Kangaroo Planet

Rapport de soutenance 1

Projet réalisé par :

Helios Bringuet, Eliott Caquelot,
Nicolas Delisle, Tanguy de Jerphanion

EPITA - A3
Long Story Short

13 janvier 2025

Table des matières

Introduction

Dans le cadre de la réalisation de notre projet Kangaroo Planet, ce rapport de première soutenance constitue une étape importante. Il a pour objectif de présenter l'état d'avancement du projet, de détailler la répartition des tâches entre les membres de l'équipe et de revenir sur les progrès réalisés depuis le début. Ce document permet également d'identifier les difficultés rencontrées et de proposer des ajustements pour les prochaines phases.

Dès le départ, nous avons défini des objectifs clairs et une organisation adaptée pour structurer notre travail. Chaque membre de l'équipe a reçu des responsabilités en fonction de ses compétences et de ses préférences, dans le but d'assurer une progression efficace et équilibrée.

Dans ce rapport, nous traiterons trois points essentiels. Tout d'abord, nous reviendrons sur la répartition des tâches telle qu'elle a été planifiée dans le cahier des charges. Ensuite, nous analyserons les étapes accomplies en comparant les résultats obtenus avec les objectifs fixés. Enfin, nous évoquerons les obstacles rencontrés et les solutions apportées pour y faire face. Cette analyse nous permettra de mieux comprendre ce qui a bien fonctionné et ce qui peut encore être amélioré.

Ce rapport n'a pas seulement pour but de rendre compte de l'avancement du projet. Il vise également à démontrer notre engagement et à montrer que nous suivons une méthode de travail rigoureuse. Nous espérons que cette première soutenance sera l'occasion d'échanger des idées et de recueillir des suggestions pour améliorer notre approche.

Ainsi, ce document reflète notre travail collectif et notre volonté d'atteindre les objectifs fixés pour Kangaroo Planet. Il marque une première étape dans la réussite de ce projet, que nous continuerons à développer avec la même énergie et la même rigueur.

Chapitre 1

Cahier des charges : objectifs, tâches et technologies

1.1 Présentation du jeu

Kangaroo Planet est un jeu vidéo d'action et de survie où le joueur incarne un kangourou défendant sa planète d'une invasion humaine. L'objectif principal est de surmonter des vagues successives d'ennemis dans des niveaux à difficulté croissante. Le jeu propose deux modes distincts :

- Un **mode principal** style die and retry avec 20 à 30 niveaux prédéfinis, intégrant des dialogues et une progression scénaristique.
- Un **mode entraînement**, où les niveaux sont générés de manière procédurale pour offrir une expérience infinie et variée.

Le jeu s'appuie sur un univers visuel distinctif, avec des environnements qui évoquent une planète sphérique et des interactions dynamiques basées sur des mécaniques de jeu simples mais engageantes.

1.2 Objectifs du projet

Les principaux objectifs du projet sont les suivants :

- Offrir une expérience de jeu accessible et divertissante avec des mécaniques solides.
- Développer une stylisation unique, où le joueur incarne un kangourou dans un univers captivant.
- Mettre en place des niveaux variés et équilibrés, à la fois prédéfinis et générés aléatoirement.
- Garantir une performance optimale et une expérience utilisateur fluide.

1.3 Répartition des tâches

Tâches	Responsable	Suppléant
Mode entraînement	Nicolas Delisle	Eliott Caquelot
Mode principal	Helios Bringuet	Tanguy de Jerphanion
Modélisation du kangourou	Eliott Caquelot	Tanguy de Jerphanion
Modélisation des ennemis	Eliott Caquelot	Nicolas Delisle
Développement de l'arme	Tanguy de Jerphanion	Helios Bringuet
Développement IA	Tanguy de Jerphanion	Eliott Caquelot
Mécanique de tir	Helios Bringuet	Nicolas Delisle
Intégration audio	Nicolas Delisle	Eliott Caquelot
Design du menu	Tanguy de Jerphanion	Helios Bringuet
Multijoueur	Eliott Caquelot	Tanguy de Jerphanion
Système de dégâts	Helios Bringuet	Nicolas Delisle
Système de statistiques	Tanguy de Jerphanion	Eliott Caquelot

1.4 Technologies utilisées

Le développement de *Kangaroo Planet* repose sur les outils et technologies suivants :

- **Langage de programmation** : C#, utilisé pour coder les mécaniques du jeu.
- **Moteur de jeu** : Unity, pour la gestion des scènes, des objets et des interactions.
- **Gestion de projet** : Trello et Discord, pour organiser les tâches et suivre l'avancement du projet.

