MINI PROJET EN C++

APPLICATION DE JEU DE COURSE À VÉLO

Réaliser par :

Nom et prenom : Oussama chekroun

N°=2229322

Nom et prenom : Aimrane Alyachiri

N°= 2334282

1.L’introduction :

Objectif du projet :

Le projet **CYCLIST DODGER** est un jeu de course à vélo développé en **C++** avec **SDL2**, dans le cadre d’un mini-projet de développement d’une application interactive. Son but est de proposer un jeu **, intuitif à jouer et rapide à terminer**, tout en mettant en pratique les notions de **programmation orientée objet**, **gestion d’événements** et **affichage graphique**. Ce jeu permet également de se familiariser avec les bases de la **conception de jeux vidéo**, notamment la gestion de collisions, l’animation, les entrées clavier et l’architecture logicielle modulaire.

Photo

Description de jeux :

CYCLIST DODGER est un jeu d’arcade dans lequel le joueur incarne un cycliste roulant sur une route semée d’obstacles (pierres).
Le but du jeu est de **parcourir la plus longue distance possible** en **évitant les obstacles** qui apparaissent progressivement à l’écran.

Le joueur peut **se déplacer latéralement** (gauche/droite) à l’aide des touches du clavier pour esquiver les obstacles.
Une collision avec une pierre met fin à la partie.

Le jeu comprend :

- Un **menu principal** pour démarrer ou quitter ou A propre .
- Un **système de score ou de chronomètre** qui mesure la performance du joueur.
- DES **interfaces simple** en 2D basée sur SDL2.

Ce jeu allie **réactivité**, **coordination** et **anticipation**, tout en restant accessible et amusant.

L’image utilise dans l’interface principale de jeux.

2.L’architucture de code :

Organization générale :

Le jeu **CYCLIST DODGER** est développé en C++ selon une **architecture orientée objet**.
Chaque entité principale du jeu (joueur, obstacles, interface, etc.) est représentée par une **classe dédiée**, ce qui permet une séparation claire des responsabilités et facilite la maintenance et l'évolution du code.

Les responsabilités sont réparties comme suit :

- Menu : affiche le menu principal et gère la navigation entre les états du jeu.
- About :affiche bref explication sur le jeux.
- Game : gère la boucle principale du jeu, les événements SDL, et le rendu général.
- Player : contrôle le cycliste (position, mouvements, affichage).
- Pierre : gère les obstacles que le joueur doit éviter.
- Compteur : mesure le temps.

3.Les Classe Principales :

3.1.les Classe :

Classe Menu:

La classe Menu est responsable de l'affichage du menu principal du jeu. Elle constitue le point d'entrée graphique permettant au joueur d'interagir avec l'interface avant de commencer une partie.

Les attributs de la classe :

- SDL_Window* windows1 :
  La fenêtre principale du menu.
  Permet d'accéder aux autres fenêtres du jeu (par exemple, la fenêtre "À propos" ou "Play").
- SDL_Renderer* render :
  Le moteur de rendu associé à la fenêtre windows1.
  Il permet d'afficher les textures (comme les boutons et l'arrière-plan) à l'écran.
- SDL_Texture* BG :
  Texture représentant l'arrière-plan du menu principal (par exemple, une image ou une couleur de fond).
- SDL_Texture* btnplay :
  Texture du bouton "Play", utilisé pour démarrer la partie. Ce bouton est lié à la classe Game.
- SDL_Rect P :
  Rectangle définissant la position et la taille du bouton "Play" sur l'écran.
- SDL_Texture* btnProprie :
  Texture du bouton "Propriétés" ou "À propos" (ou "About"). Ce bouton permet d'accéder à la section "À propos" où les informations sur le jeu sont affichées.
- SDL_Rect Prop :
  Rectangle définissant la position et la taille du bouton "Propriétés" ou "À propos" sur l'écran.
- SDL_Texture* btnQuite :
  Texture du bouton "Quitter", permettant de fermer le jeu.
- SDL_Rect Q :
  Rectangle définissant la position et la taille du bouton "Quitter" sur l'écran.
- int numero = 0 :
  Variable utilisée pour indiquer le choix sélectionné par l'utilisateur.
  - 1 → Jouer (Play)
  - 2 → À propos (About)
  - 3 → Quitter (Quit)
  - 0 → Aucune sélection
- bool Running :
  Booléen qui contrôle la boucle du menu principal.
  Tant que Running == true, le menu reste affiché et interactif.

