Université de technologie de Compiègne

RV01 – Réalité Virtuelle

Get Out ! Un escape-game virtuel.

Rapport de projet

Auteurs :
Quentin Duchemin
Aurélie Digeon

Encadrante :
Indira Thouvenin

4 janvier 2018

SOMMAIRE

I CONTEXTE

Ce projet est réalisé dans le cadre de l'Unité de Valeur (UV) RV01 (Réalité Virtuelle), durant le semestre d'Automne 2017, à l'Université de Technologie de Compiègne (UTC).

1 RÉSUMÉ

Le projet que nous avons réalisé dans le cadre de l'UV RV01 est un *escape game*, c'est-à-dire un type de jeu dont le principe consiste pour le joueur à parvenir à s'échapper d'une pièce dans laquelle il est enfermé. Il doit pour y parvenir trouver des éléments cachés dans le décor, résoudre des problèmes et suivre une séquence ou un enchaînement de mouvements précis.

Ce type de jeu s'est démocratisé depuis quelques années, avec des salles ouvrant dans de nombreuses villes de France.

Notre projet veut se distinguer des jeux d'évasion classiques par l'augmentation des possibles via l'environnement virtuel. En particulier, nous avons initialement prévu de mettre en place un système d'interactions progressif, différentes échelles physiques, une ambiance sonore réactive et différents mécanismes de déplacement et d'utilisation des objets.

2 MATÉRIEL

2.1 Environnement de développement

Nous avons utilisé Unity comme moteur de jeu, pour le développement du projet. Pour remplir notre monde virtuel, nous n'avons pas modélisé d'objets par nous même, mais avons utilisé ceux fournis gratuitement dans les asset stores.

2.2 Expérience VR

Nous avons également utilisé **le casque de réalité virtuelle HTC Vive** avec ses deux contrôleurs, qui nous ont permis de déterminer les mouvements des mains du joueur.

3 OBJECTIFS INITIAUX

Nous rappelons ci-après les différents niveaux que nous nous étions fixés pour le projet, et en discutons dans la suite du présent rapport.

3.1 Niveau 1

- Jeu fonctionnel et basique, avec uniquement quelques étapes – énigmes ;
- Niveau visuel minimum (textures – formes) ;
- Pouvoir sélectionner des objets ;
- Pouvoir se déplacer ;
- Gestion basique des sons.

3.2 Niveau 2

- Amélioration des interactions pour rendre les actions plus naturelles pour le joueur ;
- Cohérence esthétique et amélioration de l'environnement ;
- Mise en place de l'inventaire et des multi-actions sur les objets (menu) ;
- Gestion de la spatialisation du son ;
- Utilisation de la télévision avec des images fixes.

3.3 Niveau 3

- Implémentation de(s) changement(s) d'échelle ;
- Interactions fondées sur la spatialisation du son ;
- Tutoriel en temps réel pour les nouvelles interactions voire pour l'histoire ;
- Rendu auto-suffisant pour que le joueur découvre et joue.

II | CHOIX IMMERSIFS

L'un des objectifs de ce projet est de travailler sur l'immersion et les interactions. Dans cette section, nous présentons nos choix pour améliorer l'immersion du joueur.

1 JOUEUR ET AVATAR

Nous avons eu durant tout le projet une volonté forte de garder l'essence de *l'escape game* dans notre jeu. Dans les jeux d'évasion classiques, il est important de pouvoir se déplacer dans la pièce, ainsi que de pouvoir observer, manipuler et utiliser facilement les objets de la salle – ceci pour déterminer lesquels sont utiles aux énigmes et de quelle manière les utiliser. Ces choix ont pour pour but d'améliorer l'**immersion**.

