FIVE NIGHTS AT CHATELET

- PROMO 2030 -

NEEV CHANDIRAMANI - GAUTHIER ROLAND - MAXIME YE - MAXIME GONTRAN

HTTPS://FIVENIGHTSATCHATELET.NEEVCHANDIRAMANI.COM/

RAPPORT DE PROJET

TABLE DES MATIÈRES

I.Introduction

Five Nights At Châtelet est un jeu d'horreur et de survie multijoueur inspiré de la célèbre franchise *Five Nights at Freddy's*. Développé dans le cadre du projet SAE J3D de première année à l'EPITA, il a pour objectif la conception et la réalisation complète d'un jeu vidéo en équipe, mobilisant des compétences en conception fonctionnelle, en organisation de projet et en développement logiciel.

Le jeu propose une expérience en vue à la troisième personne dans une reconstitution oppressante de la station de métro Châtelet Les Halles. Les joueurs y sont répartis automatiquement en deux camps aux objectifs opposés : les Survivants, qui coopèrent pour résoudre une série d'énigmes et s'échapper, et les Infectés, qui traquent les survivants dans les couloirs pour les éliminer. Le contraste entre un style graphique volontairement cartoon et une ambiance sonore et visuelle angoissante constitue l'identité du projet.

Ce document constitue le rapport de projet final, remis à l'occasion de la troisième et dernière soutenance. Il fait le bilan complet du travail accompli depuis la validation du cahier des spécifications techniques, en reprenant le cahier des charges, en détaillant les choix d'architecture et les fonctionnalités implémentées, en décrivant la contribution individuelle de chaque membre, et en relatant le récit de la réalisation, avec ses réussites comme ses difficultés.

Note sur la composition de l'équipe : le studio est passé de cinq à quatre membres en cours d'année à la suite du départ de Maximilien Cochet. Ses tâches ont été redistribuées entre les membres restants — Neev Chandiramani, Gauthier Roland, Maxime Ye et Maxime Gontran — comme détaillé dans la section consacrée à la répartition des tâches.

Depuis la deuxième soutenance, le projet a franchi toutes les étapes restantes : l'intelligence artificielle est en place sous la forme d'un Embuscadeur réagissant au bruit du joueur, les quatre énigmes coopératives sont fonctionnelles, l'attribution automatique des rôles est gérée par le serveur, et le jeu est désormais distribué sous forme d'exécutables compilés pour Windows et Linux, accompagnés d'un installateur et désinstallateur Windows complet.

II.Reprise du cahier des charges

Le sujet imposait la réalisation, en équipe, d'un jeu vidéo programmé en Python et fonctionnel sous Windows 10/11. Au-delà de la liberté laissée sur le thème, trois éléments techniques étaient obligatoires et ont structuré l'ensemble de nos choix dès le début du projet.

Les contraintes obligatoires

- **Le framework graphique PyGame.** PyGame est utilisé pour tout l'habillage 2D du jeu : le menu principal, l'écran d'options, les overlays de screamers et l'ensemble du moteur audio. Le rendu 3D, lui, s'appuie sur Ursina Engine, dont l'usage a été validé par les responsables en complément de PyGame.
- **Une forme d'intelligence artificielle simple.** Cette contrainte est remplie par l'Embuscadeur (section XI), une entité hostile qui se repère seule dans la station, détecte le joueur en fonction du bruit qu'il produit et déclenche une attaque autonome. Elle réagit donc directement aux actions du joueur sans intervention humaine.
- **Un mode multijoueur en réseau.** Le jeu repose sur une architecture client-serveur TCP : un serveur dédié distinct synchronise en continu l'état de la partie entre les différents clients (section XIII). Il s'agit d'un véritable multijoueur en réseau, et non d'un multijoueur local sur un même exécutable.

Notre réponse au cahier des charges

Dès la conception, nous avons pris ces trois obligations comme point de départ plutôt que comme contraintes ajoutées en fin de parcours. Le réseau, en particulier, impose une architecture très différente d'un programme purement local : la séparation stricte entre la logique de jeu et le rendu graphique a guidé l'organisation de tout le code. Le tableau ci-dessous résume comment chaque exigence a été satisfaite.

Exigence du sujet	Réponse apportée dans le projet
Langage Python	Intégralité du code en Python 3.11 (jeu, serveur, énigmes, menu)
Fonctionnel sous Windows 10/11	Exécutable et installateur Windows produits automatiquement par la CI
Framework PyGame	Menu, options, overlays, mixeur audio et HUD 2D sous PyGame
Intelligence artificielle	L'Embuscadeur : détection au bruit, tension, bond autonome
Multijoueur en réseau	Serveur TCP dédié + clients synchronisés en JSON (port 5555)
Originalité	Multijoueur asymétrique, énigmes RATP, station Châtelet, screamers, jeu horreur

Synthèse de la conformité au cahier des charges

Le support multi-plateforme, optionnel selon le sujet, a également été assuré : le jeu fonctionne sous Linux en plus de Windows. Cette portabilité n'a pas été sans poser des difficultés techniques, détaillées dans la section consacrée au récit de la réalisation.

III.Concept et déroulement du jeu

Five Nights at Châtelet repose sur un gameplay multijoueur asymétrique en vue à la troisième personne. Les joueurs sont automatiquement répartis en deux camps aux objectifs opposés : les Survivants, qui doivent coopérer pour résoudre des énigmes et s'échapper de la station, et les Infectés, dont le but est de traquer et d'éliminer les survivants avant qu'ils ne parviennent à fuir.

Ambiance recherchée : un couloir de métro parisien désaffecté

Les deux rôles

Le Survivant incarne un voyageur piégé dans la station. Il ne peut pas attaquer ; sa survie repose sur la discrétion, la coopération et la résolution des énigmes. Il dispose de 100 points de vie et d'une vitesse de déplacement de référence.

L'Infecté est plus rapide (vitesse augmentée d'environ 15 %) et plus résistant (150 points de vie). C'est le seul rôle capable d'attaquer : son objectif est d'emprisonner tous les survivants avant la fin de la partie. Les modèles 3D et les statistiques de chaque rôle sont définis dans une table centrale décrite à la section IX.

Le déroulement d'une partie

Une partie s'articule autour de trois grandes phases :

- **Initialisation :** à la connexion, un écran de lobby permet aux joueurs de se signaler « prêts ». Une fois tout le monde prêt — ou si un joueur force le démarrage — le serveur tire et fige les rôles, puis chaque client affiche une animation d'annonce révélant son camp.
- **Phase active :** les survivants explorent la station à la recherche des quatre énigmes qui leur sont assignées, tandis que les infectés patrouillent. L'atmosphère est volontairement oppressante : pénombre, sons d'ambiance, chuchotements lointains, battements de cœur quand les points de vie sont bas, et screamers déclenchés par certains éléments du décor ou par l'Embuscadeur.
- **Dénouement :** la partie se termine de trois manières — par l'évasion des survivants (lorsque toutes les énigmes sont résolues, un passage s'ouvre vers la sortie), par l'emprisonnement de tous les survivants (victoire des infectés), ou à l'écoulement du temps. Un message de fin s'affiche puis le jeu revient au menu principal.

Cette boucle de jeu est entièrement fonctionnelle en réseau comme en solo. En l'absence de serveur, le jeu démarre automatiquement en mode solo après un court délai, garantissant qu'il reste toujours jouable.

IV.Organisation du groupe et méthodologie

Afin d'assurer une collaboration fluide et une progression constante du jeu, notre studio BambouX s'est appuyé sur une organisation fondée sur la communication permanente, la répartition claire des responsabilités et l'usage d'outils professionnels de gestion de code.

Gestion de la communication

La quasi-totalité des échanges s'est faite sur un canal de discussion dédié sur Discord. Cette centralisation des informations a permis à chaque membre de rester à jour, d'organiser des sessions de travail régulières et de résoudre rapidement les blocages techniques. C'est un outil qui s'est révélé indispensable pour maintenir la cohésion d'équipe sur un projet de cette durée, d'autant plus

après le départ d'un membre.

Gestion du code source et versionnage

GitHub a servi de plateforme de partage de code collaboratif. L'usage de Git était essentiel pour centraliser le code source Python, gérer les versions et permettre un travail collaboratif en parallèle sans risque de perte de données. Cette méthodologie offre une traçabilité complète des apports de chaque membre, conformément aux exigences de rigueur du projet.

Surtout, GitHub ne s'est pas limité à l'hébergement du code : un pipeline d'intégration et de déploiement continus (CI/CD) via GitHub Actions génère automatiquement les exécutables du jeu à chaque nouvelle version taguée. Cette automatisation, détaillée à la section XVII, a considérablement fiabilisé la production des livrables.

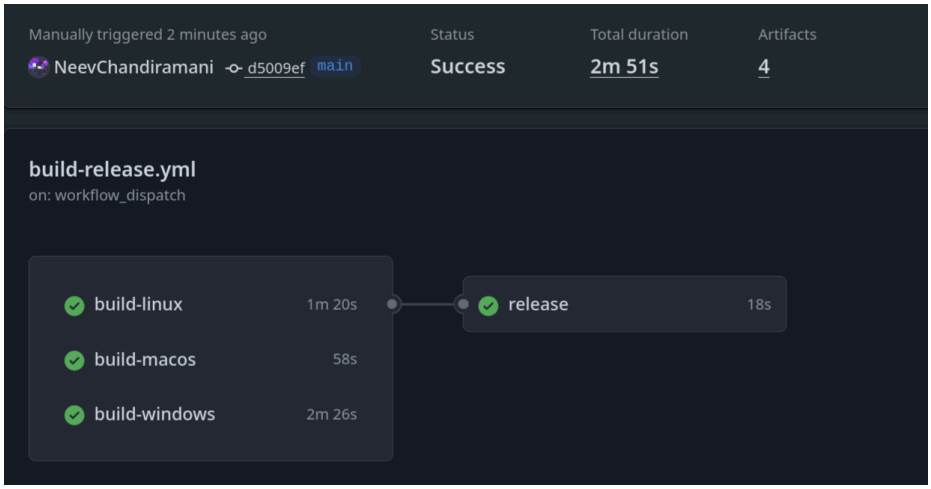


image illustrant la création des exécutables à l'aide de github actions

Méthode de travail

Le développement a été découpé en pôles techniques (graphisme, gameplay, réseau, audio) afin de paralléliser le travail. Chaque tâche s'est vu attribuer un responsable principal (noté R) et un secondant (noté S) chargé d'assurer la continuité du développement en cas d'indisponibilité. Cette redondance des compétences a notamment permis d'absorber sans rupture le départ de Maximilien Cochet en cours d'année.