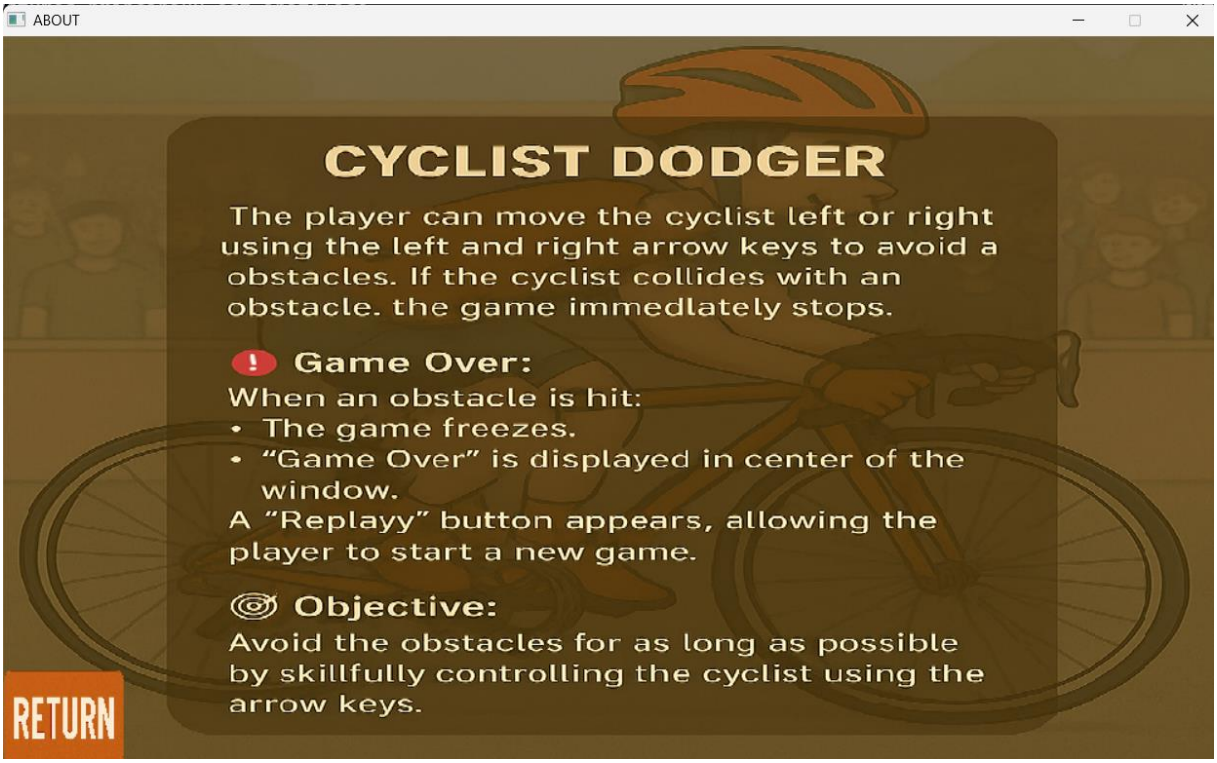
Les méthodes de la classe :

- Menu(const char* BGmenu, const char* imgbtnP, const char* imgbtnQ, const char* imgbtnProp) :

