Vent du Nord

Projet C++ - SDL2

Étudiants :
BOUDIA Yanis p2408696
KELAI Rayan p2411942
DAMICHE Hani p2409005

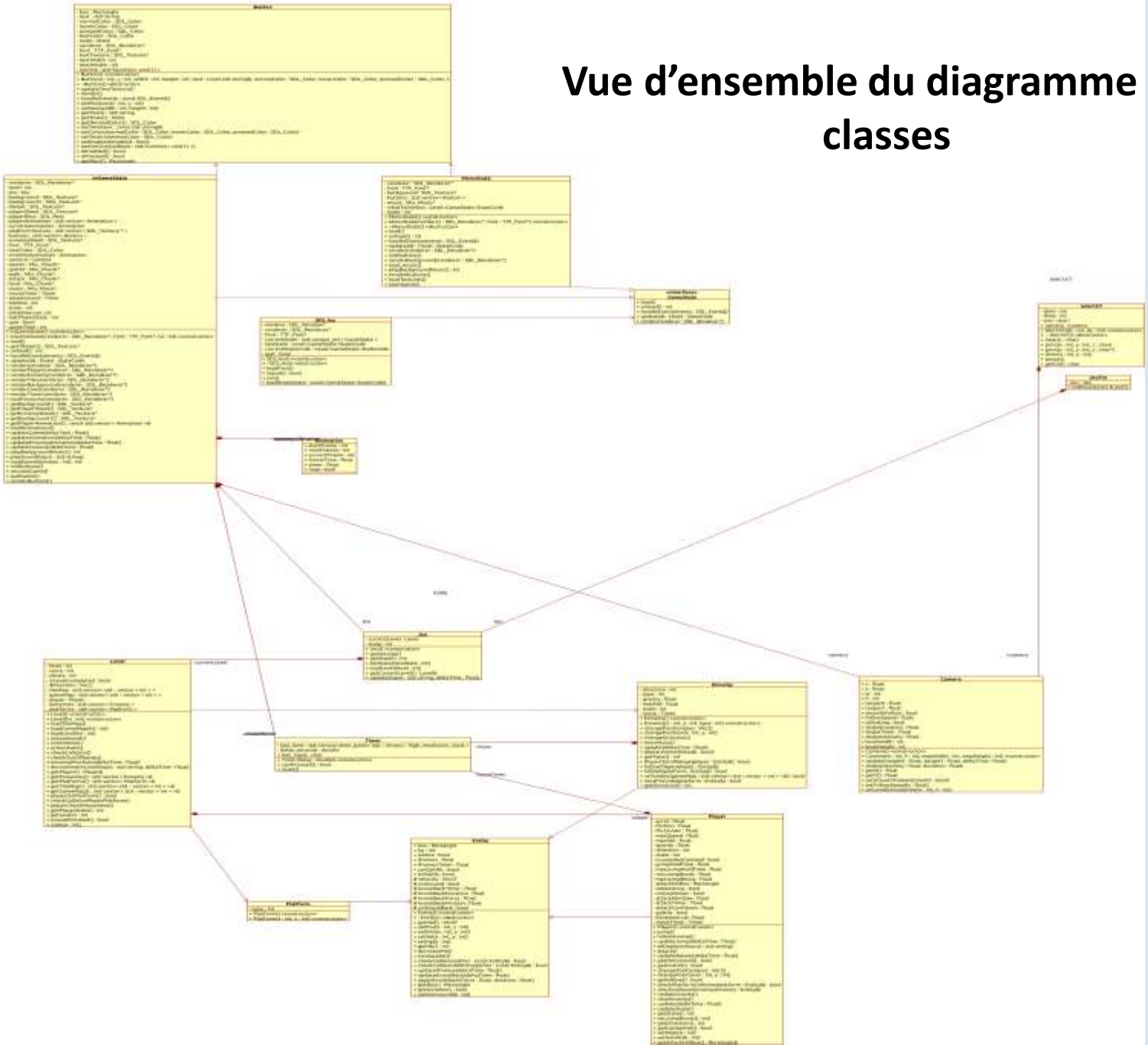
Encadrant : NICOLAS Pronost

Année universitaire : 2024 – 2025

Principe de l’application

- Vent du Nord est un jeu de plateforme en 2D développé en C++ avec la bibliothèque SDL2. Le joueur incarne un personnage qui doit progresser à travers desn niveaux en évitant des obstacles et des ennemis, en sautant sur des plateformes et en atteignant l’arrivée sans mourir. Tout ça , avec un nombre de vies initialisé a 5 et un timer et le but sera de finir le niveau avant la fin du timer .
- Le gameplay repose sur la gravité, les collisions, et des déplacements fluides. L’environnement est dynamique, avec une caméra qui suit le joueur. Des ennemis un peut partout sur les map aussi et a dhaque collision avec un ennemi le nombre de vies diminue de 1 .
- L’application est structurée autour de plusieurs classes orientées objet : Player, Ennemy, Level, InGameState…

Vue d'ensemble du diagramme des classes



Classe Ennemy

- **Attributs principaux**
- direction : orientation (LEFT / RIGHT).
- type : type d'ennemi (on a pas pu finir ça).
- velocity, gravity, maxFall : gestion du mouvement et de la chute.
- move : timer pour limiter la fréquence des déplacements.
- iFrames, hp : gestion des points de vie et de l'invincibilité temporaire.