V.Répartition des tâches et avancement

Five Nights At Châtelet a été découpé en plusieurs pôles techniques. À la suite du départ de Maximilien Cochet, la répartition a été revue pour redistribuer ses tâches et garantir que chaque membre apporte sa pierre à l'édifice. La répartition finale, telle qu'appliquée jusqu'à la soutenance finale, est la suivante.

Les pôles techniques

Pôle Graphisme et Environnement. La cohérence visuelle est portée par Gauthier Roland, responsable du map design et du character design, soutenu par Neev Chandiramani. Ce pôle crée l'atmosphère du jeu à travers l'architecture des niveaux, l'apparence des entités et les animations.

Pôle Technique et Gameplay. Maxime Gontran dirige les déplacements, la physique, le système de points de vie et le système d'attaque, avec l'appui de Maxime Ye. Maxime Ye est responsable du menu Options, du rebinding des touches et de la gestion des équipes, secondé par Maxime Gontran.

Pôle Multijoueur et Communication. Le réseau, élément obligatoire du projet, est piloté par Neev Chandiramani avec l'aide de Maxime Gontran. Le site web est géré par Neev Chandiramani et Gauthier Roland, et la chaîne de build / déploiement est assurée par Neev Chandiramani, secondé par Gauthier Roland.

Pôle Audio. L'immersion sonore, cruciale pour un jeu d'horreur, est confiée à Maxime Ye, responsable des sons, avec le soutien de Maxime Gontran.

Tâche	Responsable (R)	Secondant (S)
Map design / Character design	Gauthier Roland	Neev Chandiramani
Multijoueur / Réseau	Neev Chandiramani	Maxime Gontran
Sons / Audio	Maxime Ye	Maxime Gontran
Déplacement / Physique	Maxime Gontran	Maxime Ye
Combat / HP	Maxime Gontran	Maxime Ye
Menu / Options	Maxime Ye	Maxime Gontran

Tâche	Responsable (R)	Secondant (S)
Site Web	Neev Chandiramani	Gauthier Roland
UI / HUD	Maxime Gontran	Maxime Ye
Build / Déploiement	Neev Chandiramani	Gauthier Roland

Répartition des tâches en vigueur jusqu'à la soutenance finale

État d'avancement final

À la différence de la deuxième soutenance, où plusieurs fonctionnalités majeures restaient à faire (intelligence artificielle, énigmes, attribution automatique des rôles), l'ensemble des fonctionnalités principales du jeu est aujourd'hui terminé. Le tableau suivant dresse l'état final de chaque grande fonctionnalité.

Fonctionnalité	État
Menu principal (effets CRT, glitch, volume)	Terminé
Menu Options avec rebinding et anti-doublons	Terminé
Correction AZERTY / QWERTY	Terminé
Déplacement joueur (ZQSD + souris)	Terminé
Physique : gravité, saut, collisions multi-points	Terminé
Stamina et sprint	Terminé
Système HP, dégâts, invincibilité, respawn	Terminé
Attaque directionnelle (cône, produit scalaire)	Terminé
Modèles 3D personnages et map (Blender)	Terminé
Multijoueur réseau TCP sur serveur dédié	Terminé
Synchronisation position, rotation et animation	Terminé
Attribution automatique des rôles (lobby)	Terminé
Intelligence artificielle (Embuscadeur)	Terminé
Système d'énigmes (4 mini-jeux)	Terminé
Système de prison, libération et victoire	Terminé
Screamers et jump-scares	Terminé
Ambiance et effets sonores in-game	Terminé

Fonctionnalité	État
Exécutables Windows / Linux / macOS + installateur	Terminé
Site web avec téléchargement	Terminé

État d'avancement à la soutenance finale

VI.Architecture technique générale

Le projet combine deux moteurs complémentaires. Ursina Engine, bâti sur Panda3D, assure le rendu 3D temps réel, la gestion des entités et la boucle de jeu. PyGame gère l'habillage 2D (menu, options, overlays) et sert de moteur audio via son mixeur. Cette cohabitation a nécessité une attention particulière, notamment pour le passage du menu au jeu et pour le partage du contrôle de la fenêtre.

Organisation des fichiers

Le code est volontairement modulaire : chaque grande responsabilité est isolée dans son propre fichier, ce qui facilite le travail en parallèle et limite les conflits Git.

```
J3D/
  Five_Nights_At_Chatelet.py   # boucle principale (Ursina)
  Menu.py                      # menu principal et options (PyGame)
  server.py                    # serveur TCP multijoueur
  NetworkClient.py             # client reseau TCP
  Rooms.py                     # systeme de salles
  NavigoTask.py                # enigme : pass Navigo
  enigme_electrique.py         # enigme : tableau electrique
  enigme_plomberie.py          # enigme : plomberie
  EnigmeLabyrintheSignalisation.py  # enigme : signalisation
  Installer.iss                # installateur Inno Setup (Windows)
  ressources/                  # modeles .obj/.mtl, sons, images, screamers
  .github/workflows/build-release.yml  # CI/CD multi-OS
```

Gestion des chemins de ressources

Un point d'architecture déterminant concerne le chargement des ressources. En développement, les chemins sont relatifs au fichier source ; mais une fois le jeu compilé avec PyInstaller, les fichiers sont extraits dans un dossier temporaire. Une fonction utilitaire centralise cette résolution afin que le même code fonctionne en source comme en exécutable.

```
if getattr(sys, 'frozen', False):
    BASE_DIR = sys._MEIPASS          # execution depuis l'exe PyInstaller
else:
    BASE_DIR = os.path.dirname(os.path.abspath(__file__))

def res(path):
    return os.path.normpath(os.path.join(BASE_DIR, path))
```

Ce mécanisme, combiné à un changement explicite du répertoire de travail et à la configuration du dossier d'assets d'Ursina, a permis de résoudre les nombreux problèmes de chemins rencontrés lors du packaging (voir section XIX).

VII.Modèles 3D et environnement

Nous prenons en charge tout l'aspect graphique du jeu : la création de la map, des personnages, des animations, mais aussi leur implémentation directement en Python via Ursina. Tous les modèles sont conçus dans Blender puis exportés au format .obj/.mtl avant d'être chargés dans le moteur.

Les personnages

Deux personnages représentent les deux camps : un explorateur (le survivant) et un infecté. Le jeu a une ambiance volontairement horrifique, mais nous avons fait le choix d'un style graphique cartoon, avec de grosses têtes et de grands yeux un peu amusants. L'objectif était de créer un contraste entre le visuel des personnages et l'atmosphère oppressante du jeu, pour rendre l'ensemble plus marquant et plus original.

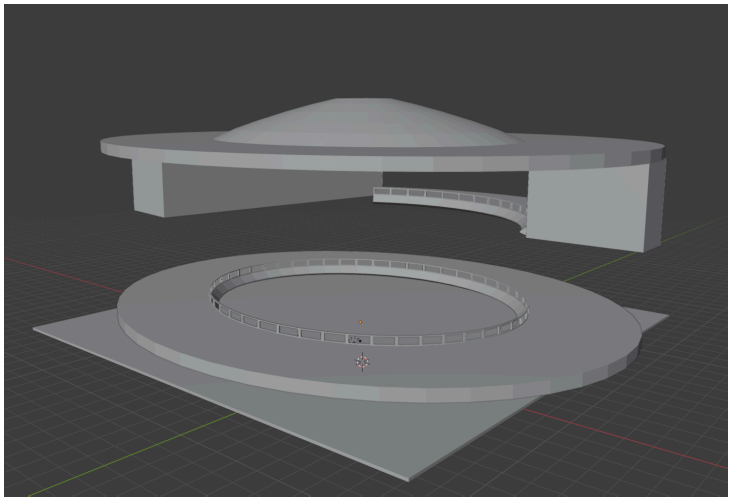
modèle 3D d'un « explorateur »

modèle 3D d'un « infecté »

Personnages articulés et animés. Les personnages ne sont pas de simples blocs : chaque modèle est découpé en six pièces (corps, tête, deux bras, deux jambes) assemblées sous un pivot. Cette structure permet une animation procédurale entièrement calculée en Python — balancement des bras et des jambes en marche et en course, légère respiration à l'arrêt, et geste d'attaque pour les infectés. Un travail de rig a également été réalisé en amont dans Blender, préparant les modèles à des animations plus poussées.

L'environnement : la station Châtelet

La map représente une station de métro de type centre commercial souterrain, librement inspirée de Châtelet–Les Halles. Elle a d'abord existé sous forme d'un squelette définissant les étages et leur organisation, avant d'être modélisée complètement puis importée dans le moteur comme une véritable scène jouable.

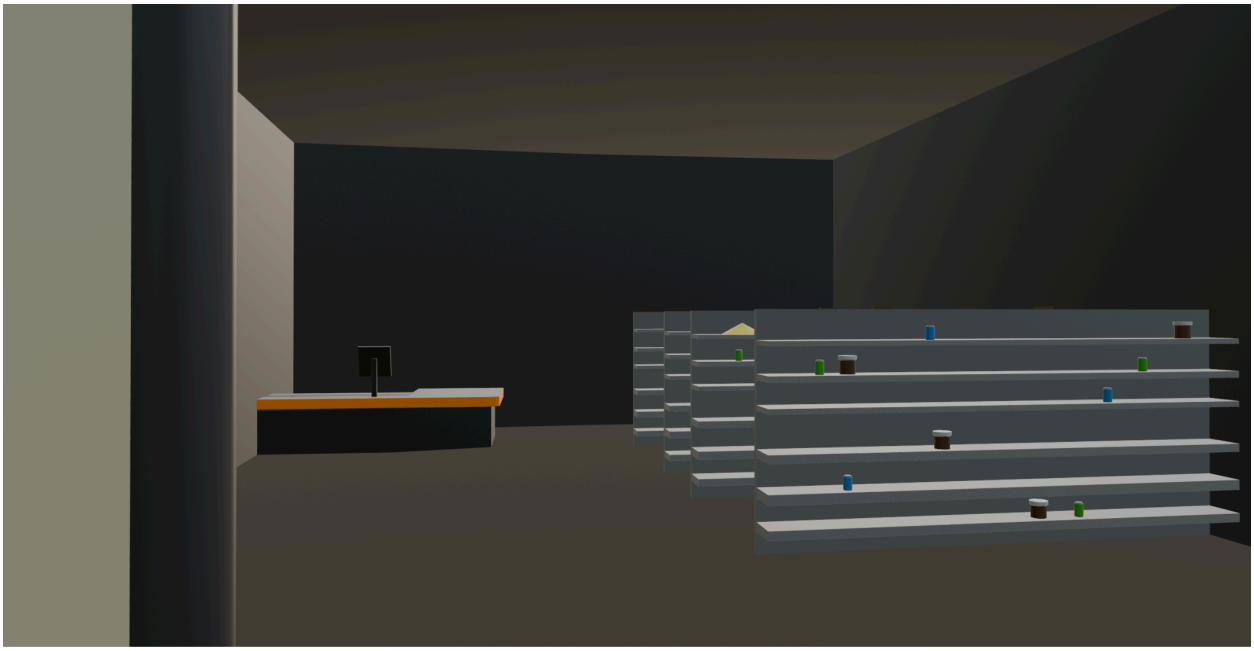


première structure de la map (vue d'ensemble)