1.5 Prochaines étapes

Les prochaines étapes du projet incluent :

- Implémentation du multijoueur !
- La stylisation complète du jeu, avec des visuels qui reflètent une planète extraterrestre et une animation distinctive pour le personnage principal.
- La finalisation des niveaux prédéfinis du mode principal, intégrant des éléments narratifs.
- La mise en place et l'encadrement de la génération automatique des niveaux dans le mode entraînement.
- Les tests et ajustements pour garantir une expérience fluide et équilibrée.

Le projet est en plein développement, et chaque membre de l'équipe reste mobilisé pour atteindre les objectifs fixés tout en maintenant une bonne dynamique de groupe.

Chapitre 2

Développement : Semaines 1 et 2 (Nicolas)

2.1 Introduction

Durant les premières semaines du projet, l'équipe s'est concentrée sur la création des bases fonctionnelles du gameplay de **Kangaroo Planet**. Les scripts développés incluent les mécanismes essentiels tels que le contrôle du joueur, l'intelligence artificielle des ennemis, la gestion des armes et des projectiles, ainsi que le système de génération d'ennemis en vagues. Ces éléments constituent le socle sur lequel repose le reste du jeu.

2.2 Système de contrôle du joueur : *Player.cs*

Le script `Player.cs` gère les actions principales du joueur, notamment le déplacement, la visée et le tir.

2.2.1 Gestion des déplacements

Le joueur se déplace à l'aide des touches directionnelles du clavier. Le script récupère l'entrée utilisateur via `Input.GetAxisRaw("Horizontal")` et `Input.GetAxisRaw("Vertical")`, formant ainsi un vecteur de déplacement. Ce vecteur est normalisé et multiplié par une vitesse définie (`moveSpeed`) pour produire une vélocité constante. Cette vélocité est ensuite transmise au composant `PlayerController` chargé d'appliquer physiquement les mouvements au personnage.

2.2.2 Système de visée

La direction du regard est contrôlée par la position de la souris. Un rayon (*ray*) est projeté depuis la caméra en direction du curseur (`Input.mousePosition`), et la position où ce rayon intersecte un plan virtuel (sol) est calculée grâce à un `Plane`. Le joueur tourne pour faire face à cette position, ce qui est essentiel pour permettre des tirs précis dans la direction souhaitée.

2.2.3 Gestion des tirs

Lorsque le bouton gauche de la souris est enfoncé (`Input.GetMouseButton(0)`), le script appelle la méthode `Shoot()` du contrôleur d'armes (`GunController`).

Cela active l'arme équipée pour tirer un projectile dans la direction définie.

2.3 Intelligence artificielle des ennemis : *Enemy.cs*

Le script `Enemy.cs` contrôle les déplacements des ennemis et leur comportement de poursuite.

2.3.1 Suivi du joueur

Chaque ennemi utilise un agent de navigation (`NavMeshAgent`) pour se déplacer de manière réaliste. L'ennemi cible en permanence le joueur, dont la position est mise à jour régulièrement à l'aide d'une coroutine (`UpdatePath()`). Cette méthode calcule un chemin vers la position actuelle du joueur toutes les 0,25 secondes afin d'optimiser les performances tout en maintenant une réactivité acceptable.

2.3.2 Gestion de la mort

Le script hérite de la classe `LivingEntity`, ce qui permet de gérer les états vitaux, comme la mort. Lorsqu'un ennemi meurt, il notifie le système via l'événement `OnDeath()`, ce qui déclenche des actions comme réduire le nombre d'ennemis actifs.

2.4 Gestion des armes : *Gun.cs*

Le script `Gun.cs` est responsable des armes et de leurs fonctionnalités principales.

2.4.1 Tirs et projectiles

Lorsque la méthode `Shoot()` est appelée, un projectile est instancié à la position du canon (`muzzle`). Le script vérifie également le temps écoulé depuis le dernier tir (`nextShotTime`) afin de respecter une cadence de tir définie (`msBetweenShots`).

2.4.2 Vitesse et direction

Le projectile instancié est propulsé à une vitesse initiale (`muzzleVelocity`), définie dans le script. Cela garantit une trajectoire cohérente et permet au joueur de tirer précisément vers les cibles.

2.5 Système de vagues : *Spawner.cs*

Un système de vagues d'ennemis a été mis en place à l'aide du script `Spawner.cs`.

