PROJET HACKATHON G15'

RAPPORT TECHNIQUE

HEMERYCK QUENTIN - DECAMPS MAX

HEH BA1P INFORMATIQUE
2023 - 2024

Table des matières

1. Introduction

Ce rapport technique détaille les aspects de conception et de développement de notre jeu vidéo créé lors du hackathon. Notre jeu est inspiré de « Hadès », un Roguelike où le héros principal doit vaincre des ennemis présents dans une salle pour pouvoir débloquer la salle suivante. Au bout d'un certain nombre de salles passées, vient la salle du boss.

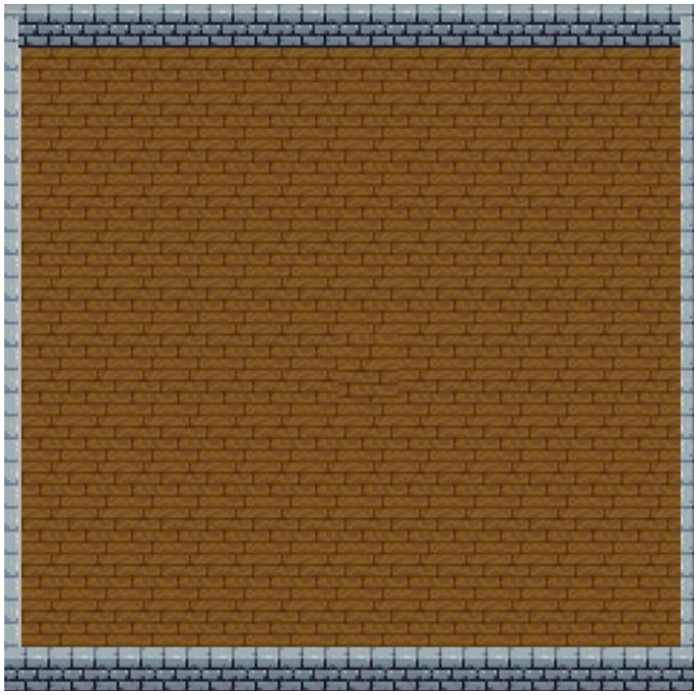
Notre jeu à pour particularités un système de combat dynamique, sa personnalisation étendue et les différentes armes dont possède notre personnage principal. À travers ce rapport

2. Mécanismes du jeu

2.1 Carte de jeu

Nous avons choisi d'utiliser un background pour notre jeu vidéo. Les dimensions sont 679x676 et nous avons choisi de laisser l'écran de jeu de cette dimension dans le but de faire penser à un jeu rétro. En effet, mettre un mode « fullscreen » aurait à notre goût gâché le côté « rétro » de notre jeu.

La carte en question :



2.1.1 Collisions de la carte :

Les collisions étaient un aspect essentiel pour notre gameplay étant donné, qu'il y a des murs aux extrémités afin de respecter un certain sens à notre gameplay.

Pour expliquer, chaque objet (player, enemy…) mobiles ont un carré de collision associé et est mis à jour à chaque fois que l'objet se déplace. Lorsque le carré se déplace, le jeu vérifié le déplacement en s'assurant que la nouvelle position ne dépasse pas les limites de la carte instaurée. (cela est vérifié grâce à la méthode Check_oob(Out-of-bounds)).