- ⚙️ **Méthodes principales**
- update(dt) : met à jour la position et la gravité.
- changeDirection() : inverse la direction horizontale.
- vaTomber(map) : détecte un trou devant l'ennemi pour éviter les chutes.
- hitWall(platform) : détecte les collisions latérales avec les murs.
- PlayerOutofRange(player) : détermine si le joueur est dans la zone de détection.
- followPlayer(player) : ajuste la direction pour suivre le joueur si proche.
- moveAuto() : logique globale de déplacement autonome (utilisé dans Level)

Rôle de la classe

•Représente un ennemi dans le jeu.
•Hérite de Entity, avec des comportements spécifiques comme le mouvement automatique, les collisions avec le décor.

Classe Player

- 🎮 **Rôle de la classe**
- Représente le personnage principal contrôlé par le joueur. Elle hérite de Entity et elle gère :
- les déplacements (q/d),
- les sauts (avec boost selon durée d'appui),
- les attaques (hitbox, cooldown),
- les collisions (plateformes et ennemis),
- l'état d'animation (IDLE, JUMP, etc.),
- la gravité, les dégâts et l'invincibilité.
- **Attributs principaux**
- velocity, accel, gravity, friction → physique du joueur.
- jumpHoldTime, maxJumpBoost → saut progressif.
- attackHitBox, attackTimer, onCoolDown → gestion des attaques.
- gotHit, canGetHit, iFramesTimer → gestion des dégâts.
- state (enum PlayerState) → animation actuelle.
- **Méthodes clés**
- seDeplacer(input) : gère les entrées clavier (q, d, espace, m).
- jump() / releaseJump() / updateJump(dt) : saut avec variation.
- attack() / updateAttack(dt) : attaque avec hitbox temporaire.
- update() : mise à jour centrale du joueur.
- checkPlatformCollision() : détecte la position de contact (haut, bas, etc.).

Classe Level

- **Rôle de la classe**
- - Gère un niveau complet du jeu : joueur, plateformes, ennemis, carte.
- - Permet de charger, initialiser, réinitialiser et dérouler un niveau.

- **Attributs clés**
- - player : objet Player contrôlé.
- - ennemies : liste des ennemis du niveau.
- - platforms : plateformes sur lesquelles marcher.
- - gameMap : carte d'entités chargée depuis un fichier .txt.
- - level, score, nbLife, isLevelCompleted : état global du niveau.

- **Méthodes principales**
- - loadGameMap(), initEntities() : chargement de la carte et génération des entités.
- - deroulementLevel(input, dt) : boucle logique principale du niveau.
- - ennemyMovAuto(dt) : déplacement autonome des ennemis (IA).
- - checkPlatformCollision(), checkCollisionEnnemy() : gestion des collisions.

- **Importance**
- - Centralise toute la logique d'un niveau.
- - Permet transitions entre niveaux, contrôle de fin, IA ennemie.

Classe InGameState

- 🕹️ **Rôle de la classe**
- Gère l'état du jeu pendant une partie : rendu, logique, interactions, sons.
- S'occupe de tout ce qui est affiché à l'écran (sprites, tuiles, fond, UI).
- Permet de passer entre les états : jeu, pause, mort, fin.

- 🎁 **Attributs principaux**
- renderer, font, camera, jeu (le modèle logique du jeu).
- Textures : tileSet, background, playerSheet, ennemySheet, etc.
- Animations : playerAnimation, ennemyAnimation.
- Sons : music, walk, jump, attack, gotHit, etc.
- buttons : boutons d'interface pour Pause, Resume...
- state : CONTINUE, PAUSE, DEAD, END…

- ⚙️ **Méthodes principales**
- load(), render(), update(dt), handleEvents().
- updateGame(), updateCamera(), updateAnimation().
- renderTiles(), renderPlayer(), renderEnnemy(), renderLives(), renderTimer().
- loadSounds(), playSound(input), initButtons().

- 🌟 **Pourquoi c'est important**
- C'est le pont entre le moteur de jeu et le joueur.

- Gère tout ce que le joueur voit et entend.

- Rend l'expérience fluide, cohérente et interactive.

Conclusion

- ✓ **Ce qui fonctionne** :
- - Le moteur de jeu est fonctionnel : déplacement du joueur, ennemis, collisions.
- - Bonne structure objet : séparation des responsabilités entre les classes (Player, Level, etc.).
- Données : Le nombre de vies du joueur et le timer marchent également, un menu principal qui permet de démarrer la partie , possiilté de faire pause et de quitter la partie . Enfin , changement de niveau
- ⚠**Difficultés rencontrée**s :
- - Débogage des collisions et interactions entre entités ou encore entres les platformes et les entités (surtout dans le mode texte).
- - Structure globale entre le core le sdl et le mode texte : au début , on a du refaire la structure globale du projet et donc le diagramme a complétement changé.
- Faire en sorte que le mode texte et le sdl marchent tous les deux pareil : en effet on avait du mal a accorder entre les deux la vitesse ou encore la gravité .
- 🔁 **À améliorer avec plus de temps** :
- - Ajouter plusieurs niveaux et un système de menus complet.
- - Intégrer des animations et des powerups .
- - Optimiser le jeu de maniére a ce que l'utilisateur soit amplement satisfait de son experience .