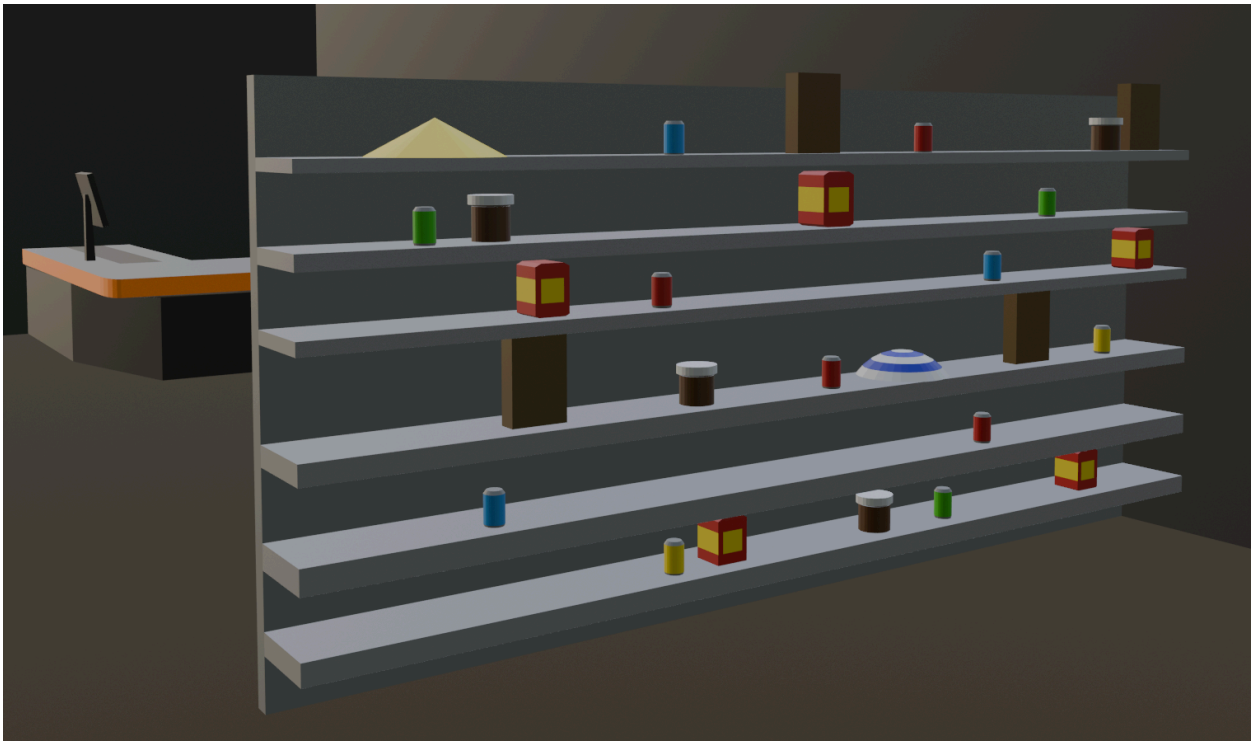
rendu de la map intégrée au moteur

La taille et la complexité de la map ont été maintenues raisonnables pour garantir une exécution fluide. Le mesh est relativement optimisé, avec un nombre de faces contenu, ce qui est essentiel pour un rendu temps réel. La map n'est pas seulement décorative : ses murs, sols et structures définissent l'espace, les déplacements possibles et la perception du joueur. Plusieurs panneaux et affiches texturés ont été disposés dans la station pour renforcer l'immersion et la cohérence de l'univers.

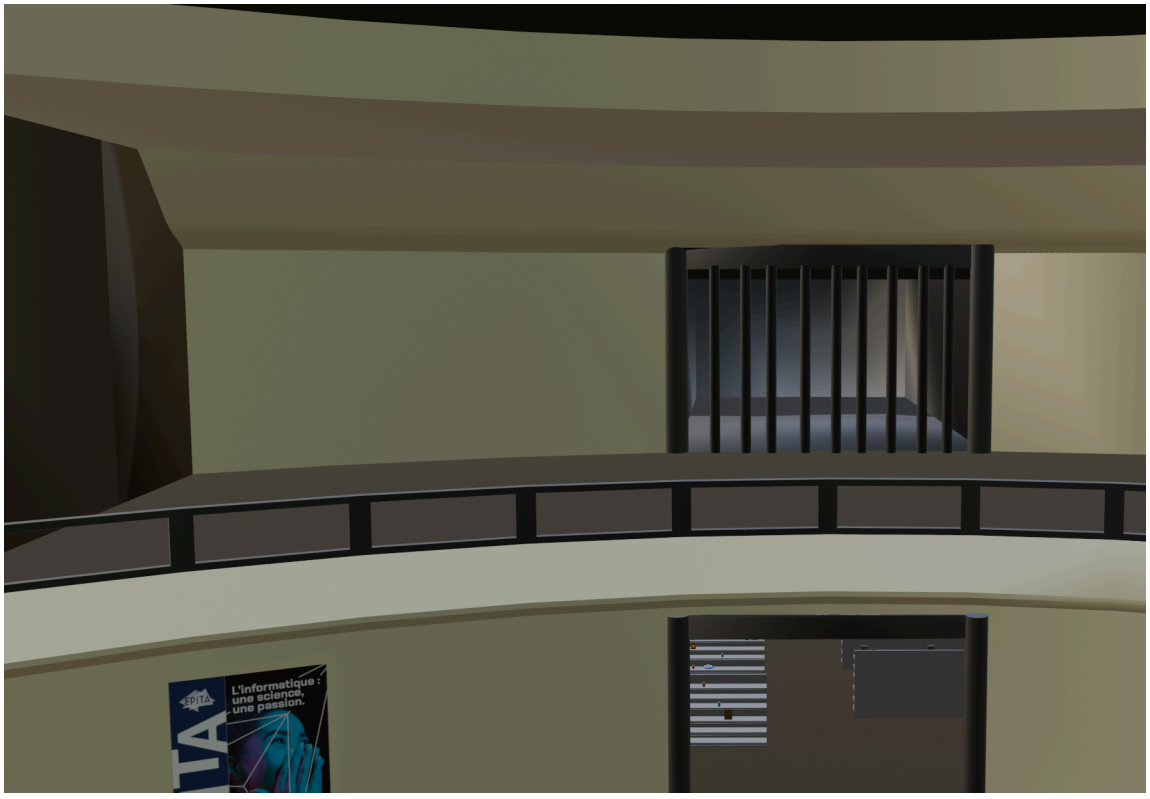
avancement de la map



une salle de la map



détails ajoutés à la map



prison de la map

VIII.Déplacement, caméra et physique

Le moteur de déplacement a été entièrement écrit à la main, sans recourir à un contrôleur prédéfini d'Ursina. Ce choix nous a donné un contrôle total sur le ressenti du jeu et nous a permis d'apprendre en profondeur le fonctionnement d'une physique de jeu, au prix de plusieurs itérations.

Caméra à la troisième personne

Lorsque le joueur entre en partie, le curseur est verrouillé et masqué. Une caméra est rattachée à un pivot lié au joueur, à hauteur de tête. Les mouvements de la souris font pivoter ce pivot ; la rotation horizontale est répercutée sur le personnage pour qu'il regarde dans la même direction que la caméra. La caméra est ensuite reculée derrière le personnage pour obtenir une vue à la troisième personne.

```
mouse.locked  = True
mouse.visible = False
camera_pivot = Entity(parent=joueur, y=2)
camera.parent = camera_pivot
camera.fov = 90

def mouvement_camera():
    camera_pivot.rotation_y += mouse.velocity[0] * 80
    camera_pivot.rotation_x -= mouse.velocity[1] * 80
    camera_pivot.rotation_x = clamp(camera_pivot.rotation_x, -30, 45)
    camera.position = (0, 0, -5)
```

La méthode clamp force l'angle vertical à rester dans l'intervalle [-30, 45], empêchant le joueur de regarder trop vers le ciel ou trop vers le sol. La sensibilité est réglée par le facteur 80.

Stamina et sprint

Le sprint est associé à un système d'endurance. En maintenant la touche dédiée, la vitesse augmente mais l'endurance diminue progressivement ; une fois épuisée, le sprint se désactive automatiquement, et l'endurance se régénère dès que le joueur cesse de courir. Un indicateur à l'écran affiche le niveau d'endurance en temps réel. Cette mécanique ajoute une dimension stratégique en empêchant le sprint illimité — et joue un rôle clé face à l'Embuscadeur, qui détecte le joueur d'autant plus loin qu'il court (section XI).

Saut et gravité

Le système de saut a été refondu pour reposer sur une physique basée sur la vélocité verticale et une gravité constante. Lorsque le joueur saute, une vitesse verticale initiale lui est appliquée, puis la gravité l'attire vers le bas frame après frame. Un raycast assure une détection du sol très précise, avec un « snap » fluide pour éviter les rebonds parasites, et empêche tout double saut tant que le personnage n'a pas atterri.

Collisions multi-points

Pour une navigation fluide dans les zones complexes de la station, le joueur teste ses collisions à l'aide de plusieurs raycasts simultanés (centre, gauche et droite, à deux hauteurs différentes). Cela lui permet de glisser le long des murs sans rester bloqué dans les angles. Des colliders invisibles — murs et rambardes — ont par ailleurs été ajoutés pour canaliser les déplacements dans les passages délicats de la map.

Création et visualisation des colliders (touche de debug K)

Une touche de débogage permet d'ailleurs de rendre visibles ces colliders pendant le développement, ce qui a grandement facilité le réglage de la map et la chasse aux zones où le joueur pouvait rester coincé ou traverser le décor.

IX.Système de rôles et attribution automatique

L'attribution automatique des rôles, qui restait « à faire » à la deuxième soutenance, est désormais pleinement opérationnelle. Elle est pilotée par le serveur, qui garantit une répartition cohérente entre tous les joueurs d'une même partie.

