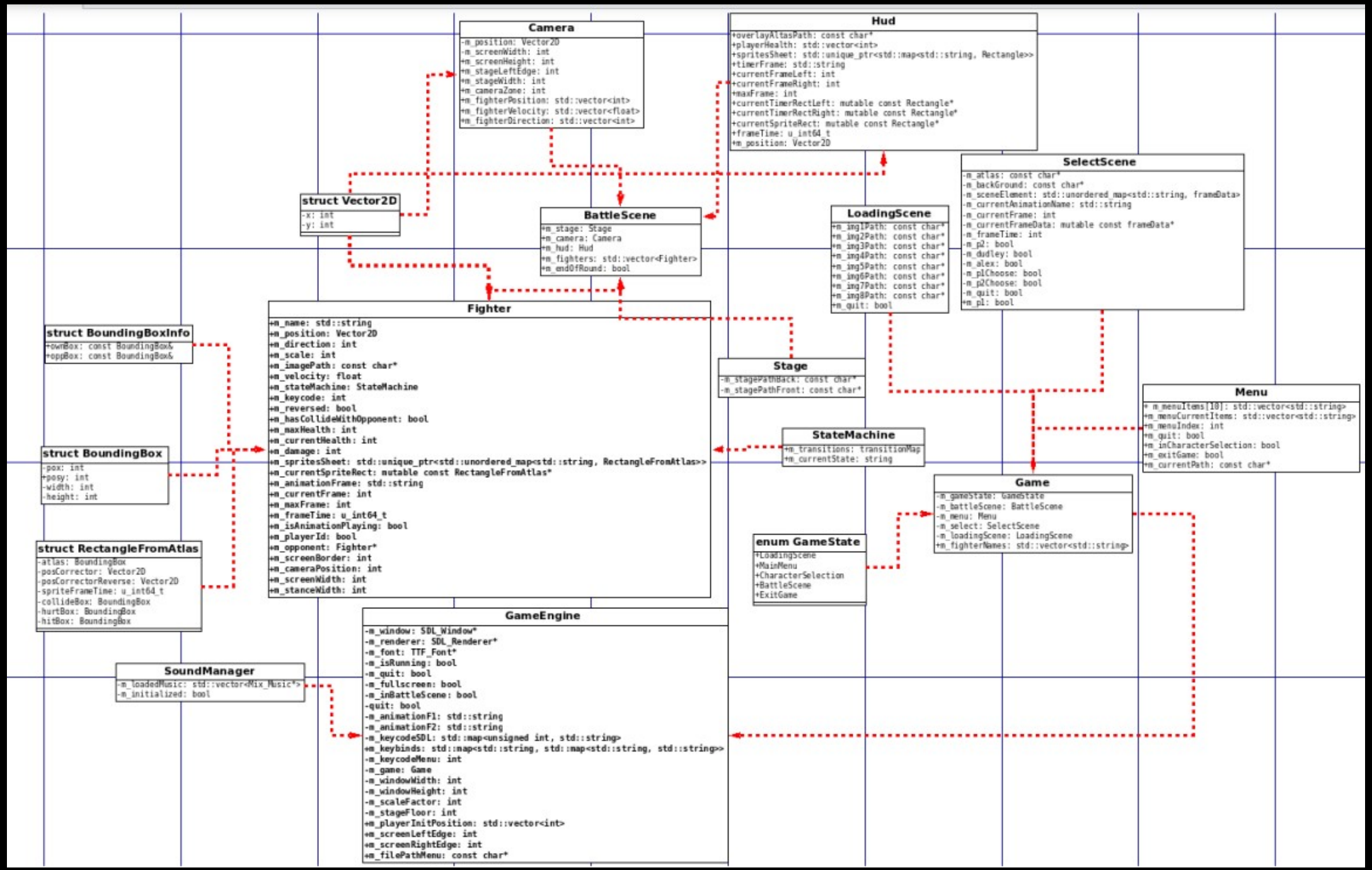
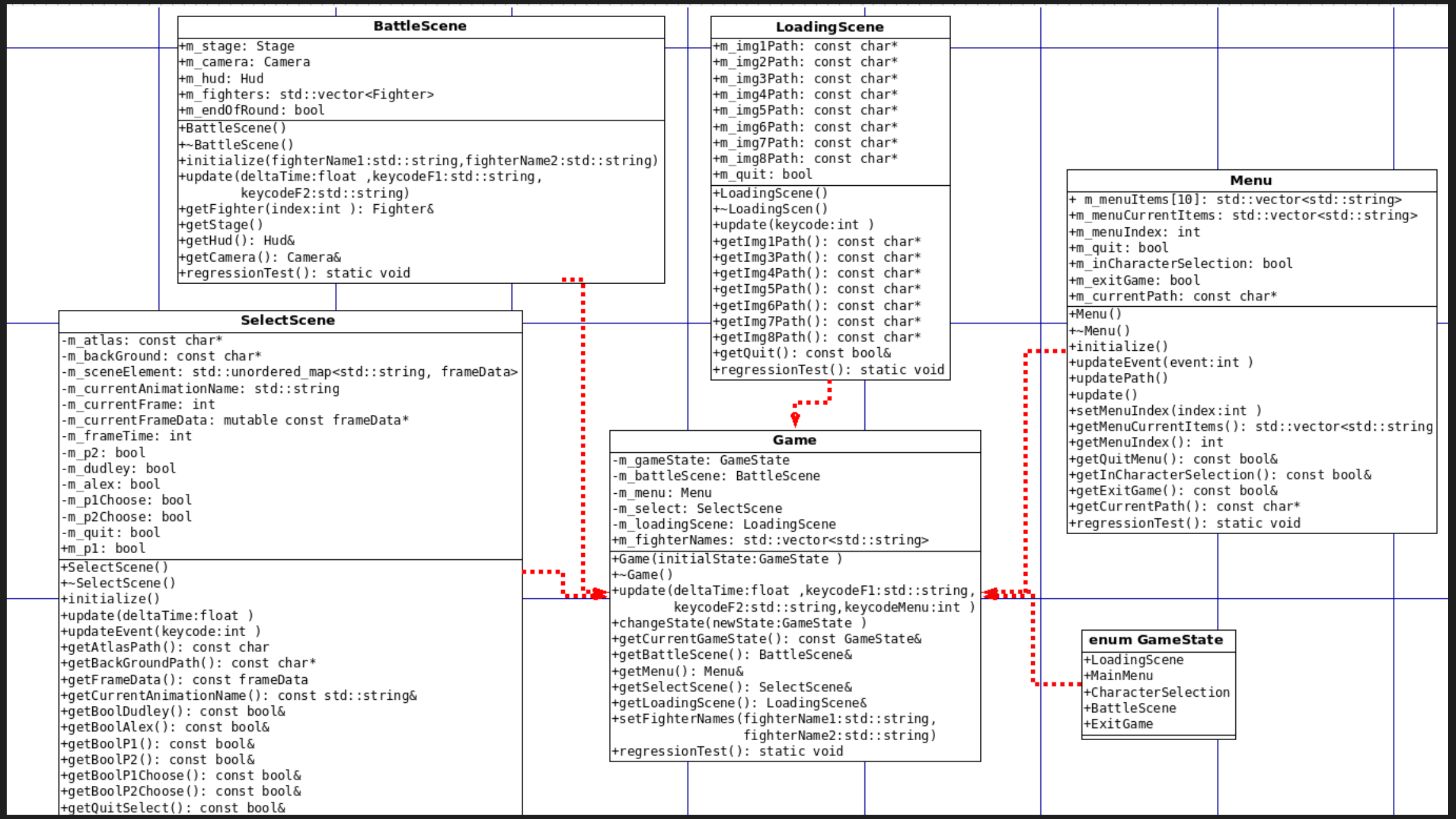
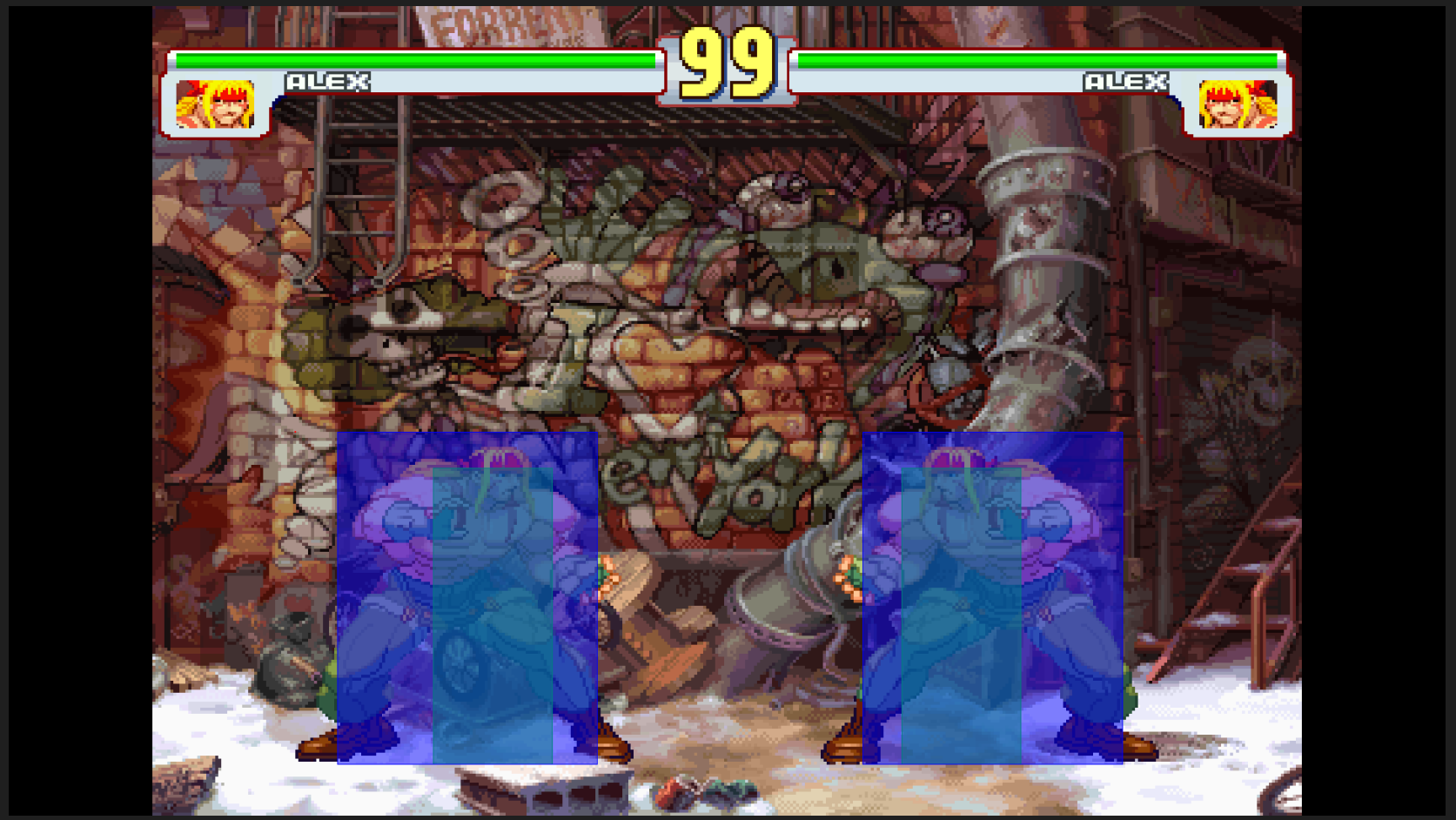
Alexandre Delort 12309795
Gaspard Retter 12101523
Tanguy Vallot 12002626

Kastagne

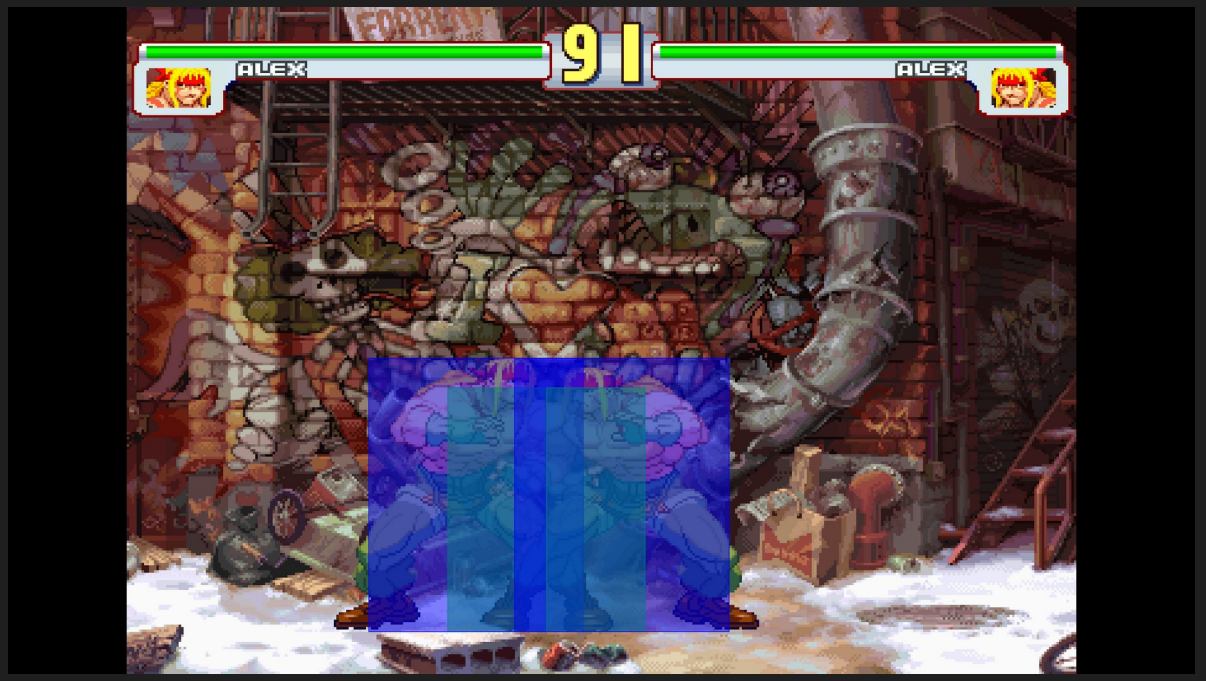

Camera
-m_position: Vector2D
-m_screenWidth: int
+m_screenHeight: int
+m_stageLeftEdge: int
+m_stageWidth: int
+m_cameraZone: int
+m_fighterPosition: std::vector<int>
+m_fighterVelocity: std::vector<float>
+m_fighterDirection: std::vector<int>
Hud
+overlayAltasPath: const char*
+playerHealth: std::vector<int>
+spritesSheet: std::unique_ptr<std::map<std::string, Rectangle>>
+timerFrame: std::string
+currentFrameLeft: int
+currentFrameRight: int
+maxFrame: int
+currentTimerRectLeft: mutable const Rectangle*