2.5.1 Génération des vagues

Les vagues sont définies à l'aide de la classe interne `Wave`, qui contient le nombre d'ennemis et le délai entre leurs apparitions (`timeBetweenSpawns`). Chaque vague commence par un appel à la méthode `NextWave()`, qui initialise les compteurs d'ennemis à générer (`enemiesRemainingToSpawn`) et à éliminer (`enemiesRemainingAlive`).

2.5.2 Gestion des ennemis

À intervalles réguliers, des ennemis sont instanciés et positionnés dans la scène. Chaque ennemi est lié à un événement `OnDeath()`, permettant au spawner de surveiller le nombre restant d'ennemis vivants et de lancer une nouvelle vague une fois la précédente terminée.

2.6 Interface de dégâts : *IDamageable.cs*

Le script `IDamageable.cs` définit une interface commune pour les entités pouvant recevoir des dégâts. Cette interface inclut la méthode `TakeHit()`, qui est appelée lorsque l'entité subit une attaque. Cela permet de généraliser la gestion des dégâts pour différents types d'entités (ennemis, joueurs, etc.).

2.7 Conclusion

Durant ces deux premières semaines, l'équipe a posé les bases du gameplay principal. Les systèmes implémentés permettent au joueur d'interagir avec l'environnement, de combattre des ennemis et de progresser à travers des vagues successives. Ces éléments fournissent une structure robuste pour intégrer des fonctionnalités plus avancées dans les phases suivantes.

Chapitre 3

Semaines 3 et 4 : Génération procédurale et algorithme Fisher-Yates Shuffle (Helios)

3.1 Génération procédurale de la carte

La génération de carte est une composante essentielle de notre projet, permettant la création dynamique et personnalisée des niveaux de jeu. Le script `MapGenerator.cs` est chargé de générer les tuiles, placer les obstacles, et définir les zones jouables pour les ennemis et le joueur. Cette approche garantit que chaque partie offre une expérience unique.

3.1.1 Structure générale du `MapGenerator.cs`

Le script s'articule autour des éléments suivants :

- **Tuiles (`tilePrefab`)** : Les tuiles sont générées à partir des dimensions spécifiées dans la classe `Map`. Chaque tuile est positionnée et dimensionnée en fonction de la grille.
- **Obstacles (`obstaclePrefab`)** : Les obstacles sont placés de manière semi-aléatoire tout en respectant des contraintes de jouabilité. Leur hauteur et couleur sont ajustées dynamiquement pour diversifier l'environnement.
- **NavMesh et masques** : Des masques sont ajoutés aux bordures de la carte pour restreindre la navigation des entités (joueur et ennemis) en dehors de la zone jouable.

3.1.2 Étapes principales de la génération

Voici les étapes détaillées de la génération :

1. **Initialisation des coordonnées** : Une liste de toutes les coordonnées possibles est générée en fonction des dimensions spécifiées (`allTileCoords`).
2. **Création de la carte** : Les tuiles sont instanciées sur une grille centrée autour de l'origine. Les tuiles sont légèrement réduites (`outlinePercent`) pour créer un espace visible entre elles.
3. **Placement des obstacles** : Les obstacles sont positionnés en utilisant une méthode de randomisation contrôlée. Chaque obstacle est évalué pour garantir que la carte reste entièrement accessible.

4. **Création des masques NavMesh** : Des masques sont instanciés autour de la carte pour restreindre la navigation au sein des limites spécifiées.

3.1.3 Contrôle de l'accessibilité

Un point critique est de s'assurer que toutes les tuiles accessibles (non occupées par des obstacles) forment une zone connectée. Pour cela, une recherche en largeur (**BFS**) est utilisée :

— L'algorithme parcourt les tuiles accessibles depuis le centre de la carte.
— Si toutes les tuiles non occupées peuvent être atteintes, la carte est validée.
— En cas d'échec, l'obstacle est retiré et une nouvelle position est testée.

3.1.4 Personnalisation des cartes

Chaque carte est définie par une instance de la classe `Map`, qui contient les paramètres suivants :

— **Dimensions** : Taille de la carte (nombre de tuiles en largeur et hauteur).
— **Pourcentage d'obstacles** : Contrôle la densité des obstacles.
— **Hauteur des obstacles** : Varie entre un minimum et un maximum.
— **Couleurs** : Les obstacles prennent une couleur qui varie progressivement pour offrir une esthétique agréable.

3.2 Algorithme Fisher-Yates Shuffle

L'algorithme Fisher-Yates est utilisé pour randomiser l'ordre des éléments dans une liste. Dans notre projet, il est appliqué pour mélanger les positions des obstacles (`shuffledTileCoords`) et des tuiles ouvertes (`shuffledOpenTileCoords`).