Player :

```
def check_oob(x, y):
    limit_x = [18, 629]
    limit_y = [2, 557]
    if x < limit_x[0] or x > limit_x[1]:
        return False
    if y < limit_y[0] or y > limit_y[1]:
        return False
    return True

moved = False
if keys[pygame.K_q] and check_oob(self.__rect.x - self.__speed, self.__rect.y):
    self.__rect.x -= self.__speed
    self.__current_direction = 'left'
    moved = True
if keys[pygame.K_d] and check_oob(self.__rect.x + self.__speed, self.__rect.y):
    self.__rect.x += self.__speed
    self.__current_direction = 'right'
    moved = True
if keys[pygame.K_z] and check_oob(self.__rect.x, self.__rect.y - self.__speed):
    self.__rect.y -= self.__speed
    self.__current_direction = 'up'
    moved = True
if keys[pygame.K_s] and check_oob(self.__rect.x, self.__rect.y + self.__speed):
    self.__rect.y += self.__speed
    self.__current_direction = 'down'
    moved = True

if moved:
    self.__current_image = pygame.transform.scale(self.__images[self.__current_direction],
                                                  size: (self.__rect.width, self.__rect.height))
    self.__rect = self.__current_image.get_rect(center=self.__rect.center)
    if self.__current_weapon == 'bow' or self.__current_weapon == 'hammer':
        self.__weapon_image = self.__weapons[self.__current_weapon][self.__current_direction]
        self.__weapon_rect = self.__weapon_image.get_rect()
```

Ennemis :

```
def check_oob(self, x, y):
    """
    Vérifie si une position est hors des limites de l'écran.
    """
    return not (x < 0 or x + self._new_size[0] > self._screen_width or y < 0 or y + self._new_size[1] > self._screen_height)
```

2.2 Attaques du joueur

Nous avons choisi d'instaurer un système d'attaque du joueur vers les ennemis avec deux types d'armes.

Un marteau :

En effet, lorsque la méthode « SwingHammer() » est utilisée, cela vérifie les collisions entre le marteau et les ennemis, et si une collision est détectée, des dégâts sont infligés à notre ennemi.

```
def swingHammer(self, enemies):
    """
    Frappe avec le marteau et vérifie les collisions avec les ennemis.
    """
    if self.__current_weapon == "hammer":
        self.__weapon_image = self.__weapons['hammer_attack'][self.__current_direction]
        self.__weapon_rect = self.__weapon_image.get_rect(center=self.__rect.center)
        hammer_damage = 25
        for enemy in enemies:
            if self.__weapon_rect.colliderect(enemy.rect):
                enemy.applyDamage(hammer_damage)
                print(f"Hit enemy with hammer at position: ({enemy.rect.x}, {enemy.rect.y})")
        self.__weapon_image = self.__weapons[self.__current_weapon][self.__current_direction]
```

Un arc :

En effet, lorsque la méthode « Shootarrow() » est appelée, cela crée une nouvelle flèche à l'ajoute à la liste des flèches du joueur.

```
def shootArrow(self):
    """
    Tire une flèche dans la direction actuelle du joueur.
    """
    if self.__current_weapon == "bow":
        direction = self.__current_direction
        arrow_image_paths = {
            'right': "assets/weapon_arrow.png",
            'left': "assets/reverse_weapon_arrow.png",
            'up': "assets/up_weapon_arrow.png",
            'down': "assets/down_weapon_arrow.png"
        }
        if direction == 'right':
            new_arrow = Arrow(self.__rect.right, self.__rect.centery, direction, arrow_image_paths[direction])
        elif direction == 'left':
            new_arrow = Arrow(self.__rect.left, self.__rect.centery, direction, arrow_image_paths[direction])
        elif direction == 'up':
            new_arrow = Arrow(self.__rect.centerx, self.__rect.top, direction, arrow_image_paths[direction])
        elif direction == 'down':
            new_arrow = Arrow(self.__rect.centerx, self.__rect.bottom, direction, arrow_image_paths[direction])
        self.__arrows.append(new_arrow)
        print(f"arrow shot from position: ({new_arrow.rect.x}, {new_arrow.rect.y})")
```

Ensuite, les flèches sont mises à jour et dessinées pour chaque frames. Si une flèche atteint sa distance maximale(250 pixels) ou sort des dimensions de l'écran, elle est retirée de la liste.

```
def updateArrows(self, screen):
    """
    Met à jour et dessine les flèches du joueur.
    """
    for arrow in self.__arrows[:]:
        arrow.draw(screen)
        if not arrow.update():
            self.__arrows.remove(arrow)
```

