Support pour Speech

Introduction :

Underworld's Call : "Underworld's Call" est le titre de notre projet de jeu vidéo, inspiré par l'ambiance et le gameplay du célèbre jeu HADES. Ce jeu plonge les joueurs dans un monde souterrain mystérieux et périlleux où ils doivent affronter divers ennemis et surmonter des défis pour progresser.

Jeu immersif : Notre objectif est de créer une expérience de jeu profondément immersive. Cela inclut des graphismes de bonne qualité, une bande sonore captivante et des mécanismes de jeu engageants qui gardent les joueurs investis dans l'histoire et les défis du jeu.

Équipe recomposée : Le projet était réalisé de base par une équipe de 4, qui s'est ensuite recomposée en 2 équipes du a une mauvaise entente.

Contexte

Contexte du Projet

Projet intensif de 5 jours : Un projet rapide et concentré, conçu pour être complété en seulement quelques jours.

Objectifs : Développer un jeu vidéo complet avec des fonctionnalités évolutives et des graphismes immersifs.

Défis de collaborations : Travailler efficacement en équipe pour surmonter les obstacles de communication et de coordination.

Durée limitée : Respecter les délais serrés pour livrer un jeu fonctionnel et les divers dossiers dans le temps imparti.

Créativité exigée : Faire preuve d'originalité et d'innovation dans la conception des armes, des niveaux et des mécanismes de jeu tout en respectant le cahier des charges.

Résume du projet

Thème médiéval : Un jeu ancré dans un univers médiéval riche en histoire et en mystère.

Personnalisation : Possibilité pour les armes du personnage leur principal et ses capacités lors de l'ouverture du menu.

Ennemis : Deux types d'ennemis avec des comportements uniques à affronter.

Salle dans l'ambiance : Chaque salle est conçue pour correspondre à l'atmosphère médiévale, ajoutant à l'immersion.

Caméra 2D : Le jeu utilise une vue en 2D pour une expérience de jeu classique et fluide.

Difficulté modulable : Les joueurs peuvent ajuster la difficulté du jeu selon leurs préférences.

Défis croissants : Des défis de plus en plus complexes à mesure que le joueur progresse.

Combat de boss : Des combats épiques contre des boss à la fin de chaque round pour augmenter la difficulté et l'excitation.

Défis et solutions :

Effectif : Défi : Problèmes avec les anciens membres d'équipe qui a pu limiter ce qui a pu l'avancement du projet et augmenter la charge de travail individuelle chacune.

Solution : Optimiser la répartition des tâches en fonction des compétences spécifiques de chaque membre et utiliser des outils de gestion de projet pour suivre les progrès.

Gestion du stress : Défi : Les délais serrés et les exigences élevées ont provoqué du stress. Solution : Pratiquer des techniques de gestion du stress comme des pauses régulières, une communication ouverte et un soutien mutuel entre les membres de l'équipe.

Difficultés de communication : Défi : Travailler en ligne peut entraîner des malentendus et une mauvaise coordination. Solution : Utiliser des plateformes de communication en temps réel comme Discord, et documenter toutes les décisions importantes sur Google Docs.

GIT : Défi : Gérer les versions du code constamment.

Solution : Établir des branches claires pour chaque fonctionnalité, faire des commits fréquents et utiliser des pull requests pour réviser le code avant la fusion.

Méthodologie de travail : Défi : Maintenir une méthodologie de travail cohérente et efficace. Solution : Utiliser des méthodologies comme le cycle V pour structurer le travail, avec des phases de spécification, conception, développement, intégration, validation, et maintenance.

UML : Défi : Créer des diagrammes UML clairs et compréhensibles pour tous les membres de l'équipe. Solution : Utiliser des outils UML en ligne et collaboratifs pour élaborer des diagrammes de classes et de séquences, et assurer une documentation précise de la structure du code.

Erreurs de code / implémentation : Défi : Les bugs et les erreurs d'implémentation peuvent retarder le projet.

Solution : Effectuer des tests constamment

Gestion du temps / délais : Défi : Respecter les délais serrés tout en maintenant la qualité du jeu. Solution : Planifier soigneusement chaque étape du projet, suivre l'avancement quotidiennement, et ajuster les priorités en fonction des besoins pour s'assurer que les tâches critiques sont terminées à temps.

Conclusion

Le projet "Underworld's Call" a été une aventure intense et enrichissante pour notre équipe. En cinq jours, nous avons réussi à transformer une idée en un jeu vidéo immersif et captivant. Travaillant sous des délais serrés, nous avons surmonté des défis variés grâce à une collaboration efficace, une méthodologie rigoureuse et une créativité débordante.