Constructeur de la classe.
Cette fonction est appelée lors de la création d’un objet Menu.
Elle prend en paramètres les chemins d'accès aux images nécessaires :
    - BGmenu : image de fond du menu,
    - imgbtnP : image du bouton "Play",
    - imgbtnQ : image du bouton "Quitter",
    - imgbtnProp : image du bouton "À propos".
      Le constructeur initialise la fenêtre (windows1), le moteur de rendu (render), et les textures des images. Il définit également les rectangles de position pour les boutons du menu.
- ~Menu() :
  Destructeur de la classe.
  Appelé automatiquement lorsque l'objet Menu est détruit. Il appelle la méthode clean() pour libérer les ressources utilisées par la classe.
- void Event() :
  Gère les **événements** du menu (clics de souris et fermeture de la fenêtre).
  Cette fonction est appelée à chaque boucle de rendu pour :
    - Détecter si un bouton est cliqué,
    - Mettre à jour la variable numero en fonction du bouton sélectionné,
    - Mettre à jour le booléen Running pour arrêter la boucle du menu si l'utilisateur clique sur un bouton ou ferme la fenêtre.
- void renderer() :
  Affiche les éléments du menu sur l'écran.
  Cette fonction :
    - Efface le rendu précédent avec SDL_RenderClear(),
    - Affiche le fond (BG) et les boutons (btnplay, btnProprie, btnQuite),
    - Rafraîchit l'affichage avec SDL_RenderPresent().
- void RunMenu() :
  Fonction principale qui contient la **boucle d'exécution** du menu.
  Tant que Running est vrai :
    - Elle appelle Event() pour gérer les événements (clics, fermetures de fenêtres),
    - Puis elle appelle renderer() pour afficher l'interface du menu.
      La boucle continue jusqu'à ce que l'utilisateur fasse un choix (Play, Quit, About).
- int getNumber() :
  Fonction d’accès qui retourne la valeur de numero, indiquant le choix de l’utilisateur :
    - 1 → Jouer (Play),
    - 2 → À propos (About),
    - 3 → Quitter (Quit).
- void clean() :
  Libère les ressources associées à la classe Menu.
    - Détruit les textures (BG, btnplay, btnProprie, btnQuite),
    - Ferme le renderer et la fenêtre windows1,
    - Appelle SDL_Quit() pour fermer proprement SDL.

Classe About:

La classe About est responsable de l'affichage de la fenêtre "À propos" du jeu. Elle permet au joueur de consulter des informations comme le nom du projet, les crédits, ou toute autre mention utile, et elle contient un bouton pour revenir au menu principal.



L'image utilise dans About.

Fonctionnalités principales :

- Affiche une nouvelle fenêtre avec des informations sur le jeu.
- Contient un **bouton "Retour"** permettant de revenir au menu principal.
- Permet au joueur de quitter la fenêtre "À propos" pour revenir à l'interface principale.

Les attributs de la classe :

- SDL_Window* about :
  La **fenêtre SDL** dédiée à la section "À propos". Cette fenêtre s'affiche lorsque l'utilisateur choisit de voir les informations sur le jeu.
- SDL_Renderer* renderer :
  Le moteur de **rendu graphique** associé à la fenêtre about. Ce moteur est utilisé pour afficher les textures (image de fond, bouton retour, texte).
- SDL_Texture* imgAbout :
  Texture représentant le **fond ou l'image** d'information dans la fenêtre "À propos". Cela peut inclure un logo ou une présentation graphique du jeu.

- SDL_Texture* revenir :
  Texture du **bouton "Retour"**, permettant à l'utilisateur de revenir à la fenêtre du menu principal.
- SDL_Rect rctR :
  Un **rectangle SDL** définissant la position et la taille du bouton "Retour" à l'écran. Il est utilisé pour gérer la zone cliquable du bouton.
- int isnumber = 0 :
  Variable qui stocke le choix de l'utilisateur :
  - 0 : aucune action,
  - 1 : retour au menu principal.
- int Running = 0 :
  Booléen utilisé pour contrôler l'exécution de la fenêtre "À propos". Tant que Running == 1, la fenêtre reste ouverte.

Les méthodes de la classe :

About(char, char*)*

Constructeur de la classe.
Ce constructeur est appelé lors de la création d'un objet About.
Il reçoit deux chemins en paramètres :

- Un pour le fond d'écran (imgAbout),
- Un autre pour l'image du bouton retour (revenir).

Le constructeur initialise la fenêtre about, le renderer, les textures et la position du bouton retour

~About()

Destructeur de la classe.
Appelé automatiquement à la destruction de l'objet About.
Il libère toutes les ressources allouées, notamment :

- Les textures (fond, bouton retour),
- Le renderer et la fenêtre.

void RunAbout()

Fonction principale de la classe, contenant la **boucle d'exécution** de la fenêtre "À propos" :

- Tant que Running == 1 :
  - Appelle la méthode Event() pour gérer les événements (clics, fermeture de la fenêtre),
  - Appelle la méthode render() pour afficher les éléments à l'écran.