3.2.1 Description de l'algorithme

Fisher-Yates procède en itérant sur la liste d'éléments à mélanger. Pour chaque élément, un indice aléatoire est choisi parmi les indices restants, et les deux éléments sont échangés. Cela garantit un mélange uniforme et sans biais.

```
for i from n - 1 downto 1 do
    j ← random integer such that 0  j  i
    exchange a[i] and a[j]
```

3.2.2 Comparaison avec d'autres méthodes

— **Random.Sort()** : Cette méthode est parfois utilisée pour mélanger une liste en triant ses éléments en fonction de valeurs aléatoires. Ce-

pendant, elle est moins efficace (O(n log n)) et peut introduire des biais.

- **Avantages de Fisher-Yates** :
  - Complexité linéaire O(n).
  - Absence de biais dans le mélange.
  - Implémentation simple et directe.
- **Inconvénients** :
  - Nécessite un générateur de nombres aléatoires de haute qualité pour des résultats uniformes.

3.2.3 Utilisation dans le projet

Dans notre projet, Fisher-Yates est utilisé pour :

- Mélanger les coordonnées des obstacles afin de garantir une répartition aléatoire.
- Mélanger les tuiles ouvertes (`shuffledOpenTileCoords`) pour déterminer les points d'apparition des ennemis.

L'utilisation de Fisher-Yates garantit que les niveaux générés sont diversifiés, tout en respectant les contraintes de performance essentielles dans un jeu en temps réel.

Chapitre 4

Semaines 5 et 6 : Approfondissement des mécaniques cruciales de Gun.cs, Enemy.cs et Spawner.cs (Tanguy)

Durant ces deux semaines, l'équipe s'est concentrée sur la reprise et l'amélioration des scripts principaux du projet, à savoir `Enemy.cs`, `Gun.cs` et `Spawner.cs`. Ces scripts, initialement implémentés lors des semaines 1 et 2, ont été revisités afin d'atteindre un niveau de perfection technique adapté aux besoins du projet. Ci-dessous, nous détaillons les modifications spécifiques apportées à chaque script.

4.1 Reprise et amélioration de `Enemy.cs`

Le script `Enemy.cs` est essentiel pour gérer le comportement des ennemis, incluant leurs déplacements, interactions avec le joueur, et réactions à divers événements (attaques, dégâts, etc.). Les modifications principales apportées cette semaine sont les suivantes :

Gestion des états

Le système de gestion des états (`Idle`, `Chasing`, `Attacking`) a été restructuré pour améliorer son efficacité. Par exemple, la logique dans la méthode `Update()` a été optimisée pour éviter des calculs inutiles.

```
void Update() {
    if (hasTarget) {
        if (Time.time > nextAttackTime) {
            float sqrDstToTarget = (target.position - transform.position).sqrMagnitude;
            if (sqrDstToTarget < Mathf.Pow(attackDistanceThreshold
                + myCollisionRadius + targetCollisionRadius, 2)) {
                nextAttackTime = Time.time + timeBetweenAttacks;
                StartCoroutine(Attack());
            }
        }
    }
}
```

Ce changement garantit que les calculs de distance sont uniquement effectués lorsque le temps de recharge de l'attaque est écoulé.

Attaques synchronisées

Le système d'attaque a été entièrement révisé pour synchroniser visuellement et mécaniquement l'animation de l'ennemi avec les dégâts infligés.

```
IEnumerator Attack() {
    bool hasAppliedDamage = false;

    while (percent <= 1) {
        if (percent >= 0.5f && !hasAppliedDamage) {
            hasAppliedDamage = true;
            targetEntity.TakeDamage(damage);
        }
        percent += Time.deltaTime * attackSpeed;
        yield return null;
    }
}
```

Paramètres dynamiques

La méthode `SetCharacteristics()` a été enrichie pour rendre les ennemis totalement personnalisables. En plus de la vitesse et de la couleur, le script calcule désormais automatiquement les dégâts en fonction de la santé initiale du joueur.

```
public void SetCharacteristics(float moveSpeed, int hitsToKillPlayer,
                               float enemyHealth, Color skinColour) {
    pathfinder.speed = moveSpeed;
    damage = Mathf.Ceil(targetEntity.startingHealth / hitsToKillPlayer);
    startingHealth = enemyHealth;
    skinMaterial.color = skinColour;
}
```

Ces changements assurent une gestion plus fluide et réaliste des ennemis, tout en améliorant la lisibilité du code.

4.2 Reprise et amélioration de `Gun.cs`

Le script `Gun.cs`, qui régit la mécanique de tir dans le jeu, a également subi des modifications importantes.