C'est pourquoi, lorsque nous avons dû réfléchir au statut du joueur dans le jeu et aux choix immersifs associés, nous avons tenté de conserver le plus possible la fluidité et les gestes intuitifs d'un joueur dans une véritable salle d'*escape game*. Pour ce faire, nous avons catégorisé les interactions classiques d'un joueur avec son environnement :

- **Observer** : obtentions d'informations supplémentaires, comme la spécialité de l'objet. Dans le cas de la VR, on peut par exemple matérialiser des informations impossibles à obtenir *in situ*, comme une odeur ou un goût.
- **Utiliser** : déclenchement d'une action associée à l'objet, *e.g.* ouverture d'une porte, mise en route d'une radio, etc.
- **Prendre** : déplacement contrôlé d'un objet. On peut vouloir le rapprocher pour mieux l'observer, ou bien l'utiliser dans le cadre d'une énigme.

Nous avons également souhaité que les mouvements du joueur dans le monde réel soient le plus fidèlement reproduits au sein du monde virtuel afin de fluidifier le jeu et d'améliorer l'immersion. Ainsi, le joueur devra s'agenouiller, se pencher, et marcher pour pouvoir résoudre toutes les énigmes qui s'offriront à lui. Ces déplacements sont *isomorphiques*, *i.e.* correspondants aux mêmes échelles dans l'environnement réel comme l'environnement virtuel.

Pour résumer, nous avons travaillé dans la volonté de minimiser la transition entre l'environnement réel et l'environnement virtuel, en gardant son champ d'action aussi large que possible et en travaillant avant tout sur l'immersion.

2 ENVIRONNEMENT VIRTUEL

Toujours dans un souci d'immersion du joueur dans le jeu, nous avons porté un soin particulier au réalisme de notre environnement virtuel. En effet, nous pensons qu'il est primordial d'avoir un décor crédible pour pouvoir plonger le joueur dans l'histoire et l'ambiance du jeu.

2.1 Objets et textures

Nous avons tout d'abord choisi soigneusement les objets présents dans la salle. Nous avons voulu donner une cohérence d'ensemble, en choisissant des objets d'une même époque. Un grand soin a également été porté en ce qui concerne les textures et les matériaux afin de proposer une ambiance homogène au joueur.

2.2 Lumières

Un autre point à noter est l'ambiance lumineuse de l'environnement virtuel. On peut citer les lumières vacillantes, les lampes s'allumant individuellement et réparties astucieusement dans la pièce pour approcher un éclairage naturel. Dans certains cas, allumer une lampe permet de mieux observer des éléments du décor, et aide donc à résoudre des énigmes. La lumière fournie par la cheminée contribue à l'atmosphère de la pièce.

2.3 Sons

Depuis le début du projet, il nous tenait à cœur de donner à notre jeu une ambiance sonore, composante essentielle de l'immersion. Même si – comme nous le développerons dans la dernière partie du rapport – nous n'avons pas pu aller aussi loin que nous le souhaitions, le son est tout de même un point important du jeu. En effet, nous avons pris soin de faire en sorte que les actions du joueur sur l'environnement entraînent un **retour sonore** à chaque fois que cela est nécessaire. On peut par exemple citer les grincements de porte, le clic des interrupteurs, le crépitement du feu, etc.

Ces sons ont pour but de *renforcer* l'immersion du joueur. Pour illustrer cela, prenons le dénouement de l'énigme du piano. Quand le joueur réussit à jouer la mélodie sur le piano, il entend un grincement, qui lui indique que le placard s'est ouvert. On peut également citer le bruit que fait la télévision du grenier lorsque le joueur réussit une étape du tutoriel.

A fortiori, le piano, qui est un élément central du jeu, ainsi que la mélodie de l'énigme [1] jouent un rôle essentiel dans l'atmosphère de notre jeu.

1. Extraite de la Bande Originale du film *La liste de Schindler*.

2.4 Animations

Pour finir, nous avons travaillé sur les animations du jeu. En effet, pour permettre une bonne immersion, il nous semblait primordial que les animations des objets soient les plus naturelles possible. On peut par exemple citer le mouvement des interrupteurs des lumières. En effet, au lieu de nous contenter d'allumer la lumière, sans changer l'apparence du bouton, nous avons pris soin de faire une animation sur l'interrupteur. On peut aussi penser à l'animation de la chute de la feuille de solfège, dans la bibliothèque, qui est fluide et naturelle.