Cela crée une boucle qui reste active jusqu'à ce que l'utilisateur clique sur "Retour" ou ferme la fenêtre.

void Event()

Gère les **événements utilisateur** dans la fenêtre "À propos" :

- Détecte la fermeture de la fenêtre ou un clic sur le bouton "Retour",
- Met à jour isnumber pour indiquer l'action choisie par l'utilisateur,
- Modifie Running à false lorsque l'utilisateur clique sur "Retour" ou ferme la fenêtre.

void render()

Affiche les éléments de la fenêtre "À propos" :

- Affiche l'image de fond (imgAbout),
- Affiche le bouton "Retour" (revenir),
- Utilise SDL_RenderPresent() pour mettre à jour l'affichage et rendre visibles les modifications.

void clean()

Libère toutes les ressources utilisées par la classe :

- Détruit les textures (imgAbout, revenir),
- Ferme le renderer et la fenêtre,
- Appelle SDL_Quit() pour fermer proprement SDL.

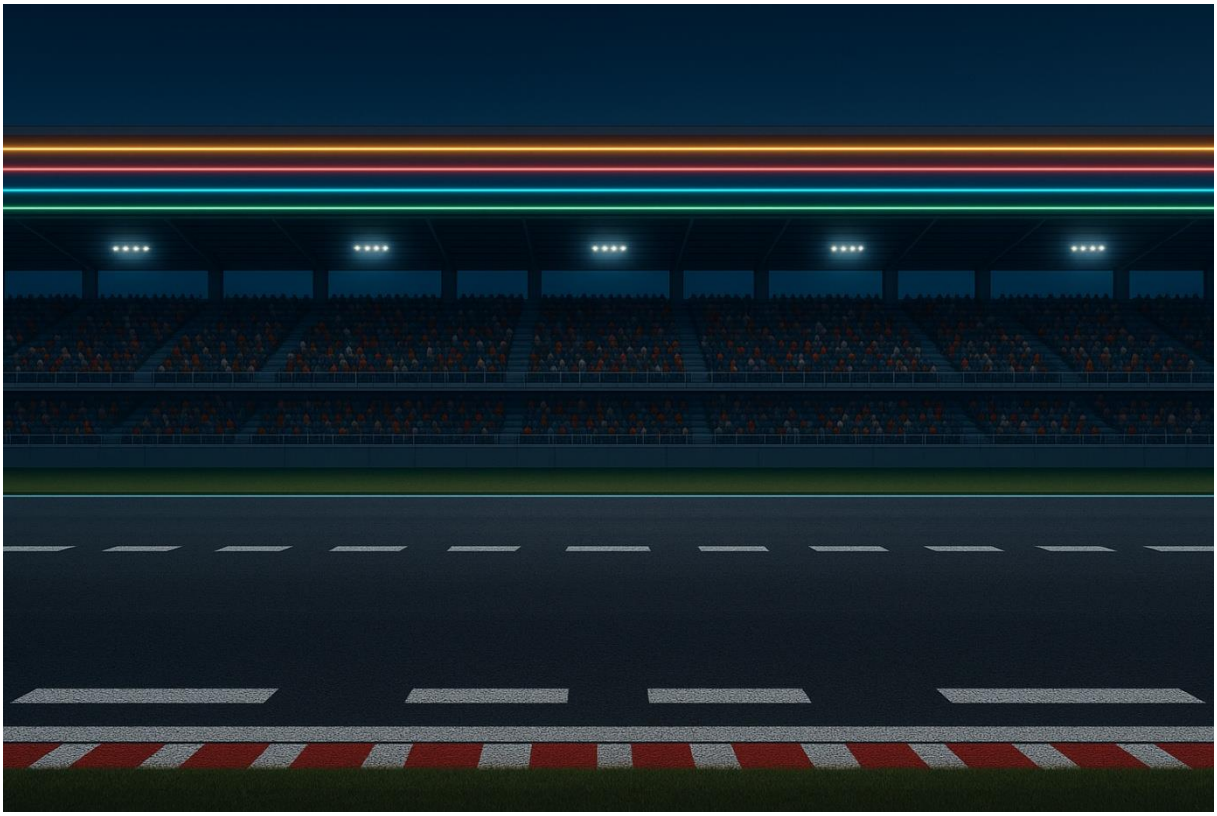
int number()

Fonction d'**accès** pour récupérer la valeur de isnumber :

- 1 : retour au menu principal.
- Permet à la classe appelante (comme Menu) de savoir si l'utilisateur a choisi de revenir au menu principal.