Précision des tirs

La gestion de la trajectoire des projectiles a été ajustée pour mieux prendre en compte l'orientation du joueur.

```
Vector3 shootDirection = (targetPoint - gunBarrelEnd.position).normalized;
Projectile newProjectile = Instantiate(projectile,
                                       gunBarrelEnd.position, Quaternion.identity);
newProjectile.SetSpeed(projectileSpeed, shootDirection);
```

Optimisation des impacts

La gestion des effets visuels et sonores à l'impact a été revue pour éviter les doublons et les appels excessifs.

```
if (Physics.Raycast(ray, out hit, range)) {
    AudioManager.instance.PlaySound("Gunshot", transform.position);
    Instantiate(hitEffect, hit.point, Quaternion.LookRotation(hit.normal));
}
```

Ces ajustements rendent les mécaniques de tir plus intuitives pour le joueur, tout en optimisant les performances globales.

4.3 Reprise et amélioration de `Spawner.cs`

Le script `Spawner.cs` a été retravaillé pour améliorer la gestion des vagues d'ennemis.

Anti-camping

Le système détecte maintenant si le joueur reste trop longtemps au même endroit (`isCamping`) et ajuste la position des ennemis en conséquence.

```
if (Vector3.Distance(playerT.position, campPositionOld) < campThresholdDistance) {
    isCamping = true;
    spawnTile = map.GetTileFromPosition(playerT.position);
} else {
    isCamping = false;
    spawnTile = map.GetRandomOpenTile();
}
campPositionOld = playerT.position;
```

Transitions de vagues fluides

La méthode `NextWave()` a été améliorée pour intégrer un signal sonore et un effet visuel indiquant la fin d'une vague.

```
if (currentWaveNumber > 0) {
    AudioManager.instance.PlaySound2D("Level Complete");
}
```

4.4 Conclusion sur le travail des semaines 5 et 6

Ces améliorations ont permis de renforcer la structure des scripts principaux du projet, rendant le code plus performant, modulaire, et extensible. Le jeu bénéficie désormais de comportements ennemis plus réactifs, d'une mécanique de tir précise, et d'une gestion des vagues d'ennemis plus dynamique. Ces évolutions constituent une étape importante dans le développement du projet, posant des bases solides pour les fonctionnalités à venir.

Chapitre 5

Semaines 7 et 8 : Effets visuels et audio (Eliott)

5.1 Gestion audio

Les semaines 7 et 8 ont été consacrées à l'intégration et à la gestion des effets visuels et audio, afin de rendre l'expérience de jeu plus immersive et dynamique. La gestion du son joue un rôle crucial dans l'ambiance du jeu, et les effets visuels permettent de renforcer le réalisme du gameplay.

5.1.1 Le script `AudioManager.cs`

Le `AudioManager` gère l'ensemble des sons du jeu, en permettant de moduler le volume de la musique, des effets sonores et du volume global via un système de curseurs.

- **Gestion des canaux audio :** Trois canaux sont gérés par le `AudioManager` : le *Master*, les effets sonores (*SFX*) et la musique. Chaque canal dispose de son propre réglage de volume.
- **Lecture de la musique :** La musique change dynamiquement en fonction de la scène active. Lors du changement de scène, un fondu enchaîné est appliqué pour rendre le passage d'une musique à une autre plus fluide.
- **Effets sonores :** Les effets sonores sont joués en fonction des actions du joueur. Les sons peuvent être spatialisés (en 3D) ou joués en stéréo, selon le type d'effet.

```
void PlayMusic(AudioClip clip) {
    musicSource.clip = clip;
    musicSource.Play();
    StartCoroutine(FadeInMusic());
}
```

5.1.2 Le script `SoundLibrary.cs`

Le `SoundLibrary` gère une collection de sons classés par groupes. Cela permet de jouer des sons aléatoires à partir de chaque groupe lorsqu'une action spécifique se produit dans le jeu.

- **Accès aux sons par groupe :** Lorsqu'un événement dans le jeu nécessite un son (tir, explosion, etc.), le `SoundLibrary` choisit aléatoirement un son dans le groupe approprié.

— **Gestion de la spatialisation :** Selon la position du joueur, les effets sonores peuvent être joués en 3D pour simuler la distance et la direction.

```
AudioClip GetRandomSound(string groupName) {
    List<AudioClip> soundList = soundGroups[groupName];
    return soundList[Random.Range(0, soundList.Count)];
}
```

5.2 Effets visuels

Les effets visuels jouent un rôle fondamental dans l'amélioration de l'immersion du joueur. En plus des effets audio, plusieurs effets visuels ont été intégrés, comme les flashes de la bouche du pistolet lors du tir et les douilles qui tombent au sol.