La classe Arrow gère le comportement des flèches et initialise la position, la direction, la vitesse de la flèche et sa position à chaque frames.

```
class Arrow:
    """
    Classe représentant une flèche tirée par le joueur.
    """
    def __init__(self, x, y, direction, image_path, speed=1.9):
        self.__original_image = pygame.image.load(image_path)
        self.__new_width = int(self.__original_image.get_width() * 1)
        self.__new_height = int(self.__original_image.get_height() * 1)

        self.__image = pygame.transform.scale(self.__original_image, size: (self.__new_width, self.__new_height))
        self.__rect = self.__image.get_rect()
        self.__rect.x = x
        self.__rect.y = y
        self.__direction = direction
        self.__speed = speed
        self.__distance = 0
        self.__max_distance = 250
```

2.3 Spawn Random d'ennemis sur la map

Pour ajouter une côté imprévisible à notre jeu, nous avons choisi d'implémenter un spawn random des entités ennemies sur notre map.
La méthode « random_position(self) » génère des coordonnées aléatoires pour positionner l'ennemi sur la carte ens 'assurant que l'ennemi n'apparaissent pas hors des dimensions de notre map.

```
1 usage
def random_position(self):
    """
    Génère une position aléatoire pour l'ennemi sur l'écran.
    """
    return random.randint( a: 0, self._screen_width - self._new_size[0]), random.randint( a: 0, self._screen_height - self._new_size[1])
```

2.3.1 Déplacement des ennemis vers le joueur

Nous avons implémenté la méthode « move_towards_player(self, player_x, player_y) » qui permet à l'ennemi de se diriger vers la position du joueur en calculant la direction et en ajustant les coordonnées de l'ennemi.

```
def move_towards_player(self, player_x, player_y):
    """
    Déplace l'ennemi vers la position du joueur.
    """
    dx, dy = player_x - self._x, player_y - self._y
    distance = (dx ** 2 + dy ** 2) ** 0.5
    if distance != 0:
        step_x = dx / distance * self._speed
        step_y = dy / distance * self._speed
        if self.check_oob(self._x + step_x, self._y + step_y):
            self._x += step_x
            self._y += step_y
```

2.4 Système de sauvegarde

Nous avons choisi d'implémenter un système de sauvegarde suite aux exigences du cahier des charges. En effet, nous avons implémentés un système qui permet de sauvegarder l'état actuel du jeu dans un fichier JSON. Cela inclut la position du joueur, sa santé, son arme actuelle et l'état des ennemis.

```
def save_game(player, enemies, wave):
    """
    Sauvegarde l'état actuel du jeu dans un fichier JSON.
    """
    game_state = {
        'player': {
            'x': player.rect.x,
            'y': player.rect.y,
            'health': player.health,
            'current_weapon': player.current_weapon
        },
        'enemies': [{
            'type': type(enemy).__name__,
            'x': enemy.rect.x,
            'y': enemy.rect.y,
            'health': enemy.mobHealth
        } for enemy in enemies],
        'wave': wave
    }
    with open('save/savegame.json', 'w') as save_file:
        json.dump(game_state, save_file)
```

Exemple du fichier JSON :


```
{"player": {"x": 42, "y": 284, "health": 30, "current_weapon": "bow"}, "enemies": [{"type": "Boss", "x": 196, "y": 415,
  "health": 20}], "wave": 2}
```


2.5 Menu Principal et Menu de pause

Nous avons créé un menu principal et un menu de pause afin de permettre au joueur de naviguer entre les différentes options du jeu. Ces menus sont intégrés avec une interface utilisateur intuitive et réactive.