Classe Game :

La classe Game représente la **partie principale du jeu**, gérant l'interaction du joueur avec les objets du jeu, ainsi que la logique du jeu en temps réel (mouvement, score, collisions, etc.).
Elle permet d'afficher les éléments graphiques, gérer les événements utilisateur et maintenir l'état du jeu (en cours ou terminé).



L’image de backgrounde Game.

Les attributs de la classe :

- Player joueur :
  Instance de la classe Player, représentant le **joueur** dans le jeu. Ce joueur peut se déplacer, éviter des obstacles, et interagir avec l'environnement du jeu.
- pierre pierre :
  Instance de la classe pierre, représentant les **obstacles** (pierres) que le joueur doit éviter dans le jeu.
- Conteur conteur :
  Instance de la classe Conteur, responsable du **compte du score** et du **temps** dans le jeu.
- SDL_Window* window :
  La **fenêtre SDL** dans laquelle le jeu est affiché. C'est la fenêtre principale du jeu.
- SDL_Renderer* rennder :
  Le **moteur de rendu graphique** associé à la fenêtre window. Il permet d'afficher les éléments graphiques (arrière-plan, objets du jeu, etc.).
- SDL_Texture* background1 :
  Texture représentant le **fond** du jeu, qui défile pendant que le joueur avance.
- SDL_Rect dest1 :
  Un rectangle SDL représentant la zone où le fond background1 est affiché à l'écran. Cela permet de faire défiler le fond de manière fluide.
- SDL_Rect dest2 :
  Un autre rectangle pour afficher la **seconde partie du fond** afin de créer l’effet de défilement continu.

- SDL_Texture* btnReplay :
  Texture du **bouton "Rejouer"** qui apparaît à la fin du jeu (lorsque la partie est terminée) pour permettre au joueur de relancer une nouvelle partie.
- SDL_Rect rctReplay :
  Rectangle de la zone cliquable pour le bouton "Rejouer".
- SDL_Texture* G_over :
  Texture de l'écran **Game Over**, affiché lorsque le joueur a perdu.
- SDL_Rect rctover :
  Rectangle pour définir où l'écran "Game Over" est affiché à l'écran.
- Le **temps du compteur**, utilisé pour mesurer le temps écoulé durant la partie.
- Uint32 timerest = 0 :
  Le **temps restant** dans la partie ou le niveau. Utilisé pour gérer un chronomètre..
- int roadScroll = 0 :
  Variable représentant la position de défilement du **fond de route**. Cette variable est modifiée pour créer l'effet de mouvement de la route sous les pieds du joueur.
- bool Running :
  Booléen qui indique si le jeu est toujours en cours d'exécution. Si Running est false, cela signifie que la partie est terminée ou en pause.
- bool overG = false :
  Booléen qui indique si la partie est terminée (Game Over). Si overG est true, l'écran "Game Over" s'affiche.

Les méthodes de la classe :

Game(const char title, int xpos, int ypos, int width, int height, bool show, SDL_Rect rctP, SDL_Rect rctCol1, SDL_Rect rctCol2, Conteur C)*

Constructeur de la classe.
Le constructeur est appelé lors de la création d'un objet Game. Il initialise la fenêtre et le rendu SDL, charge les textures (fond, boutons, etc.), et configure les objets du jeu (joueur, obstacles, etc.).
Les paramètres incluent :

- title : le titre de la fenêtre,
- xpos, ypos : la position de la fenêtre,
- width, height : la taille de la fenêtre,
- show : booléen pour afficher ou non la fenêtre,
- rctP, rctCol1, rctCol2 : rectangles pour les positions du joueur et des obstacles,
- C : un objet Conteur pour gérer le score et le temps du jeu.

