Vent du Nord

Projet C++ - SDL2

Étudiants :
BOUDIA Yanis p2408696
KELAI Rayan p2411942
DAMICHE Hani p2409005

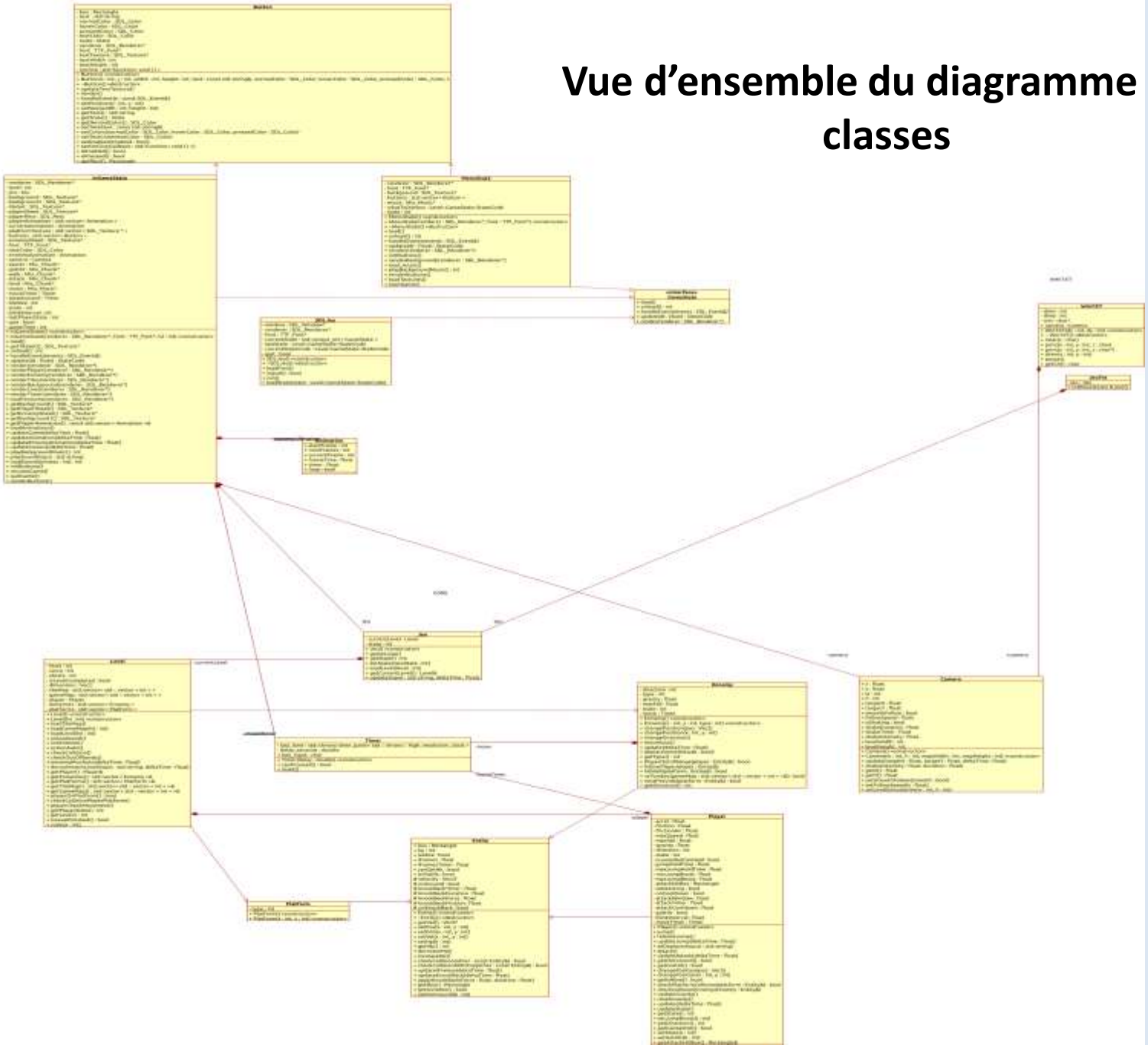
Encadrant : NICOLAS Pronost

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

- 📌 **Rôle** : Représente un ennemi dans le niveau. Hérite de la classe Entity.Ennemy représente un adversaire dans le niveau du jeu.
- Il hérite de la classe Entity, ce qui signifie qu'il possède déjà des propriétés communes comme la position, la taille, la gestion de la physique ou encore le dessin à l'écran.
- Le rôle principal d'un ennemi est de patrouiller dans une zone définie et d'interagir avec le joueur (le tuer s'il y a collision).
- **Attributs principaux** :
- Vec2f position : position de l'ennemi dans le monde
- Vec2f velocity : direction et vitesse du déplacement
- Vec2f initPos : position de départ de l'ennemi
- float patrolDistance : distance maximale de déplacement depuis la position initiale
- **Méthodes :**
  - update(Level&) : met à jour le déplacement et les collisions.
  - draw(renderer, camera) : affiche l'ennemi.
  - kill() : change l'état de vie de l'ennem.
- 🔗 Interagit avec : Level (terrain), Player (collision = game over).

Classe Player

- 📌 **Rôle** : Contrôle le personnage principal. Hérite de Entity. Gère les déplacements, les sauts, la physique, et les collisions
- ⚙**Attributs** :
- position, velocity, acceleration
- nbJumpLeft : nombre de sauts disponibles
- isJumping, isDead : états du joueur
- 🕹 **Méthodes** :
  - handleInput() : gère les touches pressées.
  - update(Level&) : applique la gravité et gère les collisions.
  - draw(renderer, camera) : affiche le joueur.
- 🔗 Interagit avec : Level (sol, plateformes), Camera (suivi), Ennemy (mort).

Classe Level

- 📌 **Rôle** : Représente un niveau du jeu (plateformes, ennemis, point de départ).
- ⚙**Attributs** : listes de plateformes et ennemis, spawn du joueur.
- 🛠 **Méthodes** :
- - load(niveau) : charge les éléments depuis un fichier.
  - update(Player&) : met à jour les entités.
  - draw(renderer, camera) : affiche les objets du niveau.
- 🔗 Fournit le décor, les obstacles, et le spawn pour le joueur.

Classe InGameState

- 📌 Rôle : InGameState représente l'état "en jeu" actif, c'est-à-dire lorsque la partie est en cours . Elle est responsable de coordonner tous les éléments du gameplay : le joueur, le niveau, la caméra, et le temps.

- ⚙Attributs : contient un Player, un Level, une Camera et un Timer.

- 🛠 Méthodes :
-

  - handleEvents() : gère les événements clavier.
  - update() : met à jour tous les composants.
  - render(renderer) : affiche la scène complète.
- 🔗 Coordonne les interactions entre les composants du jeu.

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 .