

coarv - le labirynthe du minotaure

Valentine Kempf, Salwa Aberkane, Victor Duvivier
Esteban Lecourt & Thomas Le Bourhis

pour le 30/03/20



<https://gitlab.com/vicco-rgb/labi-minotaure.git>

Contents

1 Cadre du jeu	2
1.1 Progression du jeu	2
1.2 Environnement virtuel	3
2 Fonctionnement du jeu	3
2.1 Version Desktop	3
2.1.1 Interaction Raycast	3
2.1.2 Énigme de la table	4
2.1.3 Énigme des diamants	7
2.1.4 Énigme du poème	8
2.1.5 Niveau 2	9
2.2 Version VR	10
2.2.1 Interaction Raycast	10
3 Bilan	11

1 Cadre du jeu

1.1 Progression du jeu

Notre objectif sur ce projet d'escape game est de terminer le niveau 1 commencé l'année dernière, et d'ajouter un deuxième niveau. Le niveau 1 garde le même scénario que celui prévu. Les énigmes à suivre sont :

salle 1 : énigme de la table Il faut reproduire un tableau en plaçant les objets sur la table comme ils le sont sur le tableau. Nous avons ajouté le Raycast au script associé à cette énigme pour permettre l'interaction en environnement virtuel. Quand les objets sont bien placés, le mur se détruit.

salle 2 : énigme du diamant Il faut récupérer un diamant par terre et le placer dans l'oeil droit d'un crâne de vache accroché au mur (il y a déjà un diamant dans l'oeil gauche). Ceci permet de débloquer l'énigme suivante. L'objectif du groupe précédent était qu'à ce moment là, un esprit sorte de la mâchoire du crâne, les lumières des torches deviennent vertes et un poème tombe sur le sol.

salle 2 : énigme du poème Il faut suivre les différentes étapes du poème pour allumer les torches dans le bon ordre avec le Raycast. Ceci permet d'accéder au niveau suivant, une porte s'ouvre alors.

L'objectif initial était d'allumer les torches à l'aide d'un miroir qui réfléchit un rayon de lumière entrant dans la pièce, mais nous n'avons pas eu le temps d'implémenter une telle interaction. L'interaction est réalisée avec la souris en cliquant sur les torches.

niveau 2 Ce niveau n'a finalement pas été réalisé par manque de temps. Nous pensions générer aléatoirement un labyrinthe avec plusieurs sorties possibles vers des salles à énigmes, reliées entre elles dont une seule est la véritable sortie du jeu. Il existe des générateurs de labyrinthe dans l'Asset Store Unity, nous avions commencé à faire le design de ce niveau, mais les fichiers sont restés sur l'un des ordinateurs de l'école.

1.2 Environnement virtuel

Le groupe précédent a réalisé l'environnement virtuel. L'ambiance est lugubre, puisqu'il s'agit d'une attraction de parc fermé la nuit. Pour cela, l'ambiance sonore est très importante. Il y a donc une musique lugubre avec un effet reverb pour mettre le joueur mal à l'aise et donner un certain vertige. Des cris spatialisés du minotaure ont été programmés aléatoirement (toutes les 5 ou 10 secondes).

Au niveau visuel, le joueur est placé dans un EV avec un décor "antique" lugubre avec des statues, glaives, crânes, toiles d'araignées, coffre et peu éclairé par des torches pour l'ambiance "escape game" mystérieux. Le décor est en ruine, il y a des tas de gravats au milieu de la pièce, de la végétation sauvage et de la poussière tombe du plafond.

Nous avons supprimé le minotaure derrière la porte puisqu'il ne bougeait pas et que l'environnement nous semble déjà suffisamment anxiogène. Nous avons ajouté la poussière qui tombe du plafond. L'objectif du groupe précédent était que le minotaure tape sur les murs et fasse tomber de la poussière, ils avaient donc commencé le prefab de poussière, mais cette tâche leur a pris beaucoup de temps.