Définition centralisée des rôles

Chaque rôle est décrit dans une table unique regroupant ses statistiques, ses modèles 3D et ses capacités. Cette centralisation rend le code lisible et facilite tout futur rééquilibrage.

```
ROLES = {
    'Survivor': {
        'description': 'Find the exits. Avoid the Infected.',
        'model_path': 'ressources/Perso.obj',
        'speed_mult': 1.0,  'max_hp': 100, 'can_attack': False,
    },
    'Infected': {
        'description': 'Eliminate all Survivors.',
        'model_path': 'ressources/Crackhead.obj',
        'speed_mult': 1.15, 'max_hp': 150, 'can_attack': True,
    },
}
```

Le lobby et le tirage des rôles

Au démarrage, un écran de connexion fait office de lobby. Chaque joueur peut se déclarer « prêt » ; le serveur diffuse en continu l'état du lobby (nombre de joueurs prêts sur le total). Dès que tous les joueurs présents sont prêts — ou qu'un joueur demande un démarrage forcé — le serveur tire les rôles une seule fois et les fige pour toute la partie.

image du lobby

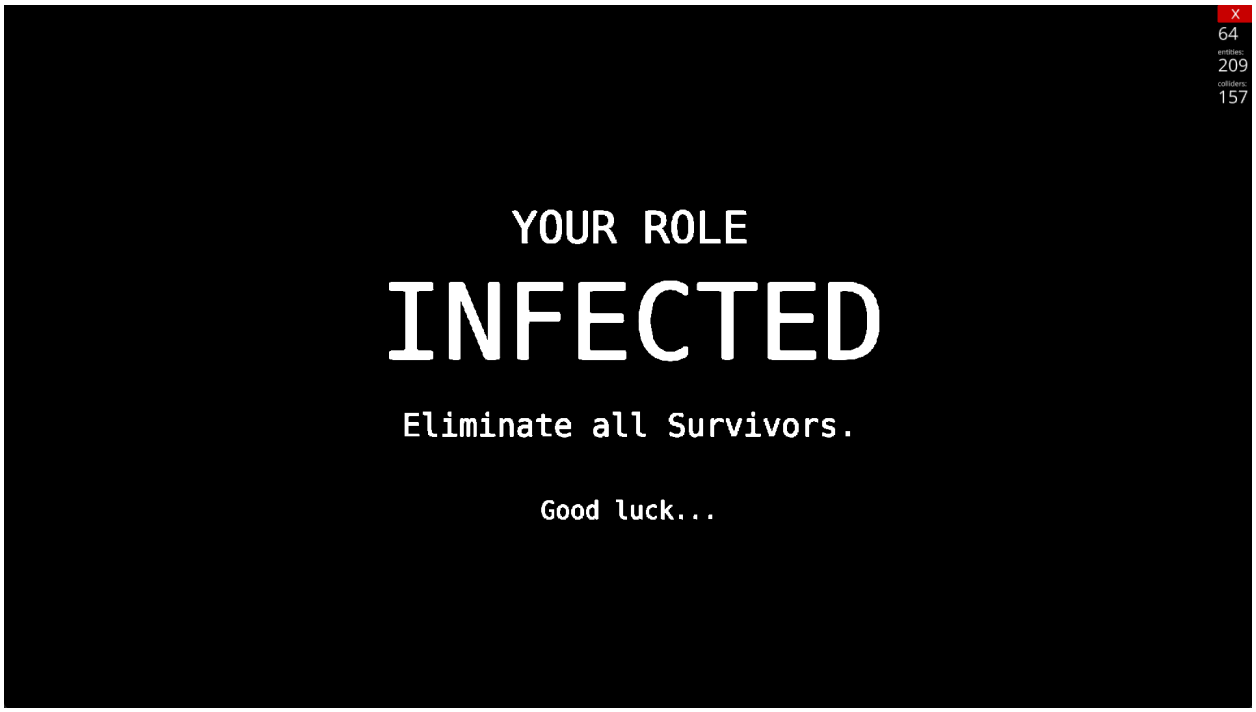


image affichant l'assignation des rôles

Proportion d'infectés. En multijoueur, le serveur désigne environ 45 % des joueurs comme infectés, avec au minimum un infecté et au moins un survivant. En solo, le même tirage probabiliste s'applique. Une fois les rôles décidés, le serveur les diffuse à tous les clients, qui appliquent localement leurs statistiques, chargent les bons modèles 3D et déclenchent l'animation d'annonce de rôle.

```
nb_infectes = max(1, min(len(pids) - 1,
                  round(len(pids) * INFECTED_RATIO)))
infectes = set(random.sample(pids, nb_infectes))
roles = {p: ('Infected' if p in infectes else 'Survivor')
         for p in pids}
```

Le système est robuste aux arrivées tardives : un joueur qui rejoint une partie déjà lancée se voit attribuer le rôle de survivant par défaut, et la liste complète des rôles est rediffusée à tous pour que chaque client affiche les bons modèles. Lorsqu'un joueur change de rôle, seul son « fantôme » (la représentation des autres joueurs côté client) est reconstruit, évitant tout recalcul inutile.

X.Les énigmes des survivants

Le cœur de l'objectif des survivants repose sur quatre énigmes coopératives, chacune implémentée dans son propre fichier Python et déclenchée par interaction (touche d'interaction) avec un objet du décor. Chaque survivant se voit assigner quatre tâches dont les emplacements sont répartis dans la station. Une fois ses quatre tâches accomplies, le survivant prévient le serveur ; quand tous les survivants ont terminé, le mur de la victoire apparaît et la sortie s'ouvre.

Placement déterministe des énigmes

Le placement des tâches est calculé en arrière-plan, dans un thread séparé, pour ne pas figer le démarrage du jeu. Un générateur aléatoire initialisé par l'identifiant du joueur garantit que chaque joueur reçoit une disposition stable et reproductible, tout en respectant une distance minimale entre les salles choisies afin d'éviter de concentrer toutes les énigmes au même endroit.

```
def choisir_salles_tasks(player_id):
    rng = random.Random(int(player_id))
    tasks = ['navigo','vanne','electrique','panneau',
             'screamer1','screamer2']
    for _ in range(1000):
        choix = rng.sample(range(len(SALLES)), 6)
        # ... rejet si deux salles sont trop proches
        if valide:
            return dict(zip(tasks, [SALLES[i] for i in choix]))
```

Les quatre mini-jeux

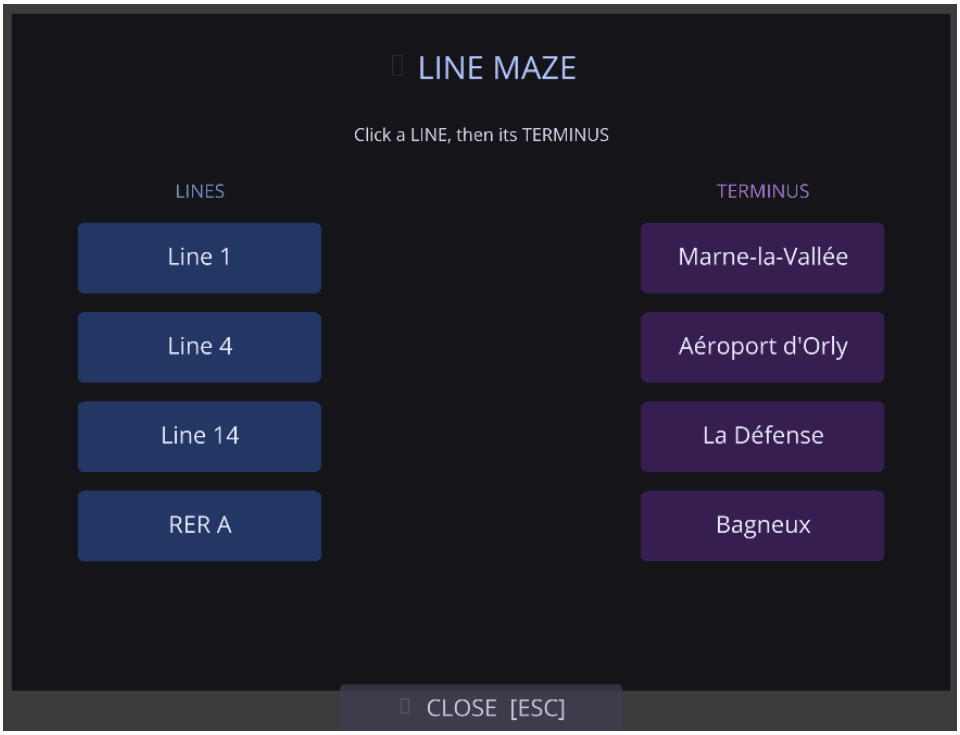
1. Le pass Navigo. Un mini-jeu de « swipe » inspiré d'Among Us : le joueur doit valider sa carte Navigo en effectuant un glissement à la bonne vitesse, ni trop rapide ni trop lent, dans une fenêtre de durée précise.

2. Le tableau électrique. Un casse-tête d'interrupteurs : cliquer sur un interrupteur en bascule trois (lui-même, son voisin de gauche et son voisin de droite). L'objectif est d'allumer les cinq interrupteurs simultanément, ce qui demande de la réflexion sur l'ordre des actions.

3. La plomberie. Un mini-jeu de timing : une aiguille oscille de haut en bas dans une jauge, et le joueur doit cliquer lorsqu'elle se trouve dans la zone verte. Trois

réussites consécutives valident la purge ; chaque échec accélère l'aiguille et retire un succès, augmentant la difficulté.

4. Le labyrinthe de signalisation. Un mini-jeu d'association à la manière du « wiring » d'Among Us, mais sur le thème de la signalétique RATP : le joueur relie chaque ligne à son terminus par des clics successifs (gauche pour la ligne, droite pour le terminus), sans glisser-déposer.



Les quatre énigmes coopératives des survivants

Validation et condition de victoire

Chaque énigme reçoit une fonction de rappel (callback) appelée à sa résolution, qui incrémente le compteur de tâches accomplies du joueur. Le suivi est robuste aux validations multiples : une énigme déjà résolue ne peut pas être comptée deux fois.

```
def valider_une_tache():
    global mes_taches_accomplies, mon_statut_fini
    mes_taches_accomplies += 1
    if mes_taches_accomplies >= total_de_mes_taches:
        mon_statut_fini = True
        network.sock.sendall((json.dumps(
            {'type': 'survivant_fini', 'id': network.my_id}) + '\n').encode())
```

Lorsque tous les survivants connectés ont signalé la fin de leurs tâches, un mur vert semi-transparent apparaît : il matérialise l'ouverture de la sortie. Le premier survivant à le franchir déclenche la victoire de tout le camp, suivie d'un retour automatique au menu après quelques secondes.

XI.Intelligence artificielle : l'Embuscadeur

L'intelligence artificielle, fonctionnalité majeure restée « à faire » jusqu'à la deuxième soutenance, est désormais implémentée. Elle prend la forme de l'Embuscadeur : une entité hostile qui se repère seule dans la station et réagit de manière autonome au comportement du joueur. Elle satisfait pleinement la contrainte d'IA du cahier des charges.

