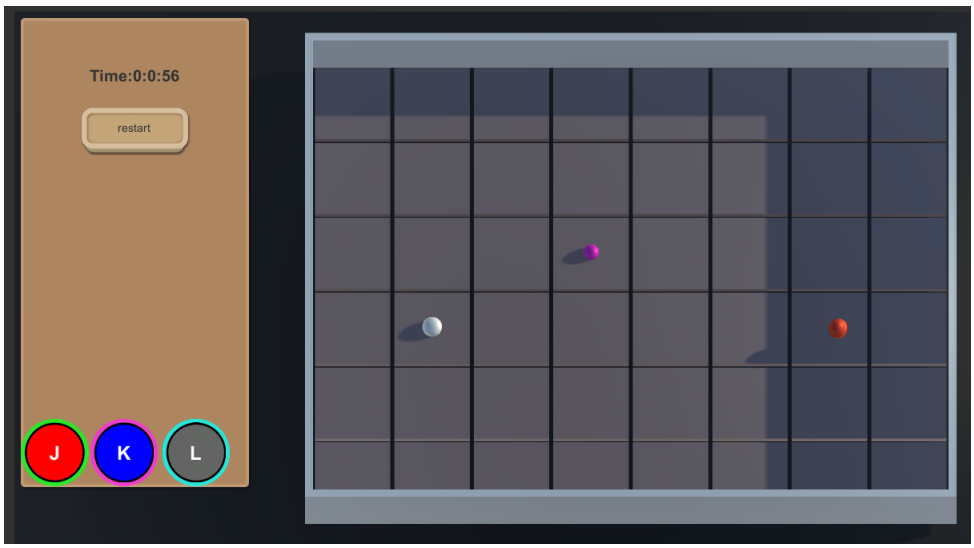

FIGURE 5 – Barre de leurre

Sur l'image ci-dessus, la barre de leurres est représentée par les 3 cercles rouge, bleu et gris.
Les lettres J, K et L sont les touches du clavier à utiliser pour activer les pièges quand ils sont posés.
Ainsi, sur l'image, la touche L ne peut pas être utilisée car dans la barre de leurre, il n'y a aucun leurre assigné à cette touche à cet instant (le cercle est gris).
La boule rouge correspond au joueur, la boule blanche correspond au cercle rouge (et à la touche J) dans la barre de leurres.
Ainsi la boule blanche est un leurre actif (tous les leurres actifs seront représentés en blanc dans le jeu et en rouge dans l'HUD).
Enfin, la boule violette correspond au cercle bleu (et à la touche K) dans la barre de leurres. Ainsi le leurre est posé mais non actif (tous les leurres posés mais non actifs seront représentés de la même couleur que le cercle qui leur correspond dans l'HUD (ici le cercle K est contouré en violet donc le leurre qui lui est assigné est violet) et en bleu dans l'HUD).
Pour l'activer, il faut tout simplement appuyer sur la touche K. Nous sommes bien conscients que les couleurs ne sont pas intuitives, elles seront modifiées avec les graphismes par la suite.

2.4.2 Le chronomètre :

Toujours dans le canvas et sur le panel j'ai créé un nouvel UI Text (appelé Timer). Sur cet UI, j'ai ajouté le script Timer.cs. Dans ce script, j'utilise les méthodes Start et Update afin de lancer le chronomètre dès qu'un niveau est commencé et de convertir le temps obtenu (en secondes) en minutes et en heures. J'ai décidé de ne garder aucune décimale dans l'affichage des secondes par souci de place (l'affichage des décimales prend de la place dans l'HUD sur le bord de l'écran et n'est donc que peu esthétique et lisible)

2.4.3 Le bouton restart :

Toujours dans le canvas et sur le panel, j'ai créé un UI Button (nommé RestartButton) auquel j'ai ajouté le sprite appelé Icon. Dans le jeu, ce bouton s'appelle Restart.

J'y ai ajouté le script appelé RestartButton.cs. Ce script est utilisé dans plusieurs cas. Tout d'abord, il sert pour réinitialiser le niveau en cours si le joueur décide de le recommencer pour une raison qui lui est propre (s'il est coincé par exemple). Il sert également dans le cas de la mort du joueur, par exemple si celui-ci se fait détecter par un ennemi.

Ce script est donc appelé dans d'autres script, d'où l'utilisation de méthodes en static. Dans tous les cas, le script réalise les mêmes actions : il enlève tous les leurres posés dans le niveau, remet la barre de leurres à 0 et renvoie le joueur à sa position de départ dans le niveau.