3 MÉTAPHORES

Pour améliorer l'expérience utilisateur et nous adapter aux contraintes de la Réalité Virtuelle, nous avons utilisé des métaphores visuelles et de déplacement. Celles-ci permettent au joueur de mieux interagir avec son environnement et de faciliter son immersion dans le jeu, en minimisant la frustration qu'entraîneraient des difficultés à interagir ou à se déplacer.

3.1 Raycast

Pour permettre au joueur de cibler précisément les objets avec lesquels il veut interagir, notamment pour l'observation, nous avons mis en place un système de *raycast*. Cette technique consiste à **matérialiser** l'environnement ciblé, ici par un rayon bleu sortant de la baguette tenue par le joueur. Cette technique est assez précise et permet au joueur de pouvoir pointer – et donc **identifier** – des objets éloignés de lui.

3.2 Halo

Nous ne pouvions pas rendre tous les objets de l'environnement virtuel manipulables ou observables. Nous avions donc besoin d'un moyen pour distinguer les objets utilisables des autres.

Pour résoudre ce problème, nous avons utilisé un halo de lumière qui entoure les objets utilisables quand le joueur les pointe avec le *raycast*. Ainsi, cela nous permet d'indiquer que cet objet peut être utilisé, sans contraindre l'utilisateur à tester tous les objets dans l'environnement, minimisant ainsi la frustration.

3.3 Déplacement au joystick

Nous avons aussi offert au joueur la possibilité de se déplacer grâce au *joystick* du contrôleur gauche, pour prévoir le cas où l'environnement réel serait trop petit pour permettre au joueur de traverser la totalité de l'environnement virtuel.

III | IMPLÉMENTATION TECHNIQUE

Cette section détaille les moyens techniques utilisés pour implémenter certaines fonctionnalités importantes du jeu. Nous nous concentrons ici sur le développement d'un mécanisme d'**interactions** et d'**événements** visant à accélérer le prototypage et les évolutions.

1 CONCEPTS

L'étude du fonctionnement du jeu nous a amené à différencier deux concepts *effectivement* similaires mais **sémantiquement** très différents : les **interactions** et les **événements**.

- Les interactions représentent une action **directe** du joueur sur son environnement, *e.g.* l'utilisation d'un objet.
- Les événements représentent une **réaction** à une interaction ou à un autre événement, *e.g.* l'ouverture d'une porte après la complétion d'une énigme.

Dans notre environnement virtuel, plus d'une vingtaine d'objets proposent des interactions : les livres, les meubles, les interrupteurs, le piano, les objets de décoration... Les événements sont également mis à contribution. Pour ne citer que quelques exemples :

- L'appui sur un interrupteur doit déclencher un certain nombre d'actions, comme l'ouverture d'une trappe ou le déclenchement de lumières.
- La complétion d'une mélodie au piano doit ouvrir un placard.
- La pose de runes sur les supports de la table doit conditionnellement éteindre la cheminée.

Nous pouvons dès lors identifier trois problématiques fortes se posant lors de l'implémentation technique.

D'une part, les éléments du jeu sont **fortement couplés**. Naïvement, le piano devrait connaître l'existence du placard pour agir sur lui, par exemple. Les supports de runes devraient connaître leur fonction **externe**, à savoir allumer ou éteindre une cheminée. C'est un sérieux problème pour la maintenance et la ré-utilisation, une modification d'un côté impliquant une modification de l'autre.

D'autre part, beaucoup de concepts concernant les interactions ou mes événements sont réutilisables : seule la sémantique change d'une interaction à l'autre. Ceci implique, par exemple, que l'implémentation naïve des interactions provoquerait une duplication massive de code et affecterait considérablement la maintenabilité.

Enfin, le développement d'un mécanisme pour un objet et un projet donnés peut prendre du temps ; il est à ce titre souhaitable de permettre la réutilisation des objets et de leurs scripts dans d'autres projets. La dépendance à l'environnement empêche la réutilisation.

Nous avons donc tenté de trouver une solution à ces problématiques que nous présentons ci-après.