Détection par le bruit

L'originalité de l'Embuscadeur est qu'il ne « voit » pas le joueur : il l'entend. Son rayon de détection dépend du bruit que produit le joueur. À l'arrêt, le joueur est totalement silencieux, donc indétectable ; en marche, il est repéré à environ 9 unités ; en sprint, le rayon grimpe à 18 unités. Cette mécanique crée une tension constante et récompense une approche prudente, en lien direct avec le système d'endurance.

État du joueur	Rayon de détection	Conséquence
Immobile	0 (silencieux)	Indétectable, l'Embuscadeur reste caché
En marche	≈ 9 unités	Risque de déclencher la tension
En sprint	≈ 18 unités	Très exposé, détection à grande distance

Détection de l'Embuscadeur en fonction du bruit produit

La machine à états

L'Embuscadeur fonctionne comme une machine à états finis comportant cinq états : caché, tension, bond, refroidissement et repositionnement. Chaque joueur possède son propre Embuscadeur, dont l'ordre des positions d'embuscade est mélangé à partir de l'identifiant du joueur, garantissant une expérience unique.

- **Caché.** L'Embuscadeur est invisible et immobile à une position d'embuscade. Si le joueur entre dans son rayon de détection (donc s'il fait du bruit), il passe en tension.

- **Tension.** Une courte temporisation s'enclenche. Si le joueur s'arrête ou s'éloigne avant la fin, l'Embuscadeur se rendort. Sinon, l'attaque se déclenche.
- **Bond.** L'Embuscadeur apparaît, s'oriente vers le joueur et se précipite sur lui en quelques fractions de seconde, accompagné d'un screamer et de dégâts. Le joueur est immobilisé brièvement.
- **Refroidissement puis repositionnement.** Après l'attaque, l'Embuscadeur disparaît, observe un temps de pause, puis se déplace silencieusement vers une nouvelle position d'embuscade avant de redevenir caché.

```
if _emb_state == 'hidden':
    if detect_r > 0 and dist <= detect_r:
        _emb_state = 'tension'
        _emb_tension_timer = EMBUSCADE_TENSION_TIME
elif _emb_state == 'tension':
    if detect_r == 0 or dist > detect_r:
        _emb_state = 'hidden'      # le joueur s'est arrete a temps
    else:
        _emb_tension_timer -= time.dt
        if _emb_tension_timer <= 0:
            _emb_trigger()         # BOND + screamer + degats
```

L'Embuscadeur : attaque autonome déclenchée par le bruit

XII.Combat, santé, prison et libération

Points de vie et dégâts

Chaque joueur dispose d'une jauge de points de vie (100 pour les survivants, 150 pour les infectés) affichée dans le HUD, qui change de couleur selon le niveau restant : verte au-dessus de 50 %, orange entre 25 et 50 %, rouge en dessous. Lorsqu'un coup est reçu, une période d'invincibilité d'une demi-seconde empêche les morts instantanées, et un flash rouge couvre brièvement l'écran pour signaler l'impact.

Attaque directionnelle

Seuls les infectés peuvent attaquer. L'attaque n'est pas une simple détection de distance : elle utilise le produit scalaire pour vérifier que la cible se trouve bien dans un cône orienté devant l'attaquant. Trois conditions doivent être réunies —

distance frontale dans la portée, écart latéral sous une certaine largeur, et différence de hauteur acceptable.

```
forward = Vec3(joueur.forward.x, 0, joueur.forward.z).normalized()
right   = Vec3(joueur.right.x,   0, joueur.right.z).normalized()
for pid, ghost in ghost_entities.items():
    delta  = ghost.position - joueur.position
    f_dist = delta.dot(forward)   # distance devant
    s_dist = abs(delta.dot(right))# ecart lateral
    if (0 < f_dist <= ATTACK_RANGE) and (s_dist <= ATTACK_WIDTH*0.5):
        ghost_hp[pid] -= ATTACK_DAMAGE
        network.send_damage(pid, ATTACK_DAMAGE)
```

Une boîte de collision rouge semi-transparente est brièvement affichée à chaque coup pour visualiser le cône, et la cible touchée clignote en rouge. Côté réseau, chaque coup est transmis au serveur, qui le relaie à la victime ; ainsi tous les joueurs voient le même résultat.

Mort, prison et libération

Lorsqu'un survivant tombe à zéro point de vie, il n'est pas éliminé définitivement mais emprisonné : il est téléporté dans une cellule, ses mouvements sont bloqués et le serveur en informe les autres joueurs. Un survivant encore libre peut alors venir actionner un levier près de la cellule pour libérer son coéquipier, qui réapparaît avec ses points de vie restaurés et une brève invincibilité.

Conditions de fin. Si tous les survivants sont emprisonnés en même temps, les infectés gagnent et un message de défaite s'affiche côté survivants (de victoire côté infectés). À l'inverse, si les survivants résolvent toutes leurs énigmes et franchissent le mur de la sortie, ils s'échappent et remportent la partie. Dans les deux cas, le jeu revient automatiquement au menu après un court délai.

XIII.Multijoueur en réseau

Le multijoueur constituait déjà l'évolution majeure de la deuxième soutenance. Il a depuis été consolidé et enrichi pour transmettre, en plus des positions, les rotations, les états d'animation, les rôles, les évènements de combat, l'état de la prison et les conditions de victoire.

Architecture client-serveur

Le projet repose sur une architecture client-serveur TCP. Un serveur dédié, indépendant du moteur graphique, écoute sur le port 5555 et gère les connexions, le lobby, l'attribution des rôles et la synchronisation de l'état du jeu. Chaque client se connecte automatiquement au démarrage. Le serveur a été hébergé sur un VPS dédié, accessible via une adresse fixe, ce qui permet de jouer entre machines distantes et pas seulement en réseau local.

Protocole d'échange

Chaque message échangé est un objet JSON terminé par un saut de ligne, ce qui permet de découper proprement le flux TCP en messages individuels. Plusieurs types de messages circulent sur le réseau :

- **Position et animation :** envoyés par chaque client toutes les 50 ms, ils contiennent la position, la rotation Y et des indicateurs d'état (en mouvement, en sprint, en attaque) pour synchroniser aussi les animations des autres joueurs.
- **Évènements ponctuels :** dégâts de combat, déclenchement de screamers, fin des tâches d'un survivant, emprisonnement et libération, attribution des rôles, état du lobby.

```
# Message de position (client -> serveur, toutes les 50 ms)
{ 'x': 15.2, 'y': 3.0, 'z': 0.5, 'ry': 90,
  'mv': 1, 'sp': 0, 'atk': 0, 'at': 1.24 }

# Message de degats (relaye par le serveur a la cible)
{ 'type': 'damage', 'target_id': 'a1b2c3d4',
  'amount': 25, 'attacker_id': 'e5f6g7h8' }
```

Côté client : robustesse

Côté client, un thread de réception tourne en arrière-plan pour ne jamais bloquer le rendu du jeu. Un timeout de connexion de trois secondes permet au jeu de basculer automatiquement en mode solo si le serveur est injoignable, garantissant une expérience satisfaisante même hors ligne. Le client maintient à jour la liste des autres joueurs, la file des dégâts reçus, les évènements de partie, l'état du lobby et les rôles attribués.

Les « fantômes » et la synchronisation

Chaque joueur distant est représenté localement par un « fantôme » articulé, reconstruit à l'identique de la structure du joueur local (six pièces sous un pivot). À chaque trame, leurs positions, rotations et états d'animation reçus du serveur sont appliqués, rendant leurs déplacements naturels. Lorsqu'un fantôme est touché, il clignote en blanc pour tous les utilisateurs connectés, confirmant visuellement le coup.

Cette séparation stricte entre logique réseau et rendu graphique améliore la modularité du projet et limite les dépendances : modifier le jeu n'oblige pas à toucher au serveur, qui reste fonctionnel en permanence. En contrepartie, tous les joueurs doivent utiliser la même version du jeu pour garantir la cohérence des échanges.

Synchronisation de plusieurs joueurs en partie

XIV.Menu, interface et identité visuelle

Le menu principal, entièrement réalisé sous PyGame, pose l'identité du jeu dès le lancement. Il s'appuie sur une image de fond inspirée du métro parisien, recouverte d'un voile sombre garantissant la lisibilité tout en installant immédiatement une ambiance pesante.

Identité visuelle : l'esthétique caméra de surveillance

Pour coller à l'univers horrifique, nous avons développé un rendu de type caméra de surveillance. Trois effets se superposent : un bruit visuel (grain / neige), un balayage de lignes horizontales évoquant un écran cathodique (CRT), et des glitchs aléatoires sur les textes qui suggèrent un matériel défaillant ou une présence paranormale. Cet ensemble renforce l'immersion dès l'écran-titre.

menu principal avec effets CRT et glitch

menu Options avec rebinding des touches

Navigation et menu Options

La navigation est hybride : le menu répond aussi bien aux flèches du clavier qu'à la souris, et permet de sélectionner intuitivement les options principales (Start, Options, Exit). À la deuxième étape, le menu Options a été rendu pleinement fonctionnel, et non plus seulement esthétique :

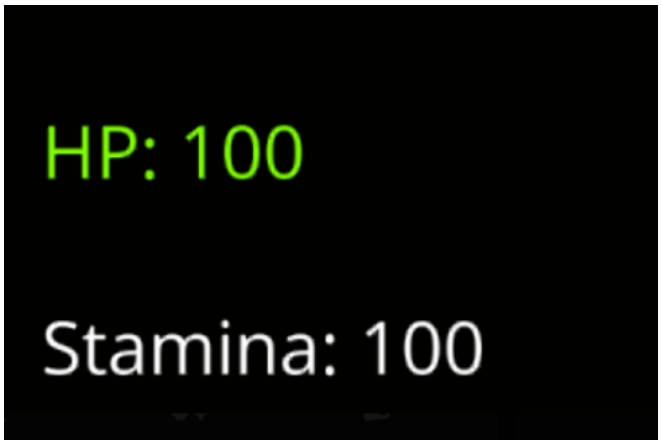
- **Rebinding des touches.** Le joueur peut choisir ses propres touches pour se déplacer, sauter et interagir.
- **Sécurité anti-doublons.** Le menu empêche d'assigner la même touche à deux actions différentes, ce qui éviterait de bloquer le personnage en jeu.
- **Sauvegarde JSON.** Au lancement de la partie, les préférences de touches sont écrites dans un fichier config_touches.json, rechargé au démarrage du jeu 3D.
- **Correction AZERTY / QWERTY.** Une table de correspondance assure la compatibilité entre les noms de touches PyGame et Ursina pour les claviers français — un point qui a demandé un soin particulier, les deux moteurs ne nommant pas les touches de la même façon.

```
azerty_to_ursina = {'a':'q','q':'a','z':'w','w':'z','m':';', ...}
binds_export = {k: azerty_to_ursina.get(pygame.key.name(v),
                                        pygame.key.name(v))
                for k, v in menu.keybinds.items()}
with open('config_touches.json', 'w') as f:
    json.dump(binds_export, f)
```

Le HUD en jeu

Le HUD (Head-Up Display) affiche en temps réel les informations essentielles : le niveau d'endurance, les points de vie colorés, un indicateur de cooldown d'attaque dans le coin bas-droit, un indicateur d'état réseau (vert si connecté avec le nombre de joueurs, rouge sinon), et l'overlay rouge de dégâts. Une aide

contextuelle s'affiche en outre lorsque le joueur s'approche d'une tâche, et un écran d'aide complet récapitule les commandes.



hud de HP et Stamina

hud menu d'aide (touche H)

XV.Ambiance sonore

Le son est l'un des piliers du genre survival-horror, et l'immersion sonore a fait l'objet d'un travail de fond. Le moteur audio repose sur le mixeur de PyGame, organisé en plusieurs canaux dédiés (ambiance, battements de cœur, respiration de l'infecté, bruits de pas) afin de gérer plusieurs sons simultanés sans qu'ils se coupent les uns les autres.