5.2.1 Le script `MuzzleFlash.cs`

Ce script gère l'affichage du flash qui se produit à la bouche du pistolet lors du tir. L'effet de flash donne une sensation de puissance au tir tout en améliorant le réalisme du gameplay.

— **Activation du flash :** Chaque fois que le joueur tire, un flash est généré et affiché brièvement. Ce flash provient d'un ensemble de sprites, choisis de manière aléatoire.

— **Durée du flash :** Le flash reste visible pendant un court moment, avant de disparaître pour ne pas perturber le gameplay.

```
void CreateFlash() {
    flash.SetActive(true);
    StartCoroutine(FlashDuration());
}

IEnumerator FlashDuration() {
    yield return new WaitForSeconds(0.1f);
    flash.SetActive(false);
}
```

5.2.2 Le script `Shell.cs`

Le `Shell` gère l'apparition des douilles qui tombent au sol après chaque tir. Ce mécanisme ajoute un effet visuel réaliste en simulant la chute des douilles avec des comportements physiques.

— **Spawn des douilles :** Lorsqu'un tir est effectué, une douille est instanciée et projetée dans une direction aléatoire, en fonction de la position et de la rotation du joueur.

— **Disparition des douilles :** Les douilles disparaissent après un court délai ou lorsqu'elles touchent le sol, créant un effet visuel de disparition fluide.

```
void SpawnShell() {
    GameObject shell = Instantiate(shellPrefab, spawnPosition, Quaternion.identity);
    Rigidbody rb = shell.GetComponent<Rigidbody>();
    rb.AddForce(Random.insideUnitSphere * 5f, ForceMode.Impulse);
}
```

5.3 Gestion de la musique et des sons dans les scènes

5.3.1 Le script `MusicManager.cs`

Le `MusicManager` contrôle la musique en fonction des scènes. Il permet un fondu fluide entre les morceaux de musique pour éviter un changement abrupt.

— **Transition musicale :** Lors du changement de scène (menu, jeu, etc.), la musique change de manière fluide grâce à un fondu. Ce fondu est effectué en diminuant progressivement le volume de l'ancienne musique avant d'introduire la nouvelle.

— **Gestion des ambiances sonores :** Selon la scène, des ambiances musicales distinctes sont jouées pour renforcer l'atmosphère du jeu.

```
void FadeOutMusic() {
    StartCoroutine(FadeVolume(0f, fadeDuration));
}

IEnumerator FadeVolume(float targetVolume, float duration) {
    float startVolume = musicSource.volume;
    float time = 0f;
    while (time < duration) {
        musicSource.volume = Mathf.Lerp(startVolume, targetVolume, time / duration);
        time += Time.deltaTime;
        yield return null;
    }
}
```

5.4 Conclusion

Les semaines 7 et 8 ont permis d'ajouter des effets visuels et sonores qui améliorent considérablement l'expérience de jeu. La gestion du son via `AudioManager` et `SoundLibrary` a permis d'optimiser l'audio en l'adaptant aux besoins du jeu. Les effets visuels, comme les flashes de bouche et les douilles, renforcent le réalisme et l'immersion. Ces ajouts rendent le gameplay plus dynamique, offrant une expérience plus engageante pour le joueur.

Chapitre 6

Organisation du travail et dynamique d'équipe

Le développement de ce projet s'est déroulé sur plusieurs semaines, durant lesquelles nous avons travaillé en étroite collaboration en tant qu'équipe. Voici un aperçu de notre organisation et de la dynamique qui nous a permis d'avancer, malgré quelques défis initiaux.

6.1 Réunions hebdomadaires

Afin de maintenir une coordination optimale et d'assurer un suivi régulier de l'avancement du projet, nous avons décidé d'organiser une réunion hebdomadaire. Ces réunions se tenaient chaque semaine à la cafétéria de l'EPITA, un endroit que nous avons choisi pour sa convivialité et son accessibilité.

Lors de ces réunions, nous discutions des tâches en cours, des éventuels problèmes rencontrés et des objectifs pour la semaine suivante. Chaque membre de l'équipe présentait ses avancées, et nous débattions ensemble des priorités à établir. Cette méthode de travail nous a permis de rester alignés et de progresser de manière structurée.

6.1.1 Planification des tâches

Pour structurer le travail, nous avons découpé le projet en plusieurs étapes correspondant aux différents systèmes à développer (gestion des ennemis, effets visuels, audio, etc.). Chacun se voyait attribuer des responsabilités spécifiques, tout en restant flexible pour s'entraider en cas de besoin.

