# Compte rendu – Projet de Programmation Multimédia

**BUT 3 – Parcours A**
IUT Aix-en-Provence – Site Gaston Berger
Année universitaire 2025–2026

---

## 1. Présentation générale du projet

Ce projet a été réalisé dans le cadre du module *Programmation Multimédia* du BUT 3 Parcours A. L'objectif était de concevoir un **mini-jeu 3D sous Unity**, à partir d'un **starter pack fourni**, en combinant programmation (C#) et infographie (modélisation 3D, textures, animations).

Le projet a été mené **individuellement**. Il s'appuie sur les bases abordées lors des travaux dirigés, tout en proposant plusieurs adaptations personnelles, notamment sur le gameplay, les interactions et l'univers du jeu.

### Concept du jeu

Le joueur incarne un chanteur devant se frayer un chemin à travers la foule et les agents de sécurité pour atteindre la scène. Le gameplay repose principalement sur **l'évitement**, et non sur le combat. Le joueur doit récupérer des **gourdes d'eau** pour s'hydrater, ainsi qu'un **objet clé (un micro)** permettant d'activer un mécanisme essentiel : le téléporteur.

Il doit également éviter les **zones mortelles**, représentées par la foule dense, ainsi que les **ennemis**, incarnés par les agents de sécurité. Une fois toutes les conditions réunies, le joueur peut accéder à la zone finale et remporter la partie.

---

## 2. Outils et technologies utilisés

Les outils et technologies suivants ont été utilisés pour la réalisation du projet :

- **Unity** : moteur de jeu 3D
- **C#** : scripts de gameplay
- **Blender** : modélisation 3D
- **GIMP** : création et retouche de textures

- **Starter Pack BUT** : scripts, animations, sons et assets de base
- **Git** : gestion de versions et rendu du projet

Le starter pack fournissait notamment :

- un système de déplacement du joueur,
- un contrôleur d'animations,
- des effets sonores (pas, ramassage),
- des scripts de caméra orbitale,
- des exemples de téléporteurs et d'objets interactifs.

---

# 3. Organisation du projet Unity

Le projet Unity est structuré de manière claire et modulaire afin de faciliter la maintenance et la compréhension du code :

- **Animations/** : animation du casier
- **Audio/** : audio pour la récupération des gourdes, du micro et du butin
- **Materials/** : matériaux Unity
- **Models/** : modèles du butin, du casier, de la gourde et du micro
- **Models/Materials/** : matériaux Unity spécifiques aux modèles
- **Models/Textures/** : textures spécifiques aux modèles
- **Prefabs/** : joueur, foule, ennemis, casier, micro, butin, téléporteur, gourde
- **Scenes/** : scène principale
- **Scripts/Enemies/** : Enemy
- **Scripts/Interactions/** : Locker, MoneyPicker, OnTriggerKill, RotateObject, Teleporter, WaterPickup
- **Scripts/Managers/** : GameManager, ScoreManager
- **Scripts/Player/** : FollowObject, Killable, NewControls, PlayerMovement
- **UI/Icons/** : icône du micro

Un travail de nettoyage a été effectué afin de supprimer les doublons issus du starter pack et de conserver uniquement les scripts réellement utilisés dans le projet.

---

# 4. Gameplay et déplacement du joueur

## 4.1 Système de déplacement

Le déplacement du joueur repose sur un script **PlayerMovement**, adapté à partir de celui fourni dans le starter pack.

Il utilise :

- un **CharacterController** pour la gestion des collisions,
- le **New Input System** pour la prise en charge du clavier et de la manette,
- une **caméra orbitale**, dont l'orientation influence directement la direction du déplacement.

Les fonctionnalités mises en place sont les suivantes :

- déplacement fluide en fonction de l'orientation de la caméra,
- gestion de la gravité,
- rotation progressive du personnage vers la direction du mouvement.

Pour renforcer le réalisme du déplacement, un système de sons de pas a été ajouté. Ceux-ci sont déclenchés dynamiquement en fonction de l'état du joueur (déplacement et contact avec le sol). Une vérification garantit la présence d'une source audio afin d'éviter toute erreur lors de l'exécution du jeu.

Le déplacement clavier utilise une configuration **WASD**, conformément aux paramètres par défaut du projet Unity. Une tentative de configuration en **ZQSD** a été envisagée, mais n'a pas pu être finalisée dans le temps imparti.

### 4.2 Paramétrage via ScriptableObject

Les paramètres de déplacement sont externalisés dans un **ScriptableObject Movement**, ce qui permet d'ajuster le comportement du joueur sans modifier directement le code. Les paramètres configurables incluent notamment :

- la vitesse maximale,
- le facteur de sprint,
- la gravité,
- la puissance du saut.

---

## 5. Défis techniques rencontrés

### 5.1 Problème de lenteur du joueur

Lors des premières phases de test, le joueur se déplaçait très lentement malgré des paramètres de vitesse élevés. Après analyse, plusieurs causes ont été identifiées :

- une valeur d'input clavier plus faible comparée à celle d'une manette,
- une courbe d'accélération trop restrictive,
- un calcul du déplacement peu adapté au contexte temporel.