Points Forts :

- **Thème et Ambiance :** Un univers médiéval bien conçu, offrant une expérience de jeu riche et immersive.
- **Personnalisation et Progression :** Un personnage principal évolutif et des améliorations en jeu, permettant aux joueurs de personnaliser leur expérience.
- **Défis et Combats de Boss :** Des ennemis variés et des combats de boss stimulants, ajoutant du piquant au gameplay.
- **Interface et Utilisabilité :** Une interface utilisateur intuitive et réactive, avec une caméra 2D pour une navigation fluide.
- **Collaboration et Communication :** Une utilisation efficace des outils de gestion de projet et de communication, tels que GIT et Discord, pour maintenir une coordination sans faille.

Défis Relevés :

- **Stress et Gestion du Temps :** En maintenant un esprit d'équipe solidaire et des pauses régulières, nous avons réussi à gérer le stress et à respecter les délais.
- **Problèmes Techniques :** Grâce à une approche méthodique de la détection et de la correction des erreurs de code, nous avons pu surmonter les obstacles techniques sans compromettre la qualité du jeu.
- **Documentation et UML :** Une documentation claire et détaillée, appuyée par des diagrammes UML précis, a facilité la compréhension et la mise en œuvre des fonctionnalités.

Leçons Apprises :

- **Importance de la Planification :** Une planification minutieuse et une gestion rigoureuse du temps sont essentielles pour le succès de tout projet à court terme.
- **Communication Efficace :** Maintenir des canaux de communication ouverts et réguliers est crucial pour coordonner les efforts et résoudre les problèmes rapidement.
- **Adaptabilité et Innovation :** Être prêt à s'adapter aux changements et à innover est clé pour surmonter les imprévus et ajouter de la valeur au projet.

En conclusion, "Underworld's Call" n'est pas seulement un jeu, mais aussi le reflet de notre capacité à travailler ensemble sous pression, à résoudre des problèmes complexes et à créer quelque chose d'unique et d'impactant. Nous sommes fiers de ce que nous avons accompli et enthousiastes quant aux futures possibilités d'amélioration et d'extension de notre projet.

Conceptions et développement :

1. Conception :

- Analyse des besoins :
  Comprendre les exigences du cahier des charges et définir les objectifs principaux que notre jeu à besoin à l'aide des interactions avec les clients disponibles pour répondre à nos questions.
- Création des Diagrammes UML :
  Élaboration de diagramme de classes évolutif par rapport à l'avancement de nos idées et du projet.
- Mise en place d'un planning pour le développement :
  Décomposition des tâches et création d'un planning pour assurer une certaine production. Notamment grâce à une « to do list » attitrée par collègue mise en place.

2. Développement :

- Développement divisé en plusieurs catégories :
  - Mécanisme de jeu :
    - Combat dynamique, implémentation de combat rapides contre divers ennemis et boss. Utilisations d'attaques au càc et à distance.
    - Progession RogueLike, passage d'une salle à une autre grâce à une dalle centrale faisant une téléportation.
    - Amélioration d'une capacité possible via le menu pause, celle-ci améliore la vitesse de déplacement.
  - Interface utilisateur :
    - Menus :
      - Mise en place de menu de démarrage
      - Mise en place d'un menu de pause.
      - Possibilité de sauvegarder et de recommencer au point de sauvegarde.
  - Méthodologie de développement :
    - Utilisation de Git :
      - Mise en place de versions pour actualiser nos versions de git et travail dans plusieurs branches. Ensuite, merge dans le main.
      - Conventions de nommage, respect des conventions de commits pour une clarté pour le professeur responsable de l'inspection du dépôt github.
      - Documentation, rédaction de README.md et d'une documentation technique détaillant la manière sur comment isntaller le jeu

Explications des Codes

Player :

Initialisation (méthode __init__): Le joueur est initialisé avec des propriétés telles que la position (x, y), la taille (width, height) et la santé. Différentes images pour le joueur et les armes sont chargées pour représenter différents états et actions (comme se déplacer dans différentes directions et attaquer).

Attributs et Propriétés : La classe utilise des attributs privés (indiqués par __) pour la position, l'image actuelle, la direction, la santé, et plus encore. Les propriétés (@property) sont utilisées pour encapsuler l'accès à ces attributs privés, permettant des mises à jour contrôlées et cohérentes.

Déplacement (move méthode): La méthode move ajuste la position du joueur en fonction des touches pressées (comme 'q', 'd', 'z', 's' pour gauche, droite, haut, et bas). Elle vérifie également si le joueur reste dans les limites de l'écran avant d'appliquer le déplacement.

Attaque (attack, swingHammer, shootArrow méthodes): Le joueur peut attaquer avec différentes armes. La méthode swingHammer gère l'attaque avec un marteau, tandis que shootArrow permet de tirer des flèches. Chaque arme a des images associées pour ses différents états (comme tiré ou non).

Gestion de la santé et de l'invincibilité (take_damage, update_invincibility méthodes): Ces méthodes gèrent la santé du joueur et un état temporaire d'invincibilité après avoir reçu des dégâts, empêchant ainsi des dommages supplémentaires pendant un bref laps de temps.

Changement d'arme (switchWeapon méthode): Permet au joueur de basculer entre différents types d'armes disponibles, ajustant les images et les rectangles de collision associés.