- **Chaque réunion débutait par un récapitulatif des objectifs de la semaine précédente,** afin de vérifier si les tâches assignées avaient été accomplies.
- **Une partie de la réunion était dédiée à l'identification des obstacles,** qu'ils soient techniques ou organisationnels.
- Enfin, nous définissions **les priorités pour la semaine à venir** et repartagions les tâches en conséquence.

6.2 Les défis de l'équipe

Bien que notre organisation ait été efficace dans l'ensemble, nous avons rencontré quelques difficultés, notamment au début du projet. Une dispute inattendue a perturbé notre dynamique initiale : l'un des membres de l'équipe, sur un sujet aussi improbable que les kangourous, a décidé de quitter le

groupe. Ce départ, survenu à la suite d'une divergence d'opinions, a réduit notre effectif de cinq à quatre membres.

Ce moment a été un défi pour l'équipe, car il nous a forcés à redistribuer les tâches du membre manquant et à revoir notre organisation. Toutefois, cet épisode nous a également soudés en tant qu'équipe, en renforçant notre capacité à collaborer et à résoudre les conflits de manière constructive.

6.3 Un effort commun et une expérience enrichissante

Malgré les obstacles rencontrés, travailler sur ce projet a été une expérience globalement positive. Chaque membre de l'équipe a apporté ses compétences et sa créativité, contribuant à faire évoluer le projet à chaque étape. Les réunions hebdomadaires à la cafétéria de l'EPITA sont rapidement devenues plus qu'un simple moment de travail : elles nous ont permis de partager des idées, de débattre des choix techniques, et même de rire autour d'une discussion sur les ennemis les plus improbables qu'on pourrait ajouter dans le jeu.

Au final, ce projet n'a pas seulement été un exercice technique, mais aussi une véritable aventure humaine. Travailler ensemble, surmonter les défis, et voir notre vision prendre vie a été une source de satisfaction pour chacun d'entre nous. Même si tout n'a pas toujours été facile, le plaisir de collaborer et de créer ensemble restera le souvenir le plus précieux de cette expérience.

Chapitre 7

Perspectives et travaux restants

Bien que le projet soit déjà bien avancé, plusieurs aspects restent à développer pour atteindre la vision finale que nous avons pour le jeu. Ces tâches, principalement liées à la stylisation, à l'extension des fonctionnalités, et à l'intégration de nouveaux contenus, représentent les prochaines étapes essentielles.

7.1 Stylisation du jeu

L'un des objectifs les plus évidents est d'apporter une identité visuelle forte au jeu. Pour l'instant, les graphismes sont fonctionnels, mais il est nécessaire de travailler sur le style pour offrir une expérience immersive et mémorable.

7.1.1 Incarner un kangourou

Le joueur doit incarner un personnage distinctif : un kangourou. Cela implique la création d'un modèle 3D ou d'un sprite représentant le kangourou, ainsi que l'ajout d'animations (saut, attaque, mort, etc.) pour rendre les actions du joueur visuellement dynamiques et cohérentes.

7.1.2 Un univers spatial

L'univers du jeu doit donner l'impression de se dérouler dans l'espace. Pour cela, le décor devra être modifié afin de représenter une planète sphérique. Bien que les mécaniques de gameplay actuelles ne changent pas, l'ajout d'une sphère comme base visuelle peut simuler une immersion dans cet univers. Des éléments comme des étoiles, des galaxies en arrière-plan, et des paysages extraterrestres devront être ajoutés pour renforcer ce thème.

7.2 Modes de jeu

Deux modes principaux restent à finaliser pour offrir une expérience de jeu complète : le *mode principal* et le *mode entraînement*.

7.2.1 Mode principal

Le *mode principal* consistera en une série de 20 à 30 niveaux prédéfinis, chacun avec des défis uniques et une progression narrative. Cela nécessite :

— La conception de chaque niveau avec des paramètres spécifiques (nombre d'ennemis, difficulté, objectifs).
— L'ajout de dialogues pour donner du contexte et enrichir l'histoire.
— Une gestion claire de la progression, avec des niveaux déverrouillables et une montée en difficulté bien dosée.

Ce mode vise à fournir une expérience structurée et engageante pour le joueur, avec une fin définie.