Menu Principal

Menu Pause :

2.6 Gestion de la santé du joueur

La santé du joueur est représentée dans le coin supérieur gauche de l'écran par des cœurs. La méthode « drawHealth » dessine les cœurs de la santé du joueur. Et se vide en fonction des dégâts subis par les ennemis.

```python
def drawHealth(screen, health):
    """
    Dessine les cœurs représentant la santé du joueur à l'écran.
    """
    heart_x_start = 10
    heart_y = 10
    hearts_full = health // 10
    hearts_half = (health % 10) // 5

    for i in range(min(hearts_full, 3)):
        screen.blit(heartFull, (heart_x_start + 32 * i, heart_y))

    if hearts_half == 1 and hearts_full < 3:
        screen.blit(heartHalf, (heart_x_start + 32 * hearts_full, heart_y))

    for i in range(hearts_full + hearts_half, 3):
        screen.blit(heartEmpty, (heart_x_start + 32 * i, heart_y))
```

2.7 Système de passage vers la prochaine vague

Notre jeu utilise un système de vagues pour augmenter progressivement la difficulté. La méthode « load_next_wave() » charge la prochaine vague d'ennemis à chaque fois que tous les ennemis de la vague actuelle sont vaincus.

```python
def load_next_wave(enemies, wave):
    """
    Charge la prochaine vague d'ennemis.
    """
    new_enemies = create_enemies(len(enemies) * 2, speed: 0.4, width, height, Lizard) + create_enemies(len(enemies) * 2, speed: 0.4,
                                                                                                       width, height, Orc)
    if wave == 2:
        new_enemies.append(Boss(speed=0.3, screen_width=width, screen_height=height))
    return new_enemies
```

3. Choix de conceptions

3.1 Design retro

Nous avons fait le choix d'un design retro pour essayer de recréer l'esthétique des anciens jeux « Zelda », ce choix influence le style artistique et les animations utilisées dans ce jeu.

3.2 Gestion des armes

Le système de gestion des armes permet au joueur de basculer entre différentes armes, en fonction de la situation de combat. Cela ajoute de la profondeur stratégique au gameplay.

3.3 Système de téléportation

Le système de téléportation permet aux joueurs de progresser d'une salle vers une autre après avoir vaincu la vague d'ennemis. Cela ajoute une structure au jeu et donne un objectif aux joueurs.

Le joueur ne peut pas utiliser la plaque si les ennemis ne sont pas vaincus, de + des effets sonores sont présent à l'activation de la plaque pour donner un feedback immédiat au joueur.

3.4 Audio et musique d'ambiance

L'audio et la musique d'ambiance sont essentiels pour créer une atmosphère de jeu et pour améliorer l'expérience de jeu.

Sounds effects et Musiques :