La solution a consisté à ajuster les seuils d'input, à utiliser correctement `Time.deltaTime` et à adapter les multiplicateurs de vitesse.

### 5.2 Gestion des erreurs et conflits de scripts

Des erreurs de type `NullReferenceException` sont apparues lors de l'accès à la caméra en dehors du mode *Play*. La solution a été de conditionner certaines exécutions à l'état `Application.isPlaying`.

Par ailleurs, la présence de plusieurs scripts similaires issus du starter pack a généré des conflits. Un tri et une adaptation du script principal ont permis de stabiliser le projet. L'utilisation de l'IA a été un appui important pour l'identification et la résolution de ces erreurs.

---

## 6. Gestion de la mort du joueur

Le joueur peut mourir au contact des ennemis (agents de sécurité) ou des zones mortelles (la foule). Cette mécanique repose sur un composant **Killable** attaché au joueur.

Fonctionnement :

- les ennemis détectent le joueur via des **colliders en mode trigger**,
- l'appel à la méthode `Kill()` désactive le déplacement et les collisions,
- le **GameManager** gère la fin de la partie et le redémarrage de la scène.

Ce système est volontairement modulaire : tout objet disposant du composant *Killable* peut provoquer la mort du joueur sans nécessiter de modification des scripts ennemis.

---

## 7. Mécanique clé : casier, micro et téléporteur

La mécanique de clé demandée dans les consignes a été adaptée :

- la clé est remplacée par un **micro**,
- le coffre est remplacé par un **casier animé**.

Le casier est animé via un **Animator** utilisant un trigger d'ouverture. Lorsque le micro est récupéré :

- il disparaît de la scène,
- son icône s'affiche dans le HUD,
- le téléporteur devient utilisable.

La détection d'interaction est gérée à l'aide de `Physics.OverlapBox`, une solution plus fiable que les triggers classiques avec un *CharacterController*.

---

# 8. Objets ramassables et condition de victoire

## 8.1 Butin final

Le butin final remplace le trophée demandé dans les consignes. Il déclenche la victoire et l'affichage du panneau de fin de jeu.

Le modèle initial simplifié a été remplacé par un **sac de pièces 3D**, tout en conservant :

- le collider,
- le script de ramassage,
- l'AudioSource.

## 8.2 Objets secondaires

Les points d'eau sont représentés par des **gourdes 3D**. Afin d'améliorer leur lisibilité visuelle, une animation de rotation continue a été ajoutée via un script dédié utilisant `Transform.Rotate()` et `Time.deltaTime`.

---

# 9. Effets sonores et particules

Des effets sonores ont été ajoutés afin de renforcer le retour utilisateur lors des interactions :

- ramassage des gourdes,
- récupération du micro,
- obtention du butin final.

Chaque objet dispose de son propre **AudioSource**, déclenché au moment de l'interaction. Un **système de particules** a également été intégré au téléporteur afin de rendre la téléportation plus lisible visuellement. Les particules ne sont déclenchées que si le micro a été récupéré.

---

# 10. Interface utilisateur (UI)

L'interface utilisateur repose sur un **Canvas Unity** contenant plusieurs panels :

- un **panel de contexte**, affiché au lancement du jeu, avec un bouton *Commencer* mettant le jeu en pause,
- un **panel de victoire**, affiché à la fin de la partie, avec des boutons *Rejouer* et *Quitter*.

La gestion de l'UI est centralisée dans le **GameManager**, ce qui garantit une séparation claire entre la logique de gameplay et l'interface. Les boutons utilisent le système d'événements `OnClick()` de Unity pour appeler les fonctions correspondantes.

---

# 11. État d'avancement du projet

Fonctionnalités opérationnelles :

- déplacement complet du joueur,
- interactions avec les objets,
- casier animé,
- téléporteur conditionnel,
- ennemis présents,
- gestion de la mort du joueur,
- interface de début et de victoire,
- effets sonores et particules.

---

# 12. Méthodologie et apprentissages

Ce projet a permis de consolider plusieurs compétences :

- gestion des interactions dans Unity,
- utilisation du *CharacterController*,
- communication entre scripts,
- intégration d'assets externes,
- workflow Blender → Unity,
- importance d'une architecture de projet claire et modulaire.

---

# 13. Conclusion

Ce projet constitue un **prototype fonctionnel et cohérent**, respectant les objectifs pédagogiques du module. Malgré un temps limité en fin de projet, les mécaniques principales sont opérationnelles et pourraient être étendues ou améliorées ultérieurement.

---

## Sources et références

- Casier : Sketchfab – Locker
- Butin : Sketchfab – Coin Bag
- Gourde : Sketchfab – Worn Water Bottle
- Agent de sécurité : Sketchfab – Low poly security guard 2
- Téléporteur : Sketchfab – Teleporter Board (Old Look)
- La foule : Sketchfab – Freebie - Lowpoly People
- Barrière : Sketchfab – Crowd Barrier / Fence
- Jukebox : Sketchfab – Jukebox
- Player : Sketchfab – Stylized Blond Girl Sculpt
- Micro : modèle personnel réalisé pour le projet Blender

L'utilisation de **ChatGPT** a servi d'aide au débogage, à la compréhension des erreurs Unity ainsi qu'à la reformulation et à la structuration du compte rendu.