2 RÉALISATION D'UN MICRO FRAMEWORK

2.1 Vue d'ensemble

Nous avons créé un **micro framework** prenant en charge les interactions et les événements, en fournissant un socle de développement, des gestionnaires et des fonctions utilitaires. Les objectifs et concepts clés du framework sont les suivants :

1. **Découplage complet** des objets du jeu ;
2. Réduction du code client à la **sémantique interne** des objets ;
3. Ouverture des interactions à plusieurs objets source, voire **plusieurs joueurs** génériques ;
4. **Rapidité** d'implémentation et **maintenabilité** ;
5. **Réutilisabilité** des objets et mécanismes du jeu.

Pour ce faire, l'idée centrale est que chaque objet ne se connaît que lui-même : il sait ce qu'il sait faire **pour lui**. Il est capable d'écouter des événements, mais ceux-ci doivent être indépendants de son environnement. À titre d'exemple, une porte sait qu'elle peut s'ouvrir et peut attendre un événement de type `OpenDoor`. Si elle s'en tient là, elle pourra être réutilisée dans un autre projet, sans modifications.

Notons enfin que le framework est **agnostique** en ce qui concerne la *manière* d'interagir avec les objets : c'est la responsabilité du code client. Par exemple, notre jeu est jouable sur PC ou en VR. Bien que les façons de déclencher les interactions soient différentes, le framework est le même.

Les éléments essentiels du framework se trouvent dans les dossiers `Interactions` et `Events` des scripts. Notons à ce titre que ce rapport n'est **pas** une documentation technique, et que

les commentaires et exemples dans le jeu sont exhaustifs quant aux utilisations possibles que nous ne détaillons pas ici.

2.2 Interactions

Les objets proposant des interactions sont invités à hériter de la classe abstraite `InteractionBase`. Cette classe propose un socle commun pour les interactions ainsi que des facilités d'implémentation.

Les objets décrivent simplement les interactions qu'ils proposent en les associant à des fonctions. Les interactions sont de trois types : `Observe`, `Use` et `Take`.

Les interactions proposées peuvent être amenées à évoluer en fonction de l'**état interne** des objets, *e.g.* une porte qui ne peut s'ouvrir qu'une seule fois. La table d'association `availableInteractions` adresse ce problème : elle peut être modifiée au cours du temps pour rendre disponibles de nouvelles interactions.

Enfin, reste la problématique principale : l'**état externe** du jeu influence les interactions disponibles à un instant donné, *e.g.* une porte qui ne peut s'ouvrir que si le joueur a la clé correspondante. La classe `InteractionManager` centralise cette logique, en proposant des méthodes génériques – **couplées** au jeu cette fois – qui indiquent si un objet peut interagir avec un autre objet. La méthode `CanInteract(GameObject source, GameObjet target, InteractionType type)`, par exemple, indique si le contexte courant du jeu permet à une source quelconque – joueur ou non – peut initier un certain type d'interaction avec une cible.

Ainsi, à chaque fois qu'une entité (*e.g.* un joueur) tente d'interagir avec un autre objet, sa méthode `GetInteractions` ne renvoie que les interactions disponibles selon l'état **interne** de l'objet qui ont été validées par `InteractionManager` selon l'état **externe** de l'environnement.

Enfin, la classe `InteractionDefaults` propose des interactions par défaut pour éviter la duplication de code :

- L'observation par défaut affiche un message paramétrable à l'écran.
- Le ramassage par défaut fixe l'objet visé à l'objet source via un `FixedJoint`.
- Une méthode générique `EndInteractions` permet de terminer les interactions en cours entre deux objets.

Nous avons décrit succinctement les principales idées, le code étant amplement documenté. Donnons un exemple pour illustrer la rapidité de mise en œuvre. Le code 1 montre le script minimal à utiliser pour rendre un téléphone utilisable et observable. Au début, on indique que

le téléphone est capable de répondre aux interactions de type `Observe` et `Use`, en leur associant des méthodes. Le reste est géré automatiquement, la méthode `GetInteractions` permettant de ne fournir que les interactions disponibles pour un contexte donné étant implémentée dans `InteractionBase`.

```
public class PhoneController : InteractionBase {
  private AudioSource ringtone;

  void Start () {
     ringtone = GetComponent<AudioSource> ();
     availableInteractions.Add (InteractionType.Observe, new UnityAction (ObservePhone));
     availableInteractions.Add (InteractionType.Use, new UnityAction (RingPhone));
  }

  public void ObservePhone() {
     defaultInteractions.Observe ("Ce vieux telephone n'a pas l'air de fonctionner...");
  }

  public void RingPhone() {
     ringtone.Play ();
  }
}
```

Listing 1 – Implémentation d'un téléphone utilisable et observable.