Main menu + Menu pause : Game is on – Tom Deneyer

Fight theme : Demon's Souls soundtrack

Teleport activation: Mario Warp Pipe Sound Effect

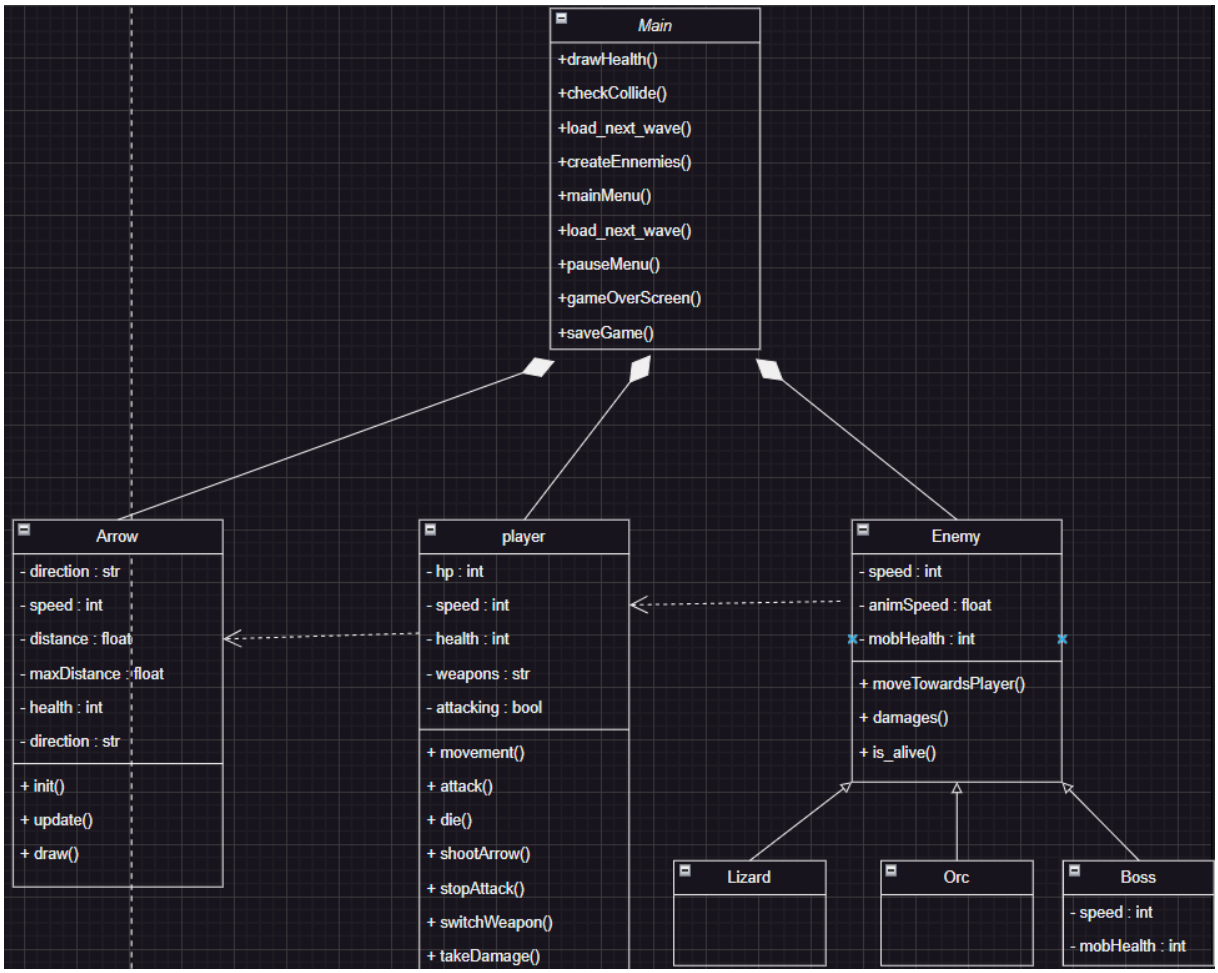
Upgrade activation: Super Mario Bross Power Up Sound Effect

4. Retour sur expérience et Méthodologie de travail :

4.1 Méthodologie de Travail Intensif

Description : Nous avons adopté une méthodologie de travail intensif pour gérer le projet de développement de notre jeu. Contrairement aux méthodes de travail existantes, notre approche se concentrait sur des périodes de travail prolongées avec des objectifs finaux. En raison de la réduction de notre effectif à deux personnes, nous avons dû être particulièrement déterminés et productifs pour produire un jeu uniquement fonctionnel comparable à aux autres groupes.

Raisons du choix : A contrario de notre choix, la méthodologie de travail intensif nous a permis de nous concentrer uniquement sur les tâches essentiel et importante au bon fonctionnement du produit finale. Cette approche a offert une meilleure vision d'ensemble du projet et a permis de s'assurer que chaque partie était bien intégrée avant de passer à la suivante. Cette méthode était particulièrement adaptée à notre situation, où une planification détaillée et une exécution rigoureuse étaient nécessaires pour compenser le manque de ressources humaines.