Sons d'ambiance et sons réactifs

Une nappe d'ambiance et des sons d'environnement (échos, grincements, chuchotements) tournent en fond pour entretenir l'atmosphère oppressante. Surtout, plusieurs sons réagissent à l'état du joueur :

- **Battements de cœur** qui se déclenchent en boucle dès que les points de vie passent sous 30 %, accentuant le sentiment de danger.
- **Respiration rauque** jouée en continu lorsque le joueur incarne un infecté, renforçant l'incarnation du rôle.
- **Chuchotements lointains** déclenchés aléatoirement à intervalles variables, ainsi qu'un son d'ambiance de gare se manifestant périodiquement pour rythmer l'exploration.
- **Sons d'action** associés au saut, à l'attaque, aux dégâts reçus, à la mort et aux interactions, qui donnent du poids à chaque geste du joueur.

Contrôle du volume

Une barre de volume interactive, présente dans le menu, est reliée directement au mixeur de PyGame, permettant au joueur de régler le son en temps réel, au clavier comme à la souris.

XVI.Screamers et jump-scares

Aucun jeu d'horreur ne serait complet sans ses jump-scares. Les screamers de Five Nights at Châtelet combinent une image plein écran et un son violent, choisis aléatoirement dans une banque dédiée. Ils peuvent être déclenchés de plusieurs façons : par l'Embuscadeur lorsqu'il bondit, par certains éléments piégés du décor que le survivant active sans le vouloir, et en réseau lorsqu'un screamer est diffusé à l'ensemble des joueurs.

```
def play_screamer(data):
    img_path, snd_path = data.split('|', 1)
    overlay = Entity(model='quad', texture=img_path,
                     scale=(camera.aspect_ratio*2, 2), parent=camera.ui)
    pygame.mixer.Sound(res(snd_path)).play()
    destroy(overlay, delay=1.15)
```

L'overlay est détruit automatiquement après un peu plus d'une seconde, et l'audio est volontairement lu via le mixeur PyGame plutôt que par Ursina, pour une lecture immédiate et fiable. Lorsqu'un screamer du décor est déclenché, le joueur est en outre brièvement immobilisé, ce qui amplifie la surprise et la vulnérabilité.

Un screamer déclenché par un élément piégé du décor

XVII.Build, packaging et déploiement

La distribution du jeu fait l'objet d'une infrastructure de build automatisée, l'un des points dont nous sommes les plus fiers. Conformément aux exigences du sujet, le projet est livré sous forme d'exécutables prêts à lancer, sans qu'aucune installation de Python ne soit nécessaire pour l'utilisateur final.

Intégration continue via GitHub Actions

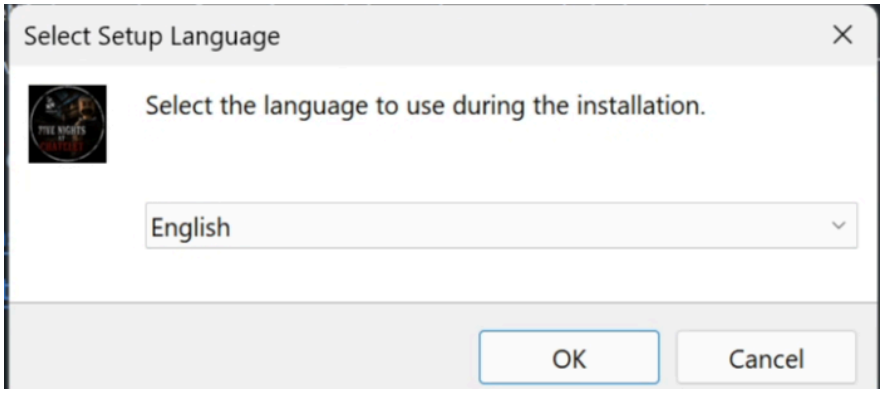
Un workflow GitHub Actions se déclenche à chaque nouvelle version taguée (ou manuellement). Il lance trois machines virtuelles en parallèle — Windows, Linux et macOS — qui installent les dépendances, compilent le jeu avec PyInstaller, puis publient automatiquement les binaires dans une release GitHub téléchargeable. Sur la machine Windows, un installateur Inno Setup est compilé en supplément.

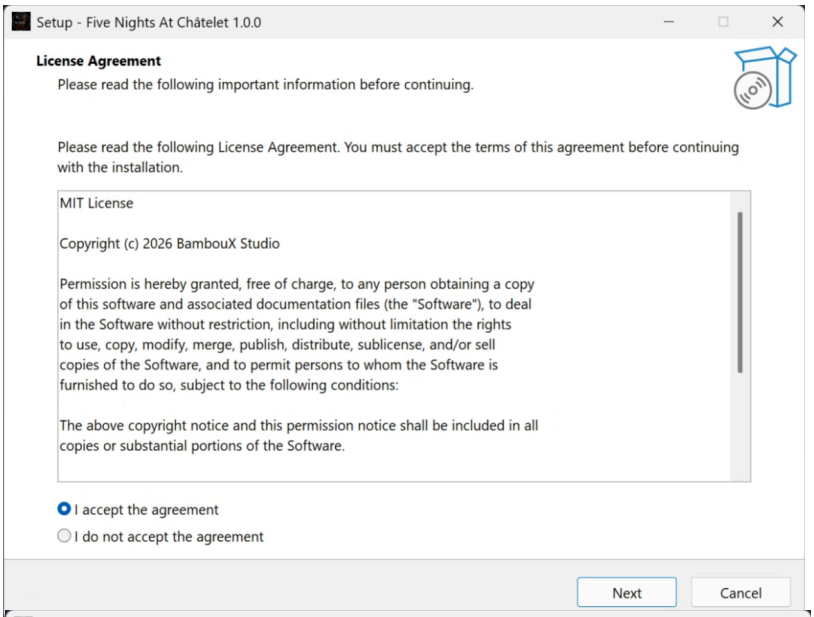
Fichier généré	Plateforme / Type
FiveNightsAtChatelet-Setup-X.Y.Z.exe	Windows — installateur (recommandé)
FiveNightsAtChatelet-Windows.exe	Windows — portable
FiveNightsAtChatelet-Linux	Linux — portable
FiveNightsAtChatelet-macOS	macOS — portable (ne fonctionne pas pour le moment)

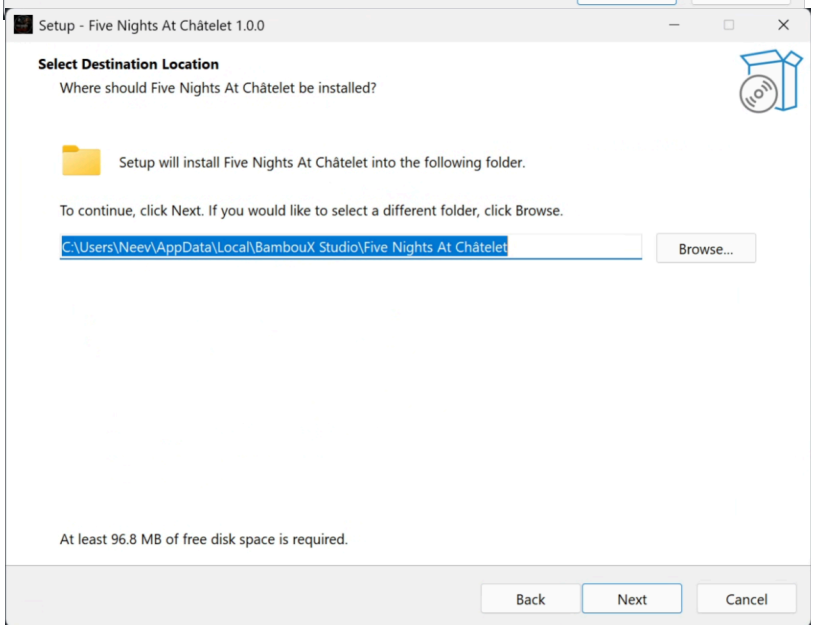
Les quatre livrables produits automatiquement par la CI

L'installateur Windows

L'installateur, décrit par un script Inno Setup, fournit un assistant en français et en anglais, le choix des composants (le jeu, et une copie locale optionnelle du site web), des raccourcis dans le menu Démarrer et sur le bureau, ainsi qu'une entrée propre dans les Paramètres de Windows pour la désinstallation. Une procédure de désinstallation complète est ainsi disponible aussi bien depuis le dossier d'installation que depuis l'utilitaire de désinstallation de Windows, comme l'exige le cahier des charges.