Donnons tout de même, pour finir, un exemple minimal d'interaction avec l'objet ci-dessus. Le code 2 montre l'utilisation du téléphone, en supposant que `objScript` est une référence au composant `InteractionBase` du téléphone.

Dans ce code, l'objet ne sera marqué comme utilisable que si le gestionnaire central a déterminé que le joueur (objet `player`) a l'autorisation d'effectuer une action de type `Use` sur `objScript`.

```
void UseObject (GameObject player, InteractionBase objScript) {
   var interactions = objScript.GetInteractions (player);
   UnityAction use;
   if(interactions.TryGetValue(InteractionType.Use, out use))
     use.Invoke ();
}
```

Listing 2 – Utilisation du téléphone.

2.3 Événements

En ce qui concerne les événements, le système peut être résumé assez simplement en trois phases :

1. Certains objets écoutent des événements ;
2. Tous les objets signalent leurs actions notables ;
3. Des liens de causalités entre actions et événements déclenchent les nouvelles actions voulues en préservant le découplage.

La classe `EventManager` gère les écoutes et les déclenchements, tandis que la classe `EventCausality` gère les causalités.

Un exemple direct sera cette fois-ci plus parlant. La porte du placard doit s'ouvrir lorsque la mélodie jouée par le joueur est correcte. Le code 3 montre la partie « événements » du piano : il se contente de signaler la fin de la mélodie.

```
if(CorrectMelody()) {
  EventManager.Done("MelodyPlayed");
}
```

Listing 3 – Signalement de fin d'action du piano.

Le code 4 montre l'écoute du *trigger* d'ouverture de la porte par le placard.

```
void Start() {
  EventManager.StartListening("TriggerSmallDoor", new UnityAction(TriggerSmallTable));
}

void TriggerSmallTable() {
  animator.SetBool(...)
  audio.Play(...)
}
```

Listing 4 – Attente d'un trigger par le placard.

À ce stade, il ne reste plus qu'à lier la fin d'action du piano (« *pre-trigger* ») au déclenchement de l'ouverture de la porte (« *trigger* »). Le code 5 montre comment faire, et il est à implémenter dans `EventCausality`.

```
void Awake () {
  EventManager.AddCausality("MelodyPlayed", "TriggerSmallDoor");
}
```

Listing 5 – Lien de causalité entre mélodie jouée et ouverture de porte.

Ces trois extraits de codes suffisent à lier les actions tout en les découplant. Bien entendu, ce système n'est pas limité à deux actions : elles peuvent être chaînées. L'examen du code du micro framework permettra de s'en convaincre.

IV | RÉALISATION

Cette section détaille les éléments que nous avons implémenté au regard de nos prévisions et de nos objectifs, fait état des difficultés rencontrées au cours du projet et conclut le rapport.

1 MISE À JOUR DES OBJECTIFS

Au cours du développement de notre jeu, nous nous sommes rendu compte que certains de nos choix initiaux d'interfaces étaient peu intuitifs et immersifs et avons donc décidé de les abandonner au profit d'autres métaphores et techniques.

Tout d'abord, nous avions prévu de développer un système d'inventaire, qui aurait permis à notre joueur de stocker des objets. Nous nous étions inspiré des jeux classiques sur PC, comme les *point and click*, proposant généralement ce fonctionnement. Mais après avoir développée une première version de notre jeu, nous nous sommes rendu compte que ce système ne correspondait pas du tout à la Réalité Virtuelle : peu ergonomique, peu intuitif et brisant l'immersion. Nous avons donc décidé d'abandonner la notion d'inventaire, et avons à la place mise en place une interaction avec les objets initialement non-prévue : le fait de pouvoir les prendre et les déplacer. Cette interaction est bien plus intuitive que l'inventaire et participe de plus à l'immersion du joueur.