Brève explication de notre UML :

Le diagramme illustre une structure où la classe Main agit comme un contrôleur central, gérant les principales dynamiques et interactions entre le Player, Enemy, et Arrow. Ces interactions facilitent non seulement le combat et les déplacements mais aussi les transitions entre différents états du jeu.

4.2 Implémentation

- Phases de Travail : Le projet était divisé en phases de travail distinctes, chacune se concentrant sur un aspect majeur du développement, comme la conception et le développement des fonctionnalités principales.
- Réunions de Bilan : Des réunions de bilan étaient tenues chaque matinée et fin de journée afin d'évaluer les progrès, identifier les problèmes, et ajuster les plans pour les phases suivantes.
- Adaptation des Horaires : En raison de la réduction de l'effectif de notre équipe, nous avons dû adapter nos horaires de travail pour pouvoir produire quelque chose de correcte. Cela impliquait des journées de travail plus longues et une planification rigoureuse pour maximiser notre productivité malgré les ressources humaines limitées.

4.3 Outils Utilisés

- OneNote : Utilisé pour la mise en place et la distinction de ce que nous avions réalisé et ce qu'il nous restait à faire. OneNote nous a aidé à organiser nos idées, nos notes de réunion et à suivre l'avancement global du projet.
- Git : Utilisé pour le contrôle de version. Nous avons tenté de mettre à jour nos branches de manière régulière pour être sûr que chacun disposait de la dernière version des codes.
- Discord : Utilisé pour les communications en temps réel. Les discussions sur Discord nous ont permis de travailler de manière coordonnée, commune et ainsi donc ainsi, décisions importantes. Nous avions des réalisé avant le commencement du projet un discord comprenant des canaux spécifiques pour les différentes parties du projet (Ressources, Code, Diagrammes,etc..) afin d'organiser les discussions et d'éviter les confusions. Discord a également été utilisé pour les appels vocaux lors des sessions après les différentes journées et réunions hebdomadaires.
- One Drive : Utilisé pour partager les documents word et les présentations Powerpoints. One Drive nous a permis de collaborer en temps réel sur des documents, de stocker et d'organiser les différents rapport et analyses à déposer, cela a permis d'assurer que tous nous avions accès aux informations les plus récentes et parallèle.

4.4 Communication et Collaboration

Description : La communication au sein de notre équipe de deux a été un point crucial pour la réalisation de projet. Cette dernière nous a permis d'assurer nous étions sur la même longueur d'onde, ce qui a été essentiel pour la coordination des efforts et la résolution rapide des problèmes. Nous avons mis en place plusieurs outils et pratiques pour maintenir une communication fluide et efficace.

Exemple :

- Réunions informelles : Bien que notre équipe ait été réduite, nous avons maintenu des réunions informelles via Discord. Chacun de nous deux partageait ses avancées et obstacles rencontrés, ce qui a permis de s'ajuster rapidement et de s'entraider. Ces longues réunions ont favorisé une communication claire et ont aidé à identifier et résoudre rapidement les problèmes.

4.5 Gestion des Défis Techniques

Description : Notre projet a rencontré des défis techniques qu'il a été important de gérer efficacement.

Défis rencontrés :

- GIT : Des problèmes de compréhension des différentes branches dans Git.
- Implémentation du Menu base / pause : l'implémentation initial pour la création du menu de départ a été difficile à inclure dans le fichier main.
- Difficultés de Communication : La nécessité de maintenir une communication claire et efficace avec l'effectif initiale.
- UML : La nécessité de structurer et d'interpréter correctement les diagrammes UML pour une mise en œuvre efficace.

Solutions Apportées :

- GIT : Mise en place de différentes questions posées aux professeurs afin d'éclaircir quelles fonctionnalités inclure et comment organiser ces points.
- Implémentation du Menu de Base / Pause : Reconsidération complète du menu. Nous avons revu notre conception initiale pour créer une structure plus modulaire. Le menu de base et le menu de pause ont été rassemblés en composants communs. Nous avons également utilisé les gestionnaires d'événements pour simplifier l'interaction entre ces derniers.