~Game()

Destructeur de la classe.
Appelé automatiquement à la destruction de l'objet Game.
Il libère les ressources allouées, telles que les textures et le rendu, et ferme proprement la fenêtre SDL.

void handleEvents()

Gère les **événements utilisateurs** dans le jeu :

- Les déplacements du joueur,
- Les clics de souris (par exemple, sur le bouton "Rejouer"),
- La détection de la fermeture de la fenêtre.

void update()

Met à jour la **logique du jeu** :

- Mise à jour des positions des objets (joueur, obstacles),
- Vérification des collisions,
- Mise à jour du score et du chronomètre.

void Drawimage()

Dessine toutes les **images à l'écran** :

- Affiche le fond (background1),
- Affiche le joueur et les obstacles,
- Affiche les éléments du HUD (score, temps).

void render()

Affiche les éléments graphiques à l'écran :

- Actualise l'affichage du rendu avec SDL_RenderPresent(),
- Permet de voir les changements effectués dans le jeu (mouvement, collisions, etc.).

void RunGame()

Fonction principale du jeu, qui contient la **boucle d'exécution** du jeu :

- Tant que Running est vrai, elle appelle :
  - handleEvents() pour traiter les entrées utilisateur,
  - update() pour mettre à jour les positions et la logique du jeu,
  - Drawimage() et render() pour dessiner les éléments graphiques à l'écran.
- Crée une boucle fluide jusqu'à ce que le jeu soit terminé (Game Over).

void checkReplaye()

Vérifie si l'utilisateur a choisi de **rejouer** après la fin de la partie. Si le bouton "Rejouer" est cliqué, le jeu recommence.

bool getover()

Retourne **l'état du jeu** (si le jeu est terminé ou non).
Si overG == true, la partie est terminée et le message "Game Over" est affiché.

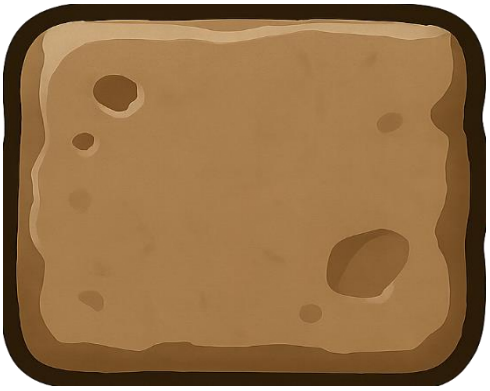
void clean()

Libère toutes les ressources utilisées par le jeu :

- Détruit les textures,
- Ferme le renderer et la fenêtre,
- Appelle SDL_Quit() pour fermer SDL proprement.

Classe Pierre :

La classe pierre représente les obstacles (pierre) dans le jeu. Ces obstacles sont positionnés à différents endroits sur la route et doivent être évités par le joueur.



L'image de pierre.

Les attributs de la classe :

- SDL_Rect Pierre1, Pierre2 :
  Deux rectangles SDL représentant les positions et tailles des deux pierres à l'écran. Ces pierres sont les obstacles que le joueur doit éviter.
- SDL_Texture* imgPierre :
  Texture représentant l'image de la pierre. C'est ce qui sera affiché à l'écran pour représenter l'obstacle.
- int cmp = 0 :
  Un compteur utilisé pour gérer des éléments comme la vitesse de déplacement ou l'apparition des pierres. Cela peut aussi être utilisé pour limiter l'apparition des pierres à des intervalles réguliers.

Les méthodes de la classe :

pierre(SDL_Rect, SDL_Rect)

Constructeur de la classe.
Ce constructeur initialise les deux pierres avec les positions et tailles définies par les deux SDL_Rect passés en paramètres.

~pierre()

Destructeur de la classe.
Libère les ressources allouées, comme la texture des pierres.

void init(SDL_Renderer ren)*

Initialise la texture de la pierre à partir du moteur de rendu SDL (ren). Cette fonction charge l'image de la pierre pour pouvoir l'afficher à l'écran.

void initXpos()

Initialise la **position horizontale** des pierres. Elle peut être utilisée pour ajuster la position des pierres à chaque niveau ou réinitialiser la position après une collision.

void Update()

Met à jour la **position** des pierres sur l'écran, en déplaçant les pierres à une certaine vitesse pour simuler le mouvement de la route.

bool checkcollisse(const SDL_Rect& rect)

Vérifie si la pierre entre en **collision** avec un autre objet (par exemple, le joueur). Cela compare la position de la pierre avec celle d'un autre rectangle, comme le rectangle du joueur.

void render(SDL_Renderer ren)*

Affiche la **pierre** à l'écran. Utilise le renderer pour afficher la texture de la pierre aux positions définies par Pierre1 et Pierre2.

void clean()

Libère toutes les ressources utilisées par la classe, comme la texture des pierres.