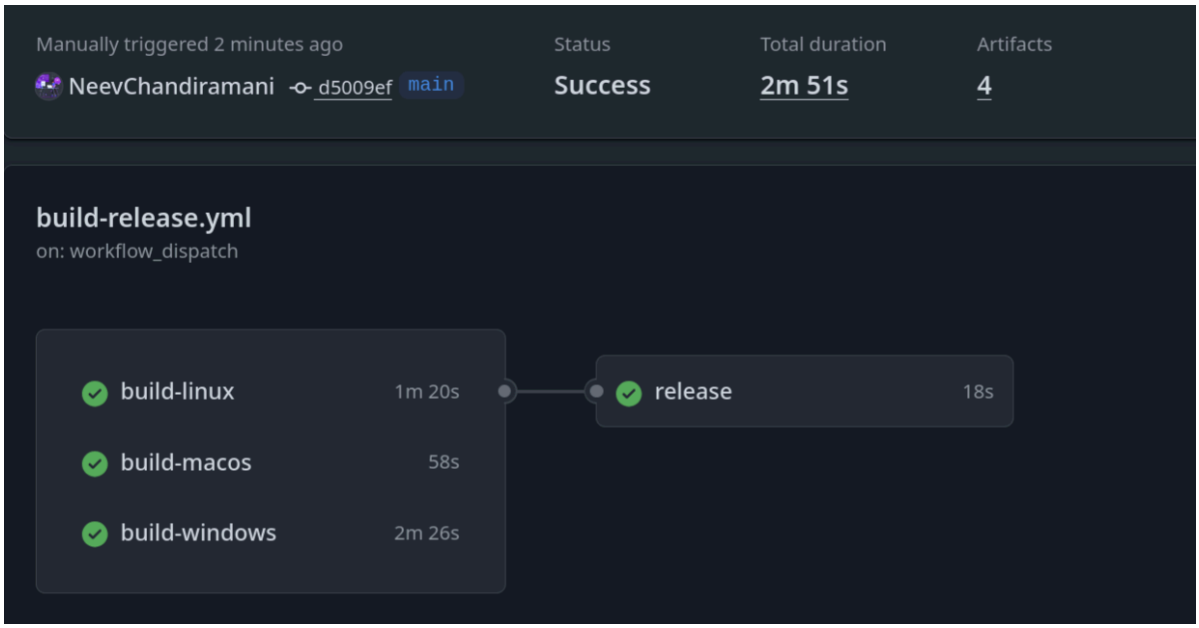


début des étapes d'installation du jeu

La commande de build

La compilation PyInstaller embarque l'intégralité des ressources et force la collecte des modules d'Ursina et de Panda3D, tout en excluant un module inutilisé afin d'éviter une incompatibilité d'architecture sur macOS.

```
pyinstaller --onefile --windowed --icon=icon.ico \
  --add-data 'ressources:ressources' \
  --collect-all panda3d --collect-all ursina \
  --hidden-import panda3d.core --exclude-module panda3d.rocket \
  --name 'FiveNightsAtChatelet-Linux' Five_Nights_At_Chatelet.py
```

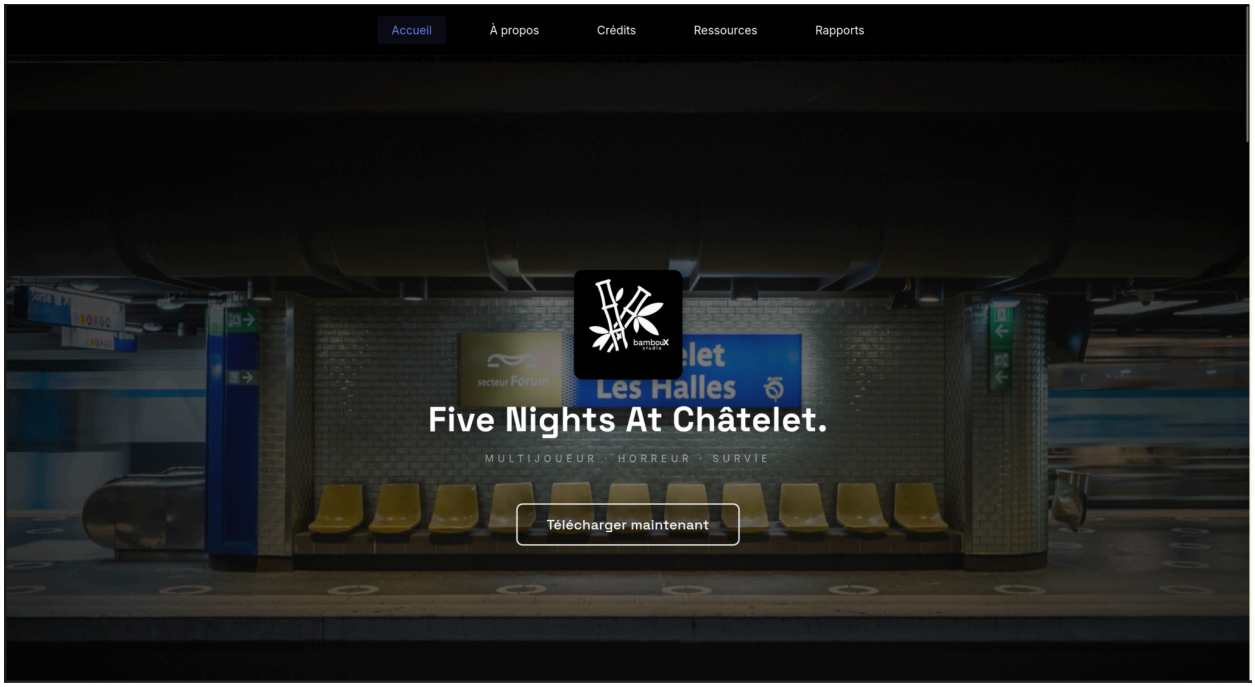


XVIII.Site web

Conformément aux exigences du cahier des charges, un site internet fait office de vitrine et de centre de ressources pour le projet. Mis à jour à chaque étape, il est accessible à l'adresse suivante :

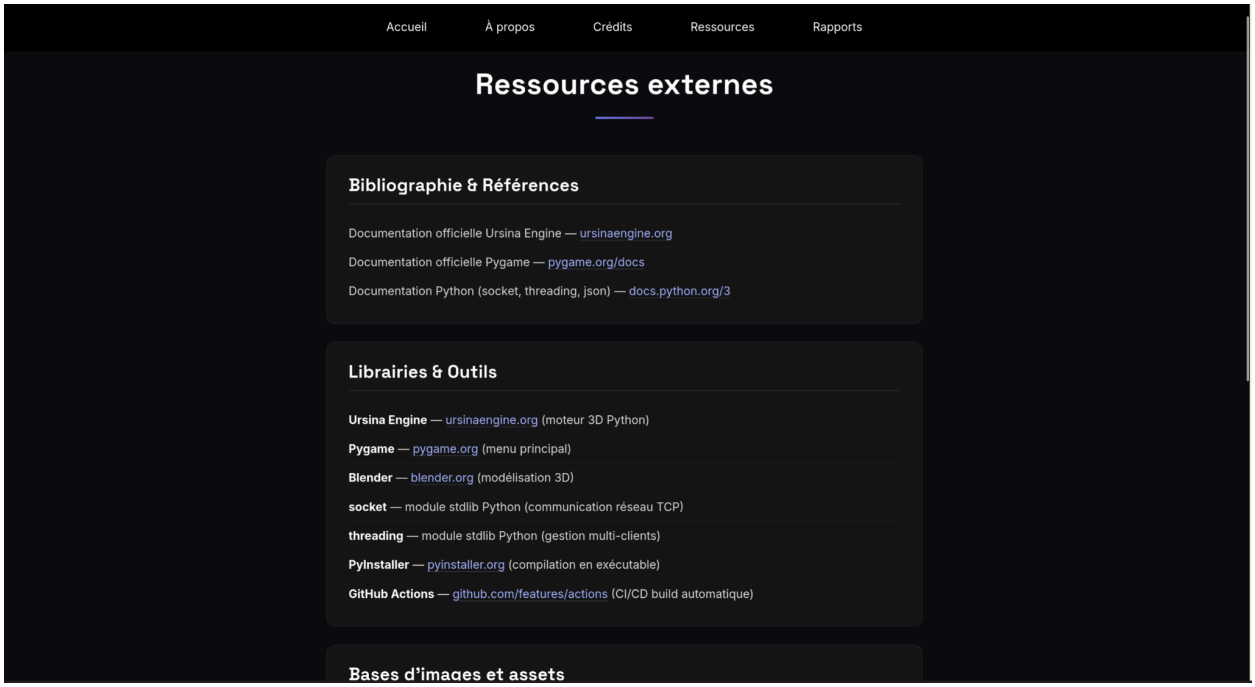
https://fivenightsatchatelet.neevchandiramani.com/

Le site remplit trois fonctions essentielles : une présentation détaillée du concept et de l'équipe (avec les rôles mis à jour après le départ de Maximilien Cochet), un centre de documentation regroupant toutes les ressources externes mobilisées, et une plateforme de téléchargement donnant accès direct aux exécutables compilés et aux rapports de soutenance.



page d'accueil du site

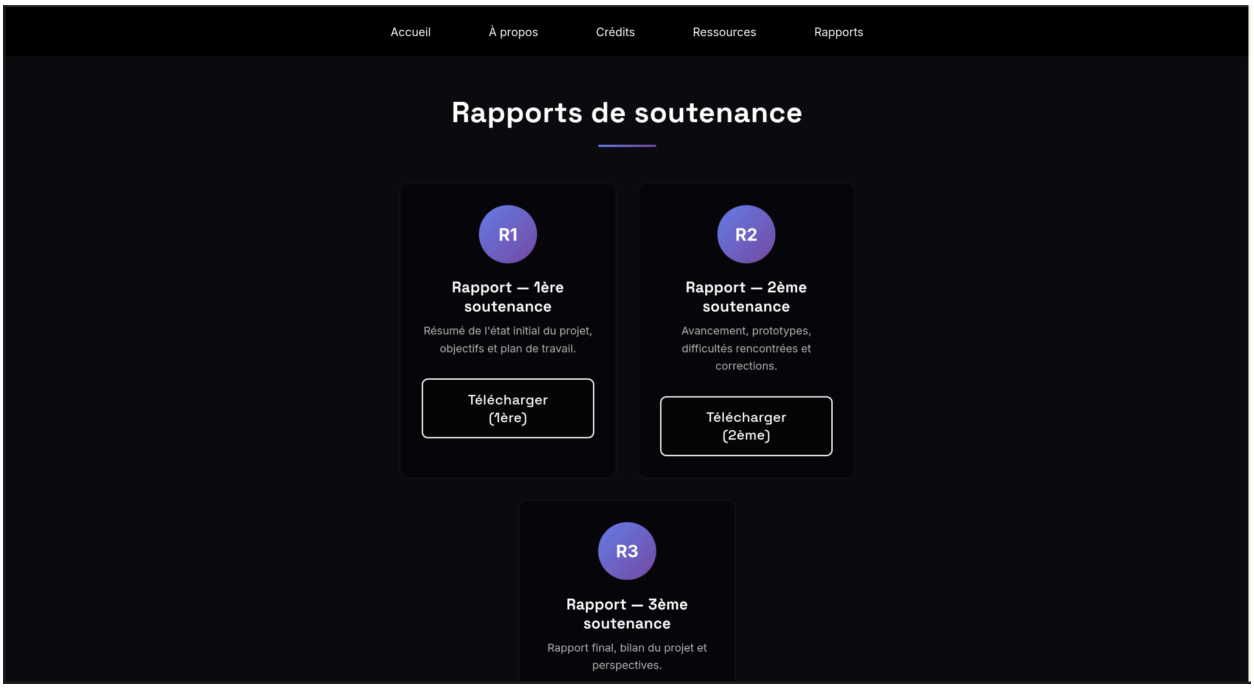
La section Ressources liste précisément les bibliothèques et outils utilisés (Ursina Engine, PyGame, Blender, modules standard socket / threading / json de Python, PyInstaller, GitHub Actions) ainsi que les banques d'assets sonores et graphiques, avec leurs liens. Cette transparence sur les ressources externes répond directement à l'exigence du sujet.



page des ressources externes du jeu

Le site est développé en HTML, CSS et JavaScript, avec une charte graphique sombre cohérente avec l'univers du jeu, et déployé automatiquement. Une copie locale fonctionnelle est par ailleurs distribuée avec l'installateur, comme demandé dans le cahier des charges.

page à propos du jeu



page de téléchargement des rapports de soutenance

XIX.Récit de la réalisation : joies et peines

Au-delà du résultat technique, ce projet a été une aventure humaine et un véritable apprentissage. Cette section en retrace le déroulement, avec ses satisfactions comme ses difficultés.