2 Fonctionnement du jeu

2.1 Version Desktop

2.1.1 Interaction Raycast

Nous avons choisi d'implémenter une interaction avec les objets en utilisant le Raycast qui part de la caméra vers la position de la souris. Et pour faire mieux, la position de la souris est visible dans la scène sous forme d'une petite sphère pour qu'on puisse la repérer.



Figure 1: position où pointe la souris

Le script qu'on a écrit est standard et qui contient les méthodes dont on aura besoin pour toutes les énigmes.

GetObject() qui permet de retourner le gameobject détecté par le Raycast. Tous les objets avec lesquels le Raycast peut interagir doivent être configuré sur le Layer "Interactable".

GetHitPosition() qui retourne la position du gameObject intersecté par le Raycast.

GetPointerPosition() qui retourne encore la position de l'intersection de Raycast avec le gameobject.

2.1.2 Énigme de la table

Nous avons repris le scénario qui a été présenté par les anciens.

”Dans l’entrée de la salle 1 se trouve la table ci-dessus. Des accessoires sont posés dessus : un crâne, une épée, un vase et une statuette de discobole. En face de lui, derrière les gravats, le joueur peut apercevoir un tableau sur le mur, qui représente cette table, avec une disposition précise des objets. L’objectif est de reproduire cette disposition avec une précision suffisante pour pouvoir débloquer la suite du jeu.

La réussite ou non de l’énigme de la table est gérée par le script `Disposition.cs`. Ce script a pour attribut public une liste de `Vector3` correspondant aux positions locales visées pour les GameObects enfants

de l'objet auquel le script est attaché. Autrement dit, pour le bon fonctionnement de l'énigme, il faut associer le script `Disposition.cs` au GameObject `Table`, et s'assurer qu'au cours du jeu, les accessoires à placer restent bien des enfants dans la hiérarchie Unity (ce qui est notamment à prendre en compte pour le développement des techniques d'interaction), et que l'ordre de ces sous-objets reste bien le même. Lorsque les positions locales de tous les enfants de la table sont suffisamment proches (à `tolerance` fixé près) des positions `objectifs` (attributs privés), le booléen `goal` prend la valeur `true`.

Aucune vérification des objectifs n'était implémentée. Nous avons donc dû ajouter un script qui à chaque frame vérifie la position de chacun des objets disposés sur la table. Pour cela, le script vérifie si la distance entre la position de l'objet et son objectif est inférieure à la tolérance définie au préalable. Si tous les objets vérifient cette condition alors on peut lancer la transition vers la deuxième énigme.

Pour gérer les déplacements des objets nous avons utilisé le raycasting pour sélectionner l'objet à déplacer avec un clic maintenu sur la souris. Lorsqu'un objet est ainsi sélectionné, le script détecte le point de collision entre le rayon et la table et déplace l'objet sélectionné à cet emplacement.

Nous avons implanté notre code de telle manière que nous pouvons choisir combien d'objet sur les 4 nous devons bien placer avant de passer à l'éénigme suivante. On a alors changé la tolérance pour qu'elle prenne en compte le nombre d'objets considérés.

```
nItemConsidered = Mathf.Min(3, transform.childCount);

objectifs.Add(new Vector3(0.3f, 0.7f, -0.2f)); //vase
objectifs.Add(new Vector3(-0.1f, 0.7f, 0f)); //sword
objectifs.Add(new Vector3(0.1f, 0.7f, 0.2f)); //discobole
objectifs.Add(new Vector3(-0.4f, 0.7f, 0.1f)); //skull

tolerance = 0.02f*nItemConsidered;
```



Figure 2: texture du tableau dans la scène et réalisation du positionnement

Une fois on résout cette énigme (dans le script EnigmeManager attaché au gameobject Player), le mur se détruit pour aller vers la 2eme salle. Cet effet est réalisé en supprimant le mur ainsi que son collider. Pour rendre l'effet plus spectaculaire, on rajoute une émission de particules avec comme particule élémentaire des cubes ayant la forme de briques.