Classe Player :

La classe Player représente le cycliste contrôlé par le joueur. Elle gère l'affichage, les mouvements et l'animation du personnage sur l'écran.

Attributs de la classe :

- SDL_Texture* player1 :
  Texture du joueur sous une première apparence (image 1 du cycliste).
- SDL_Texture* player2 :
  Deuxième texture du joueur (image alternative, pour animation ou effet de mouvement).
- SDL_Texture* switchP :
  Pointeur temporaire pour basculer dynamiquement entre player1 et player2, afin d'animer le joueur (ex : pédalage en mouvement).
- SDL_Rect dst :
  Rectangle définissant la position et la taille du joueur à l'écran. Ce rectangle est utilisé pour l'affichage et pour les collisions.

Méthodes de la classe :

Player(int x, int y, int w, int h)

Constructeur
Initialise la position (x, y) et la taille (w, h) du joueur à l'écran via dst.

void init(SDL_Renderer ren)*

Charge les textures du joueur (player1 et player2) à l'aide du renderer SDL.
Cette fonction est appelée au début pour préparer l'affichage.

void render(SDL_Renderer ren)*

Affiche le joueur à l'écran en utilisant la texture active (switchP) et la position définie dans dst.

void Update(int click)

Met à jour la **position** du joueur en fonction de l'entrée utilisateur (click).
Permet de déplacer le joueur à gauche ou à droite selon l'action (clic, touche, etc.).

SDL_Rect& getRectP()

Renvoie une **référence** au rectangle du joueur (dst) pour pouvoir le tester dans les collisions ou autres calculs.

void switchPlayer()

Change la texture affichée (switchP) entre player1 et player2 pour créer une **animation** fluide du joueur (ex : simulation de pédalage).

void cleanP()

Libère les **textures** utilisées par le joueur. Cette fonction est appelée lors de la fermeture du jeu pour éviter les fuites mémoire.

Classe Conteur :

La classe Conteur représente un **chronomètre visuel** affiché pendant la partie, qui indique le temps écoulé ou restant. Elle utilise la bibliothèque **SDL_ttf** pour afficher du texte (le temps) à l'écran.

◈ Attributs de la classe :

- SDL_Texture* texture :
  Texture générée à partir du texte du compteur (affiché à l'écran).
- SDL_Surface* surface :
  Surface intermédiaire utilisée pour créer la texture à partir du texte formaté.
- SDL_Rect rect :
  Rectangle de base qui définit la position et taille du compteur.
- TTF_Font* font :
  Police de caractère utilisée pour dessiner le texte du compteur (ex: Arial, Tahoma...).
- char texte[16] :
  Tableau de caractères qui contient la chaîne de texte affichée (le temps au format "mm:ss" ou "ss").
- SDL_Color couleur :
  Couleur du texte affiché (ex : blanc, rouge...).
- SDL_Rect dst :
  Rectangle d'affichage final à l'écran, où la texture du texte sera dessinée.

◈ Méthodes de la classe :

Conteur(SDL_Rect Conteur, const char police)*

Constructeur
Initialise le compteur avec une position (Conteur) et une police de caractères (police). Charge la police et prépare la zone de dessin.

void formater_temps(Uint32 ms)

Convertit le temps en millisecondes (ms) en un format texte (texte[]) lisible :
Puis met à jour la texture à partir de ce texte.

void render(SDL_Renderer ren)*

Affiche la texture du texte (le temps) à l'écran à la position définie par dst, en utilisant le renderer.

void clean()