- Difficultés de Communication Au début du projet, nous avons rencontré des problèmes de communication avec nos anciens membres d'équipe, ce qui a conduit à des malentendus et des retards. Pour résoudre ce problème, nous avons décidé de nous séparer de nos anciens membres et de continuer le projet en équipe réduite mais plus cohérente. Cela a permis une meilleure coordination et une communication plus fluide.
- UML : Prise en compte des retours sur le diagramme UML et amélioration continue du modèle pour qu'ils reflètent précisément aux besoins du projet.

4.6 Retour sur Expérience

Ce qui a bien fonctionné :

- Collaboration : L'utilisation de Discord et de OneDrive a fortement amélioré notre collaboration et notre capacité à résoudre rapidement les problèmes. Ces outils ont permis une communication continue et claire, essentielle pour une équipe réduite.
- Méthodologie de Travail Intensif : La division du projet en phases distinctes et les réunions de bilan régulières ont permis de garder le projet sur la bonne voie. Cette approche a permis une concentration approfondie sur chaque aspect du développement et une intégration fluide des différentes parties du projet.
- Utilisation de Git : Bien que des problèmes de compréhension aient apparu au début, l'utilisation régulière de Git et la mise en place de règles strictes pour les commits et les fusions ont aidé à maintenir un code cohérent et à gérer les versions efficacement.
- Utilisation de OneNote : OneNote nous a aidé à organiser nos idées, nos notes de réunion et à suivre l'avancement global du projet. Cet outil a été essentiel pour garder une trace de ce qui avait été fait et de ce qu'il restait à faire.

Ce qui pourrait être amélioré :

- Gestion de GIT: Bien que nous ayons posé des questions aux professeurs et mis en place des pratiques pour minimiser les conflits, ceux-ci ont encore causé des retards. Une meilleure compréhension initiale des branches Git et une planification plus rigoureuse des fusions pourraient aider à éviter ces problèmes à l'avenir.
- Communication avec les Anciens Membres : Au début du projet, la mauvaise communication avec nos anciens membres d'équipe a conduit à des malentendus et des retards. Une meilleure structure de communication dès le départ aurait pu prévenir ces problèmes.
- Optimisation du Menu de Base / Pause : La création initiale du menu de départ a été difficile à inclure dans le fichier main. Une meilleure planification et une structure plus modulaire dès le départ auraient pu simplifier cette tâche.

Leçons Apprises :

- Importance de la Communication : Des réunions régulières, même informelles, et des discussions ouvertes sont essentielles pour le succès d'un projet d'équipe. Après nous être séparés de nos anciens membres, l'amélioration de notre communication a été cruciale pour la progression du projet.
- Adaptabilité : Être prêt à adapter les plans en fonction des défis rencontrés et des nouvelles informations est crucial pour progresser efficacement. La reconsidération complète du menu de base et de pause en est un bon exemple.
- Qualité du Code et Documentation : Maintenir un code propre, bien documenté et structuré a facilité la compréhension du code. L'utilisation de OneNote pour suivre l'avancement global du projet et documenter les idées a également été bénéfique.
- Utilisation de Diagrammes UML : Les retours donné sur le diagramme UML et son amélioration continue a été essentiels pour structurer et interpréter correctement les concepts du projet. Une meilleure formation et étude initiale sur les meilleures pratiques UML aurait pu améliorer encore plus cette partie.
- Gestion du Temps : Une planification détaillée et l'adaptation des horaires de travail ont été cruciales pour maximiser notre productivité. La gestion rigoureuse du temps a permis de respecter les délais et de produire un travail de qualité.
- Importance du sommeil **:** le sommeil est un élément clé pour maintenir une production maximale et optimale lors de projets de cette ampleur. Un bon repos permet de mieux gérer le stress et d'optimiser les performances de développement.