Figure 3: particules briques

2.1.3 Énigme des diamants

Nous avons repris pour cette énigme l'ancien principe de base mais avons tenté de varier l'utilisation du script `MissingEye` pour l'aide au positionnement de l'objet de l'énigme (un oeil en diamant). Nous avons également gardé le scénario.

"L'énigme se présente ainsi: dans une pièce secrète, se trouve un crâne bovin avec un diamant en guise d'oeil gauche, l'autre orbite étant vide. Le joueur doit donc rechercher cet autre diamant pour le placer à sa place: sur l'orbite vide. Lorsque le diamant est suffisamment proche de l'orbite, il sera automatiquement attiré par sa place pré définie au sein de l'orbite à l'aide d'un customizable joint."

Le script d'aide au positionnement est attaché à l'oeil "manquant", soit l'oeil qui est caché dans le décor.

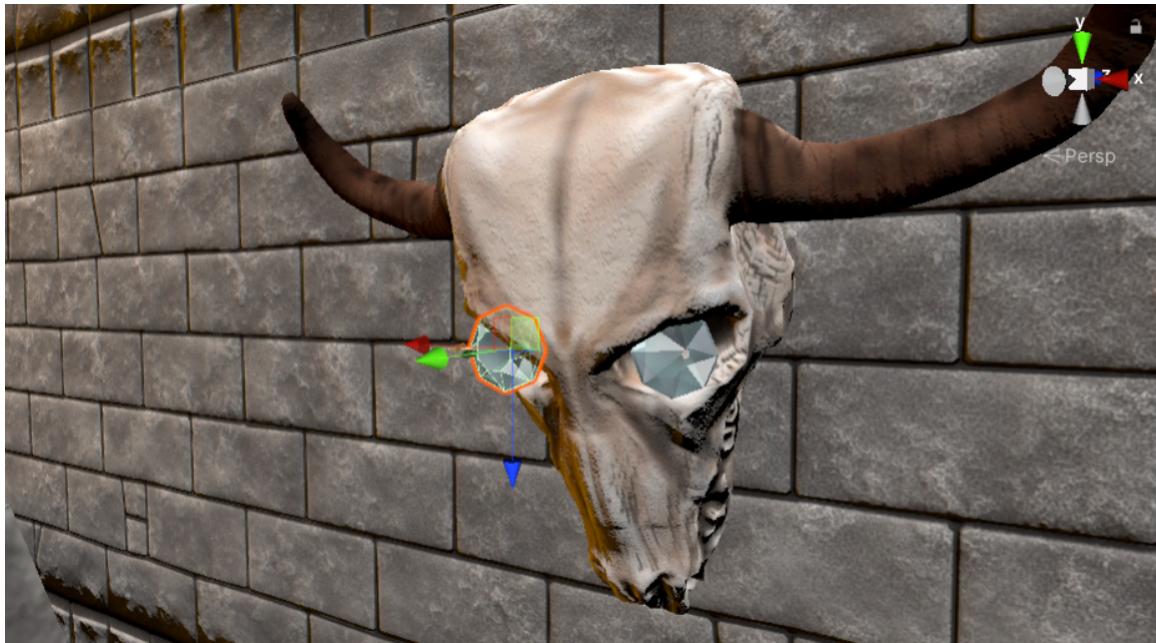


Figure 4: oeil *manquant*, ici dans son emplacement

On commence par positionner un 3ème oeil au bon endroit (au niveau du Eye-Socket), qu'on rend inactif et qu'on cache donc pour le moment. La position de l'oeil est basée sur les données de position qu'on récupère dans la scène lorsqu'on place l'oeil au bon endroit. Cependant celles-ci demandent de se placer dans le bon repère : selon si on est dans celui de la scène ou de l'objet parent, on n'obtient pas le même résultat, ce qui rend compliqué l'obtention de l'endroit exact où l'on veut placer l'objet.