+currentTimerRectRight: mutable const Rectangle*
+currentSpriteRect: mutable const Rectangle*
+frameTime: u_int64_t
+m_position: Vector2D
SelectScene
-m_atlas: const char*
-m_backGround: const char*
-m_sceneElement: std::unordered_map<std::string, frameData>
-m_currentAnimationName: std::string
-m_currentFrame: int
-m_currentFrameData: mutable const frameData*
-m_frameTime: int
-m_p2: bool
-m_dudley: bool
-m_alex: bool
-m_p1Choose: bool
-m_p2Choose: bool
-m_quit: bool
+m_p1: bool
struct Vector2D
-x: int
-y: int
BattleScene
+m_stage: Stage
+m_camera: Camera
+m_hud: Hud
+m_fighters: std::vector<Fighter>
+m_endOfRound: bool
LoadingScene
+m_img1Path: const char*
+m_img2Path: const char*
+m_img3Path: const char*
+m_img4Path: const char*
+m_img5Path: const char*
+m_img6Path: const char*
+m_img7Path: const char*
+m_img8Path: const char*
+m_quit: bool
Fighter
+m_name: std::string
+m_position: Vector2D
+m_direction: int
+m_scale: int
+m_imagePath: const char*
+m_velocity: float
+m_stateMachine: StateMachine
+m_keycode: int
+m_reversed: bool
+m_hasCollideWithOpponent: bool
+m_maxHealth: int
+m_currentHealth: int
+m_damage: int
+m_spritesSheet: std::unique_ptr<std::unordered_map<std::string, RectangleFromAtlas>>
+m_currentSpriteRect: mutable const RectangleFromAtlas*
+m_animationFrame: std::string
+m_currentFrame: int
+m_maxFrame: int
+m_frameTime: u_int64_t
+m_isAnimationPlaying: bool
+m_playerId: bool
+m_opponent: Fighter*
+m_screenBorder: int
+m_cameraPosition: int
+m_screenWidth: int
+m_stanceWidth: int
struct BoundingBoxInfo