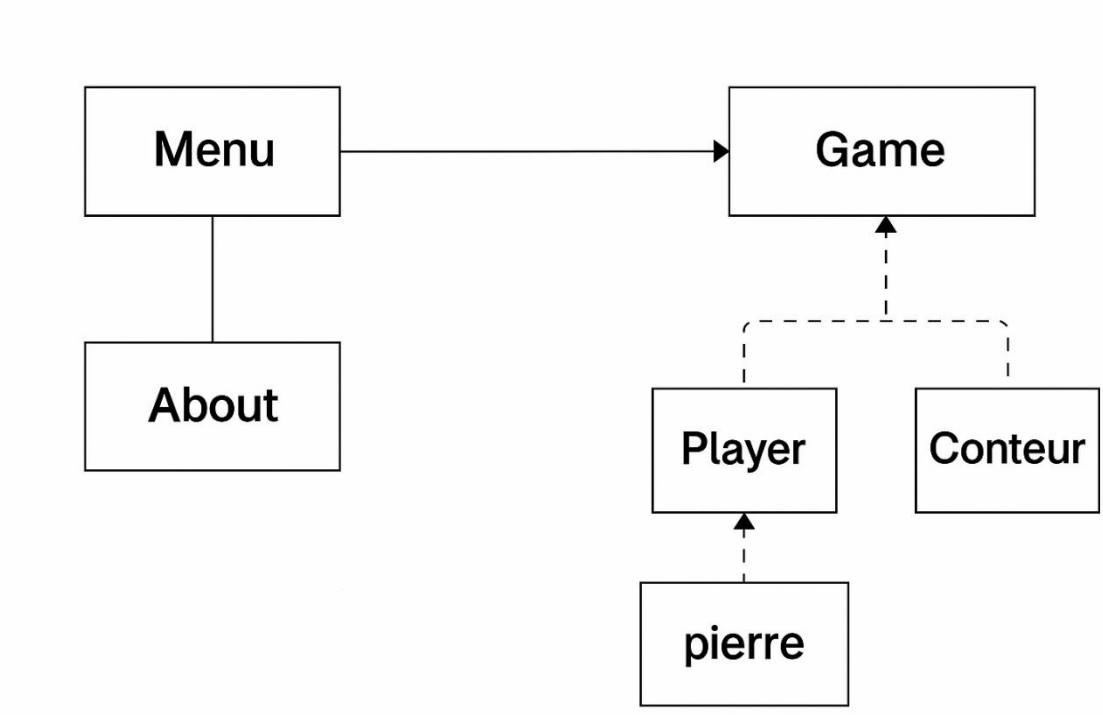
Lière les ressources allouées : police (font), surface et texture. À appeler en fin de jeu.

~Conteur()

Destructeur
Fait appel à clean() pour s'assurer que toutes les ressources sont bien libérées à la destruction de l'objet.

3.2. les relation entre les classe :

Schéma UML



Association : Menu → Game, Menu → About

Agrégation interne : Game → Player, Game → pierre.

Agrégation externe :Game → Conteur.

4.Fonctionnement du jeu:

Logique de jeux :

Le jeu **Cyclist Dodger** est un jeu de type *runner* dans lequel le joueur contrôle un cycliste qui doit éviter des obstacles (pierres) apparaissant sur la route

Bibliothèques utilisées:

Pour développer ce jeu, plusieurs bibliothèques SDL ont été utilisées :

- **SDL2** (*Simple DirectMedia Layer*) :
  Fournit les fonctions de base pour créer la fenêtre du jeu, gérer le rendu graphique, les événements clavier/souris, etc.
- **SDL_image** :
  Permet de charger des images (PNG, JPG) utilisées pour les textures des boutons, du joueur, des obstacles et du fond.
- **SDL_ttf** :
  Utilisée pour afficher du texte (notamment le chronomètre), en important et gérant des polices TrueType (TTF).

4.Défisrencontrés:

Difficulté dans la liaison entre les classes (Menu, Game, About)

Il y avait des difficultés à **organiser la transition entre les classes**, notamment pour passer d'un menu à une autre partie du jeu (jeu ou informations).

Solution :

- Utilisation d'une **variable d'état (numero)** dans Menu pour déterminer quelle classe activer.
- Ajout de conditions pour lancer la bonne action (ex. : si numero == 1, lancer Game).
- Structuration claire de chaque classe avec des méthodes comme RunMenu(), RunGame(), etc.

7. Conclusion :

- Ce projet démontre une bonne maîtrise des bases de la **programmation orientée objet** en C++ tout en utilisant la bibliothèque **SDL2** pour la création d'une application interactive.