Nous avons également réévalué la nécessité de l'utilisation des menus contextuels sur les objets. Notre idée de départ était de permettre au joueur, une fois un objet sélectionné, de choisir plusieurs actions possibles dans un menu. De même que pour l'inventaire, nous nous sommes vite rendu compte que c'était peu intuitif, dynamique et immersif pour le joueur. De plus, nous avons vite compris que nos objets allaient avoir un très faible nombre d'interactions différentes possibles (observer, utiliser et prendre) et qu'il suffisait donc d'associer un bouton ou une action à chacune d'entre elles.

Avec ces deux modifications, nous avons réduit le nombre d'éléments *UI (User Interface)*, qui certes ajoutent des interactions et des possibilités dans certains jeux, mais casse l'immersion du joueur. Le seul élément *UI* que nous avons gardé dans notre projet est la bulle apparaissant pour afficher la description des objets lors de leur observation.

Enfin, arrivé à la fin du projet, nous avons dû faire des choix sur les parties à développer en priorité. Nous avons choisi de privilégier la mise en place d'un tutoriel intuitif au détriment

du développement du son spatialisé ainsi que du changement d'échelle. En effet, même si ces deux points auraient été très intéressants à travailler, le jeu peut très bien s'en passer. Au contraire, il était primordial de mettre en place un tutoriel travaillé permettant de présenter le fonctionnement du jeu à l'utilisateur tout en conservant l'immersion. Les quelques personnes ayant testé notre jeu ont ainsi pu apprendre très rapidement les interactions proposées et commencer à jouer presque immédiatement.

2 DIFFICULTÉS RENCONTRÉES

Même si nous sommes aujourd'hui satisfaits du rendu final de notre projet, nous aurions aimé pouvoir aller plus loin, mais avons rencontré des difficultés qui nous ont freinées.

Tout d'abord, le passage d'un jeu classique en rendu 2D à un jeu en VR a été beaucoup plus long que nous ne l'avions prévu. Nous avons eu beaucoup de mal à trouver de la documentation de qualité et des tutoriels correspondants à nos besoins. Nous avons passé de nombreuses heures à comprendre et appliquer les principes permettant le passage à la VR, ce qui nous a empêché de développer d'autres fonctionnalités initialement prévues.

En particulier, nous avons rencontré des difficultés en ce qui concerne la gestion des *colliders* face aux différents moyens de déplacement, ainsi qu'avec la gestion de la prise d'objets, notamment au niveau de la conservation de la gravité.

Il faut également noter que le découplage des objets et des événements que nous avons voulu mettre en place dans notre projet n'a pas toujours été trivial, ce qui nous a pris également un certain temps.

Enfin, le développement de notre framework de gestion des interactions et des événements est un atout de notre projet, mais nous a demandé un temps important qui n'avait pas été prévu dans notre planning prévisionnel. Cela a participé au fait qu'il nous a manqué du temps pour le développement d'autres fonctionnalités.

3 CONCLUSION

Ce projet a été l'occasion pour nous de nous familiariser avec différents concepts. En particulier, nous avons beaucoup appris sur la technique avec l'utilisation d'Unity et des outils de Réalité Virtuelle, mais aussi sur la cognition, en travaillant sur les concepts d'immersion, d'interface utilisateur, de tutoriels et de retours.

Nous aurions aimé ajouter d'autres énigmes, développer l'histoire et enrichir l'environnement, mais Get Out ! est, à ce stade, un projet qui s'auto-suffit : il est jouable sur PC et en VR, contient un tutoriel complet, plusieurs énigmes et une fin.

Enfin, le développement d'un micro framework permettant de découpler les objets du jeu et d'accélérer le développement nous a amené à réfléchir à l'architecture technique de notre code, et à l'optimiser pour le rendre maintenable. Peut-être que celui-ci continuera à évoluer, si nous en avons le temps…