Pour se faire, j'utilise 3 méthodes : Update, Fake, et Restart. Les méthodes Fake et Update sont les mêmes à un point près : la méthode Fake n'est pas en static, contrairement à la méthode Update tout simplement parce que les méthodes statiques ne fonctionnent pas avec les UI Button si on veut les utiliser à la souris.

De son côté, la méthode Update fonctionne très simplement : si le joueur appuie sur la touche F9 de son clavier, le niveau se réinitialise. Toutefois, le bouton fonctionne également avec la souris, en cliquant dessus. La méthode Fake est utilisée pour faire fonctionner le bouton grâce à la souris, comme expliqué précédemment. Enfin, la méthode Restart qui est appelée dans Fake et Update réalise toutes les actions nécessaires à la réinitialisation du niveau.

2.4.4 Le HUD manager :

Ce script est rattaché au canvas nommé HUD. Il sert juste à initialiser la barre de leurres à chaque lancement de niveau.

2.4.5 Difficultés rencontrées :

Au début de ce projet j'ai rencontré beaucoup de difficultés liées à l'utilisation de Unity.

En effet, je n'avais jamais utilisé ce logiciel ni aucun de ce type et j'ai donc eu besoin d'un temps de prise en main assez important pour réussir à faire ce que je voulais. En raison de ce temps de prise en main j'ai passé plusieurs heures pour réussir à faire la barre de leurres, notamment pour réussir à faire un sprite qui coupe mon image avec mes trois cercles en 3 cercles distincts.
J'ai également rencontré des problèmes lors de la gestion des scènes, et des GameObject.
Concernant les scripts C#, je n'avais pas compris tout de suite le fonctionnement de la méthode Update et je ne l'utilisais donc pas correctement

Une fois toutes ces difficultés surmontées, j'ai pu commencer à vraiment travailler sans devoir repasser sans cesse sur ce que j'avais déjà fait pour corriger mes erreurs et retirer des éléments inutiles.

3 Ce qu'il nous reste à faire, retard :

Globalement, nous avons pris du retard sur la création des niveaux (nous n'en avons pas créé un seul alors qu'il devait y en avoir au moins un). Mis à part ce léger retard, nous devons apporter des améliorations globales sur nos divers scripts et graphismes : entre autres, graphisme, divers menus, système de sauvegarde.

Nous devons commencer à travailler sur la conception du site, du son et des niveaux.

Nous devons aussi ajouter de nouveaux ennemis pour la deuxième soutenance. Nous pensons qu'il sera difficile de réaliser la conception complète du tutoriel pour la deuxième soutenance.

Ci-dessous se trouve une mise à jour des tableaux de réalisation des tâches et du planning des tâches pour chaque soutenance.

Tâches	Lise	Arnaut	Paul	Tom
Sound Design				X
Graphisme		X		
Conception des niveaux			X	
Création du site internet	X			
Création de nouveau ennemis	X	X		X
Initialisation du tutoriel			X	

Table 3 – Répartition des tâches

Tâche/Soutenance	1^{ere} soutenance	2^{eme} soutenance	3^{eme} soutenance
Mode créatif entièrement fonctionnel	X	X	X
Premiers niveaux (tour seulement)		X	
Nouveaux ennemis		X	X
Nouveaux niveaux (dont tutoriel)		X	
Ajouts des derniers ennemis, et du 1er boss			X

Table 4 – Planning tâche/soutenances

4 Manuel d'utilisation du jeu :

Le jeu est destiné a être joué au clavier, néanmoins la souris peut être utilisée à certaines occasions. C'est pourquoi cette partie tient lieu de manuel d'explication à l'utilisation du jeu.

4.1 Les touches du clavier :

- Z : Déplace le joueur vers le haut.
- Q : Déplace le joueur vers la gauche (en mode histoire) ou tourne la tour de 90°C vers la gauche (en mode créatif).
- S : Déplace le joueur vers le bas.
- D : Déplace le joueur vers la droite (en mode histoire) ou tourne la tour de 90°C vers la droite (en mode créatif).
- F9 : Réinitialise le niveau.
- Up : Scroll vers le haut dans le menu créatif.
- Down : Scroll vers le bas dans le menu créatif.
- Espace : Dépose un leurre sur la case actuelle.
- J : Active le leurre assigné à cette touche.
- K : Active le leurre assigné à cette touche.
- L : Active le leurre assigné à cette touche.
- U : Tourne la caméra vers la gauche.
- I : Tourne la caméra vers la droite.
- O : Bloquer ou débloquer la caméra.
- C : Passer du mode créatif au mode test et inversement.
- Echap : Ouvrir le menu.

De plus, comme indiqué précédemment, la souris peut servir pour cliquer sur les boutons des différents menu au lieu d'utiliser le clavier.