7.2.2 Mode entraînement

Le *mode entraînement* permettra au joueur de s'entraîner dans des niveaux générés de manière procédurale. Les objectifs sont de créer une génération automatique des niveaux et d'encadrer cette mécanique pour garantir une expérience équilibrée. Les étapes pour ce mode incluent :

— La mise en place d'un algorithme de génération procédurale pour créer des niveaux variés à l'infini.
— La définition de règles pour éviter des configurations impossibles ou trop faciles.
— L'intégration d'un système de score ou de temps, afin de motiver les joueurs à s'améliorer dans ce mode.

Ce mode offrira une rejouabilité infinie, permettant aux joueurs de s'entraîner et de relever des défis variés.

7.3 Résumé des travaux restants

En résumé, les prochaines étapes pour compléter le projet comprennent :

— **Styliser le jeu** : Donner une identité visuelle forte en incarnant un kangourou et en ajoutant un décor spatial immersif.
— **Finaliser le mode principal** : Concevoir une série de niveaux prédéfinis avec une progression narrative.
— **Mettre en place le mode entraînement** : Créer une génération procédurale de niveaux infinis et équilibrés.
— **MULTIJOUEUR**

Ces tâches représentent une partie ambitieuse mais passionnante du développement. Leur réalisation permettra de transformer ce projet en une expérience complète, immersive et mémorable, à la hauteur de notre vision initiale.

Chapitre 8

Aperçu du jeu

FIGURE 8.1 – Capture d'écran du menu principal.

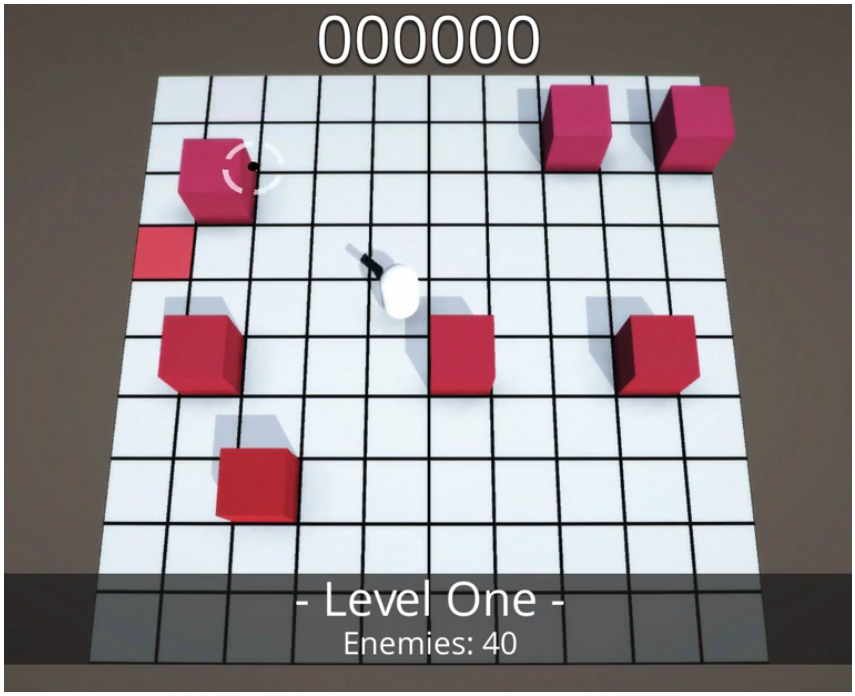

FIGURE 8.2 – Capture d'écran du gameplay.

Chapitre 9

Conclusion

À ce stade du projet, nous avons déjà accompli des avancées significatives, mais il reste encore un long chemin à parcourir. Les mécaniques de base du jeu sont en place, et nous avons posé des fondations solides pour les systèmes de gameplay principaux. Cependant, de nombreuses fonctionnalités et améliorations restent à implémenter pour atteindre notre vision finale.

Notre équipe a su surmonter des défis complexes, qu'ils soient techniques, comme l'équilibrage des vagues d'ennemis ou la génération procédurale, ou organisationnels, avec l'ajustement de nos méthodes de travail en équipe. Chaque étape nous a permis de mieux cerner les exigences du projet et de renforcer notre collaboration.

Les prochaines phases seront particulièrement excitantes : elles incluent la stylisation complète du jeu, la mise en place des deux modes (campagne et entraînement), et l'ajout de nouveaux éléments narratifs et visuels. Ces objectifs représentent à la fois des défis ambitieux et des opportunités de donner vie à notre concept.

Ce projet est encore en pleine évolution, et nous sommes impatients de poursuivre son développement. La suite s'annonce exigeante mais prometteuse, et nous sommes motivés par l'idée de créer un jeu qui allie créativité, technique et plaisir de jouer.