Nos joies

La plus grande satisfaction fut sans doute le moment où le multijoueur a fonctionné pour la première fois entre deux machines distantes : voir un coéquipier se déplacer en temps réel dans la station, avec ses animations synchronisées, a concrétisé des semaines de travail sur le réseau. La mise en place de la chaîne de build automatique fut une autre grande fierté : taguer une version et récupérer quelques minutes plus tard trois exécutables et un installateur a donné au projet une vraie dimension professionnelle.

L'Embuscadeur (l'IA) a également été l'une des fonctionnalités les plus gratifiantes à concevoir. Son système de détection au bruit transforme complètement le ressenti du jeu et crée exactement la tension que nous recherchions.

Nos peines

Le départ d'un membre. Le départ de Maximilien Cochet en cours d'année a représenté un défi organisationnel. Ses tâches ont dû être redistribuées entre les membres restants, Maxime Gontran absorbant notamment une grande partie du travail de gameplay. La redondance R/S mise en place dès le début a heureusement permis d'amortir ce contretemps.

La compatibilité Python 3.14 / Ursina. La version de Python installée par défaut sur nos machines récentes provoquait des plantages (segmentation faults) avec Panda3D, le moteur sous-jacent d'Ursina. La solution a été de figer un environnement virtuel sous Python 3.11 pour tout le développement, les tests et le build.

Les chemins d'assets dans les exécutables. Le chargement des ressources dans les exécutables PyInstaller a demandé de nombreuses itérations : Ursina,

PyGame et OpenCV gèrent les chemins de fichiers différemment. PyInstaller extrait les fichiers dans un dossier temporaire, alors qu'Ursina cherchait ses assets dans le dossier de lancement. La solution finale combine un changement de répertoire de travail, la configuration explicite du dossier d'assets d'Ursina, et l'usage de chemins absolus pour PyGame et OpenCV.

Le build macOS. Les machines de build macOS de GitHub utilisent une architecture Apple Silicon, incompatible avec certaines bibliothèques de Panda3D. Il a fallu exclure du build un module inutilisé pour obtenir un exécutable fonctionnel.

Chacune de ces difficultés a été l'occasion d'approfondir notre compréhension de l'écosystème Python, du packaging et des subtilités multi-plateformes.

XX.Bilan individuel

Neev Chandiramani

Neev s'est investi dans les aspects réseau et multijoueur du projet, qui constituaient un 'élément technique exigeant. Il a conçu et développé l'architecture client-serveur TCP, le serveur dédié, le protocole d'échange JSON, le système de lobby et l'attribution automatique des rôles. Il a également mis en place la chaîne de build et de déploiement (CI/CD via GitHub Actions, PyInstaller, installateur Inno Setup) et hébergé le serveur sur un VPS dédié. Enfin, il a créé et géré le site web du projet et contribué au pôle graphisme en soutien. Il a également contribué à la gestion des équipes.

Gauthier Roland

Gauthier a joué un rôle central dans la direction artistique. Il a conçu les modèles 3D des personnages et de la map sous Blender, ainsi que les animations et le rig, contribuant fortement à l'identité visuelle particulière du jeu. Il a également participé au site web et secondé le pôle build / déploiement.

Maxime Gontran

Maxime Gontran s'est concentré sur le moteur de jeu : déplacements, physique (saut, gravité, collisions multi-points), système de points de vie, système d'attaque directionnelle et interface (HUD). À la suite du départ de Maximilien Cochet, il a absorbé une grande partie du travail de gameplay et a secondé plusieurs autres pôles.

Maxime Ye

Maxime Ye a apporté une contribution importante à l'ambiance sonore, élément clé de l'expérience d'horreur : recherche, sélection et intégration des musiques, des sons d'ambiance et des effets réactifs. Il a également pris en charge le menu Options, le rebinding des touches, la correction AZERTY/QWERTY, tout en participant aux aspects techniques du gameplay.

Mention. Maximilien Cochet a participé à la première phase du projet, notamment à la logique des évènements et du système d'interactions, avant son départ en cours d'année. Ses tâches ont ensuite été redistribuées comme décrit ci-dessus.

XXI.Conclusion et perspectives

Five Nights at Châtelet représente une aventure collaborative ambitieuse menée par notre équipe BambouX, composée de Neev Chandiramani, Gauthier Roland, Maxime Ye et Maxime Gontran. Chacun a apporté ses compétences techniques, sa créativité et son investissement personnel pour mener à bien un jeu d'horreur multijoueur immersif, aujourd'hui fonctionnel sur l'ensemble de ses aspects.

Par rapport à la deuxième soutenance, le chemin parcouru est considérable. Les trois contraintes obligatoires du sujet sont pleinement satisfaites : PyGame structure toute l'interface 2D et l'audio, l'Embuscadeur fournit une intelligence artificielle autonome réagissant au joueur, et le multijoueur en réseau fonctionne sur un serveur dédié distant. Au-delà, le jeu propose un système de rôles asymétrique complet, quatre énigmes coopératives, un système de combat, de prison et de libération, une ambiance sonore et visuelle soignée, et une distribution professionnelle sur trois plateformes.

Le projet nous a permis de mettre en pratique et de dépasser largement les connaissances acquises en cours : programmation Python avancée, réseau, gestion de projet en équipe, intégration continue, modélisation 3D et conception de gameplay. Il nous a aussi appris à composer avec l'imprévu, qu'il s'agisse du départ d'un membre ou des nombreuses difficultés techniques que nous avons rencontrés.

Perspectives

Plusieurs pistes permettraient d'enrichir encore le jeu au-delà du cadre scolaire :

- exploiter pleinement dans le moteur les animations de personnages déjà préparées dans Blender
- ajouter de nouvelles énigmes et varier les conditions de victoire pour augmenter la rejouabilité
- doter l'Embuscadeur d'un comportement de patrouille plus élaboré, voire d'une recherche de chemin (pathfinding)
- rendre fonctionnel l'exécutable sous mac-os.

- optimiser davantage les échanges réseau et ajouter une gestion de versions côté serveur pour fiabiliser les parties entre clients.

Five Nights at Châtelet est le fruit de huit mois de travail et marque une progression importante dans notre apprentissage. C'est avec fierté que nous présentons ce produit fini, packagé et complet, à l'occasion de cette soutenance finale.

XXII.Annexes

A. Commandes du jeu

Action	Touche par défaut
Avancer	Z
Reculer	S
Gauche	Q
Droite	D
Sauter	Espace
Sprint	Maj gauche
Interagir	E
Attaquer (infecté)	Clic gauche
Aide	H
Pause	Échap
Plein écran	F11

Commandes par défaut (reconfigurables depuis le menu Options)

B. Stack technique

Composant	Technologie
Langage	Python 3.11
Moteur 3D	Ursina Engine
Interface 2D / audio	PyGame
Réseau	Sockets TCP
Modélisation 3D	Blender (.obj / .mtl)
Packaging	PyInstaller + Inno Setup
Intégration continue	GitHub Actions (Windows / Linux / macOS)
Site web	HTML / CSS / JavaScript

Synthèse de la pile technologique du projet

C. Ressources externes

Les ressources externes mobilisées au cours du projet sont listées ci-dessous, regroupées par catégorie.

Documentation et références

- Documentation officielle Ursina Engine — https://www.ursinaengine.org/
- Documentation officielle PyGame — https://www.pygame.org/docs/
- Documentation Python (modules socket, threading, json) — https://docs.python.org/3/
- Documentation officielle Blender — https://docs.blender.org/
- Documentation PyInstaller — https://pyinstaller.org/
- Documentation Inno Setup — https://jrsoftware.org/isinfo.php
- Documentation GitHub Actions — https://docs.github.com/actions

Bibliothèques et outils

- **Ursina Engine** — moteur de rendu 3D temps réel (sur Panda3D).
- **PyGame** — menu principal, options, overlays 2D et mixeur audio.
- **Panda3D** — moteur 3D sous-jacent à Ursina.
- **Blender** — modélisation 3D des personnages et de la map (export .obj / .mtl).
- **socket / threading / json** — modules de la bibliothèque standard Python (réseau TCP, multithreading, sérialisation).
- **tomlkit** — dépendance Python complémentaire.
- **PyInstaller** — compilation du jeu en exécutable autonome.
- **Inno Setup** — création de l'installateur Windows.
- **GitHub Actions** — intégration et déploiement continus (build multi-OS).

Banques d'assets sonores et graphiques

- Freesound.org — banque d'effets sonores et de musiques d'ambiance libres.
- Images d'arrière-plan et textures personnalisées pour le menu et la station.

Cette liste complète, accompagnée des liens directs, est également maintenue à jour dans la section Ressources du site web du projet.

Ressources (dossier ressources/)

Le dossier des ressources regroupe l'ensemble des assets du jeu :

- 27 modèles 3D (.obj) et leurs matériaux (.mtl) : personnages (survivant, infecté, découpés en six pièces), map (Mall, Meubles), panneaux et affiches de la station.
- 21 fichiers audio : musique de menu, ambiance, battements de cœur, respiration de l'infecté, bruits de pas, sons d'attaque, de dégâts, de mort, d'interaction, chuchotements et son de gare.
- 15 images et textures : arrière-plans du menu, visuels des panneaux et affiches de la station.
- 8 assets de screamers : images plein écran et sons associés pour les jump-scares.

Documentation fournie avec le projet

- **README.md** - présentation, installation, contrôles et structure du projet.
- **BUILDING.md** - procédure complète de build et de release.
- **Manuel d'installation et manuel d'utilisation** - constituant le dossier d'exploitation (livrés au format PDF).