Il faut ensuite sélectionner l'oeil manquant et on se sert pour ça du raycasting pour sélectionner l'objet à déplacer avec un clic maintenu sur la souris. Lorsque la

position de l'objet passe sous une distance minimale de la position de l'emplacement d'arrivée, on veut pouvoir l'y placer directement en le lâchant.

Nous avons essayé plusieurs méthodes ici. La première était celle utilisée l'année précédente, où à l'aide d'un joint on venait forcer la fixation de l'objet à son emplacement prévu. Cependant les tests n'étaient pas fructueux et le manque d'explications ne permettaient pas de poursuivre sur cette voie.

Nous avons donc tenté de modifier simplement la position de l'objet tenu dans les mains lorsqu'il entrait dans la zone de fixation automatique. Mais ici nous avons rencontré beaucoup de problèmes liés à la position voulue de l'oeil et aux repères (comme expliqué précédemment).

La dernière solution essayée fut celle qui utilise le 3ème oeil créé en début de script. En effet, on vient cacher cet oeil et on manipule celui sur lequel est appliqué le script et qui est, lui, caché dans la pièce à l'origine. Dès lors que l'oeil manipulé entre dans la zone de fixation automatique, on rend actif le 3ème oeil et on supprime celui qui est manipulé, donnant l'illusion d'une fixation "magnet".

Cependant le résultat final permettait bien d'aller jusqu'à la destruction de l'oeil dans la zone souhaitée, mais l'oeil caché n'apparaissait pas sur le crâne. Il reste donc des problèmes dans la gestion de la position finale.

2.1.4 Énigme du poème

Nous avons implémenté le script "TorchInteraction" pour assurer l'interaction de l'utilisateur avec les torches. On a toujours gardé le même scénario déjà prévu.

*Trois frères et une sœur
Eurent à traverser.
L'aîné, de tout son cœur,
Avança le premier.
En prudent guerrier,
Le cadet se tint prompt.
Confiant du premier,
Le dernier prit second.
Grand fut l'étonnement,
Quand la sœur vit son choix,
Profitant de l'instant,
Au dernier les doubla.*

Il correspond en réalité à un ordre d'allumage particulier des torches, qui sont situées à côté de chaque statue. Chaque personnage du poème correspond à une des statues. Les torches sont allumées à l'aide d'un miroir et d'un faisceau de lumière

arrivant du plafond. C'est à l'utilisateur de bien positionner le miroir pour faire coïncider le faisceau sur la torche pour l'allumer. Cette énigme utilise pleinement les interactions RV en ce qui concerne la position et l'orientation dudit miroir !



Figure 5: Énigme du poème pour allumer les torches dans le bon ordre

Dans la scène, on a numéroté les torches selon leur ordre d'allumage pour que l'énigme soit résolue.

L'interaction avec les torches se fait encore avec le Raycast, au moment on clique et qu'on pointe bien sur la torche qu'on souhaite allumer, cette dernière active son **PointLight** et donc s'allume. Cliquer sur une torche qui est déjà allumée éteint toutes les autres torches et donc l'utilisateur est amené à tout refaire.

Il faut aussi savoir que les Point Light sont dans un Layer **Interactable** puisque encore une fois, le Raycast n'interagit qu'avec les objets dans ce Layer.

Une fois l'énigme est résolue, on regarde que la variable **EnigmaResolved** se coche dans la scène. Le script de cette énigme est attaché au gameobject **EnigmeTorches**. Quand on résout cette énigme, la porte s'ouvre pour aller au deuxième niveau.

2.1.5 Niveau 2

Pour le niveau 2, nous avions commencé à réaliser l'EV. Il s'agit d'un labyrinthe généré aléatoirement à chaque entrée dans la pièce, pour que l'on soit réellement perdu. Le choix de la technique de déplacement sera donc très importante puisque les distances à parcourir seront grandes, et il faut que le jeu reste dynamique et intéressant.