Dessin et mise à jour (draw, updateArrows méthodes): Ces méthodes sont responsables de dessiner le joueur et ses flèches à l'écran, et de mettre à jour leur état pour refléter le déplacement et les actions en cours.

Enemy :

Initialisation (__init__) : Les ennemis sont initialisés avec une liste d'images, une vitesse, une taille (new_size), et des dimensions de l'écran (screen_width, screen_height). Les images sont chargées et redimensionnées.

Attributs : Incluent la vitesse (_speed), la santé (_mobHealth), les dimensions (_screen_width, _screen_height), et une position initiale aléatoire.

Méthodes principales :

random_position() : Génère une position initiale aléatoire sur l'écran.

check_oob(x, y) : Vérifie si les coordonnées fournies sortent des limites de l'écran.

draw(surface) : Dessine l'ennemi sur une surface donnée, animant les images selon la vitesse d'animation.

update () : Met à jour la position de l'ennemi de manière aléatoire tout en restant dans les limites de l'écran.

move_towards_player(player_x, player_y) : Déplace l'ennemi en direction du joueur en calculant la direction et la distance.

applyDamage(damage) : Applique des dégâts à l'ennemi et retourne True si l'ennemi est mort.

isAlive() : Vérifie si l'ennemi est encore en vie.

Sous-classes Lizard, Orc, Boss :

Chacune de ces classes hérite de Enemy et est spécialisée avec des images spécifiques et des paramètres tels que la taille et la vitesse. Par exemple, le Boss a plus de santé et une vitesse légèrement accrue par rapport aux autres ennemis.

Arrow

Initialisation (__init__ method) :

Paramètres : La position initiale (x, y), la direction de mouvement (direction), le chemin vers l'image de la flèche (image_path), et la vitesse de déplacement (speed, valeur par défaut de 1.9).

Chargement et mise à l'échelle de l'image : L'image de la flèche est chargée à partir du chemin spécifié, puis redimensionnée à sa taille originale.

Rectangle de positionnement : Un rectangle (pygame.Rect) est utilisé pour gérer la position et les dimensions de la flèche. Ce rectangle est initialisé à la position spécifiée.

Propriétés (property et setter) :

rect : Permet d'accéder et de modifier le rectangle (pygame.Rect) qui gère la position et les dimensions de la flèche.

Méthodes principales :

update() : Met à jour la position de la flèche en fonction de sa direction. La position est ajustée en ajoutant ou soustrayant la vitesse à l'axe correspondant (x ou y). La méthode suit également la distance parcourue par la flèche et retourne False si cette distance dépasse une distance maximale (__max_distance de 250 pixels), ce qui peut être utilisé pour détruire la flèche.

draw(screen) : Dessine la flèche sur une surface donnée (screen). Utilise la méthode blit de pygame pour afficher l'image de la flèche à sa position actuelle

Main :

Ce script Python configure un environnement de jeu complet utilisant la bibliothèque pygame pour créer un jeu d'action avec des éléments interactifs comme des joueurs, des ennemis, et des projectiles.

Initialisation et Configuration :

Pygame : Initialisation des modules de base, y compris le système de son.

Fenêtre de jeu : Définition de la taille et du titre de la fenêtre de jeu, ainsi que de l'icône.

Images de fond : Chargement et mise à l'échelle des images utilisées pour le décor du jeu.

Sons : Chargement et configuration des effets sonores et de la musique de fond.

Classes et Objets Importés :

Importation des classes Player, Arrow, et Enemy depuis d'autres fichiers pour gérer les personnages du joueur, les flèches, et les ennemis.

Gestion des Ennemis et du Joueur :

Ennemis : Création d'instances de différents types d'ennemis (Lizard, Orc, Boss).

Joueur : Initialisation de l'objet joueur avec position et dimensions spécifiques.

Fonctions de Gameplay :

drawHealth : Dessine des icônes de cœur sur l'écran pour représenter la santé du joueur.

checkCollide et checkHammerCollide : Gèrent les collisions entre les flèches, le marteau du joueur et les ennemis.

all_enemies_defeated : Vérifie si tous les ennemis ont été vaincus.

load_next_wave et create_enemies : Chargent de nouvelles vagues d'ennemis.

Menus et Interaction Utilisateur :

main_menu et pause_menu : Affichent les menus principal et de pause, permettant à l'utilisateur de choisir entre différentes options de jeu.

game_over_screen : Affiche un écran de fin de jeu lorsque le joueur perd toute sa santé.

save_game : Sauvegarde l'état actuel du jeu dans un fichier JSON.

Boucle Principale de Jeu :

Cette boucle gère les entrées de l'utilisateur (touches pressées), déplace le joueur, vérifie les collisions, et met à jour les éléments du jeu à chaque cycle. Elle redessine également l'écran à chaque itération pour refléter les changements.

Gestion des États de Jeu :

La boucle vérifie l'état de santé du joueur pour passer à l'écran de game over si nécessaire, et gère les transitions entre les vagues d'ennemis.