Nous avons trouvé sur l'Asset Store un projet de labyrinthe aléatoire, avec plusieurs cul de sac contenant des pièces à récupérer. L'objectif est de mettre des portes à la place de ces pièces, qui mènent vers des salles à énigmes, et amènerai vers une autre porte en sortie. Après avoir testé toutes les salles du labyrinthes, on atteint la dernière porte qui mène à l'écran de fin. Malheureusement, nous n'avons pas récupéré

les fichier du niveau 2, il est donc libre au groupe suivant de le réaliser entièrement, ces idées ne sont que des suggestions, mais nous vous encourageons à les développer. Lien vers l'asset de labyrinthe aléatoire gratuit :

<https://assetstore.unity.com/packages/tools/modeling/maze-generator-38689>

Il faut bien sûr revoir l'échelle et changer les textures des murs et du sol pour un aspect qui correspond mieux au design du jeu et est adapté à des déplacements en VR.

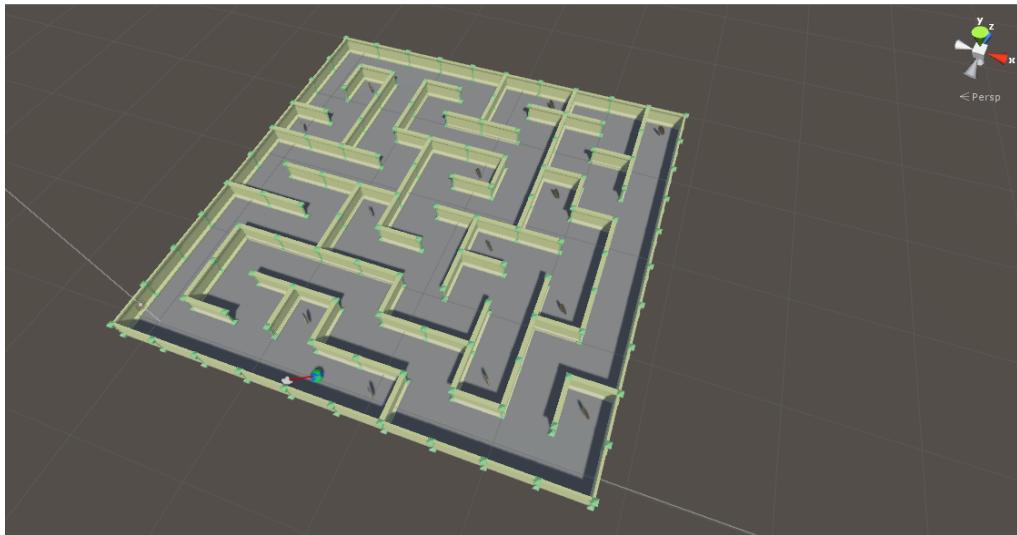


Figure 6: labyrinthe généré aléatoirement

2.2 Version VR

2.2.1 Interaction Raycast

Nous avons basé sur plusieurs tutoriels qu'on a trouvés pour mettre en lieu cette fonctionnalité en VR

<https://www.youtube.com/watch?v=rgTshoVCZVQ>
<https://www.youtube.com/watch?v=CPyaWkj6Ss>

Nous avons mis un raycast en VR en utilisant le component LineRendererer qu'il faut configurer correctement (voir 1er lien). Le principe est de bouger le raycast dans la scène avec une sphère (dont on désactive le collider) à son extrémité pour qu'il collide avec les objets dans la scène (la sphère appelée sur la hiérarchie pointer). Le

raycast change sa longueur en continu en fonction de la position de collider avec quoi la sphère collisionne.

En cliquant sur le trackpad (voir 2ème lien), on détecte pour le moment qu'il y'a un clic avec un Debug ("clicked"), il fallait rajouter d'autres instructions dans ce bloc, à voire interagir avec les objets dont le script Interactable est associé (ou juste dont le layer est interactable comme on a fait dans les scripts Desktop) pour qu'on puisse les choisir ou les bouger, autrement on voulait traduire toutes les fonctionnalités qu'on était arrivé à faire avec la souris en VR.

Sur le script interactionVR/physicsPointer, il y'a toutes les méthodes dont on avait besoin pour mettre en place le raycast.

3 Bilan

Répartition des tâches

Esteban	raycast, énigme table
Salwa	énigme statue (torches ou poèmes), raytracing VR
Thomas	énigme du diamant
Valentine	niveau 2
Victor	gitMaster, transitions, personnage, particules, diamant

Le niveau 1 est presque terminé, il reste éventuellement à ajouter un minotaure pour l'ambiance et les miroirs pour allumer les torches. Il y avait beaucoup plus de travail que ce que le rapport du groupe de l'année dernière laissait à penser. En effet, aucune interaction ne fonctionnait et implémenter intégralement le Raycast et la résolution de toutes les énigmes a été notre tâche principale sur ce projet. L'état du git laissait aussi à désirer. Les améliorations suggérées par nos prédecesseurs étaient :

1. Implémenter l'allumage des torches à l'aide d'un rayon : fait en raycast directement, sans l'idée du miroir.
2. Implémenter l'éénigme du poème (ouverture de la porte et accès au niveau suivant) : fait, il faut maintenant faire le niveau suivant.
3. Implémenter l'interaction énigme de la table -*i*, chute du mur de la seconde pièce : fait
4. Terminer l'écran de fin de jeu (avec le temps passé dans le labyrinthe) : à faire, nous ne savons pas où est l'écran de fin du jeu. Peut-être perdu dans une branche sombre du git.

5. Spatialiser le minotaure (particules et cris) lorsque le joueur se déplace : à faire si on veut garder un minotaure dans le jeu, bien qu'il n'y ait pas d'interactions avec lui.

Le principal problème a évidemment été le manque de temps, notamment puisque nous n'avions plus accès au matériel à la fin du projet, quand nous finissions d'implémenter les interactions en VR. Le projet a aussi mis longtemps à démarrer, nous nous étions concentré sur les TP de COARV au début. De plus, le P2RV et les TP nous ont pris plus de temps que prévu, au détriment de ce projet d'Escape Game VR. Nous nous sommes répartis les tâches au cours d'une réunion au début, mais n'avons finalement pratiquement jamais travaillé en groupe, chacun a développé sa tâche seul, mais nous communiquions bien pour éviter les problèmes lié à git.

Nous avons probablement passé 20h chacun en moyenne sur ce projet, comme le groupe précédent. Etant donné que nous avons travaillé seul la plupart du temps, il est difficile d'évaluer le volume horaire. Cependant, il est important de noter qu'ils étaient 7 et nous ne sommes que 5. Nous ne pouvions donc pas faire autant de choses qu'eux sur ce projet, en plus du manque de temps.

La gestion des problèmes de Git a également été très chronophage, et source de légères tensions parfois au sein du groupe. Ce projet nous a cependant permis de maîtriser un peu mieux cet outil indispensable au développement de projets informatiques. Le hackathon à Laval nous avait déjà bien aidé pour une première prise en main réelle de l'outil.

Nous avons donc principalement déblayé le travail du groupe précédent, qui avait été un peu trop ambitieux peut être, comme ils le disent dans leur bilan. Nous laissons normalement un niveau 1 jouable, donc l'objectif maintenant est de créer le niveau 2 après la porte qui s'ouvre suite à l'énigme du poème. Nous laissons donc un projet plus libre au groupe suivant, car nous avons regretté de ne pas avoir cette liberté de création au cours de notre projet, qui a consisté principalement à finir le travail du groupe précédent.