+ownBox: const BoundingBox&
+oppBox: const BoundingBox&
Stage
-m_stagePathBack: const char*
-m_stagePathFront: const char*
Menu
+ m_menuItems[10]: std::vector<std::string>
+m_menuCurrentItems: std::vector<std::string>
+m_menuIndex: int
+m_quit: bool
+m_inCharacterSelection: bool
+m_exitGame: bool
+m_currentPath: const char*
struct BoundingBox
-pox: int
+posy: int
-width: int
-height: int
StateMachine
+m_transitions: transitionMap
+m_currentState: string
Game
-m_gameState: GameState
-m_battleScene: BattleScene
-m_menu: Menu
-m_select: SelectScene
-m_loadingScene: LoadingScene
+m_fighterNames: std::vector<std::string>
struct RectangleFromAtlas
-atlas: BoundingBox
-posCorrector: Vector2D
-posCorrectorReverse: Vector2D
-spriteFrameTime: u_int64_t
-collideBox: BoundingBox
-hurtBox: BoundingBox
-hitBox: BoundingBox
enum GameState
+LoadingScene
+MainMenu
+CharacterSelection
+BattleScene
+ExitGame
GameEngine
-m_window: SDL_Window*
-m_renderer: SDL_Renderer*
-m_font: TTF_Font*
-m_isRunning: bool
-m_quit: bool
-m_fullscreen: bool
-m_inBattleScene: bool
-quit: bool
-m_animationF1: std::string
-m_animationF2: std::string
-m_keycodeSDL: std::map<unsigned int, std::string>
+m_keybinds: std::map<std::string, std::map<std::string, std::string>>
-m_keycodeMenu: int
-m_game: Game
-m_windowWidth: int
-m_windowHeight: int
-m_scaleFactor: int
-m_stageFloor: int
+m_playerInitPosition: std::vector<int>
+m_screenLeftEdge: int
+m_screenRightEdge: int
+m_filePathMenu: const char*
SoundManager
-m_loadedMusic: std::vector<Mix_Music*>
-m_initialized: bool

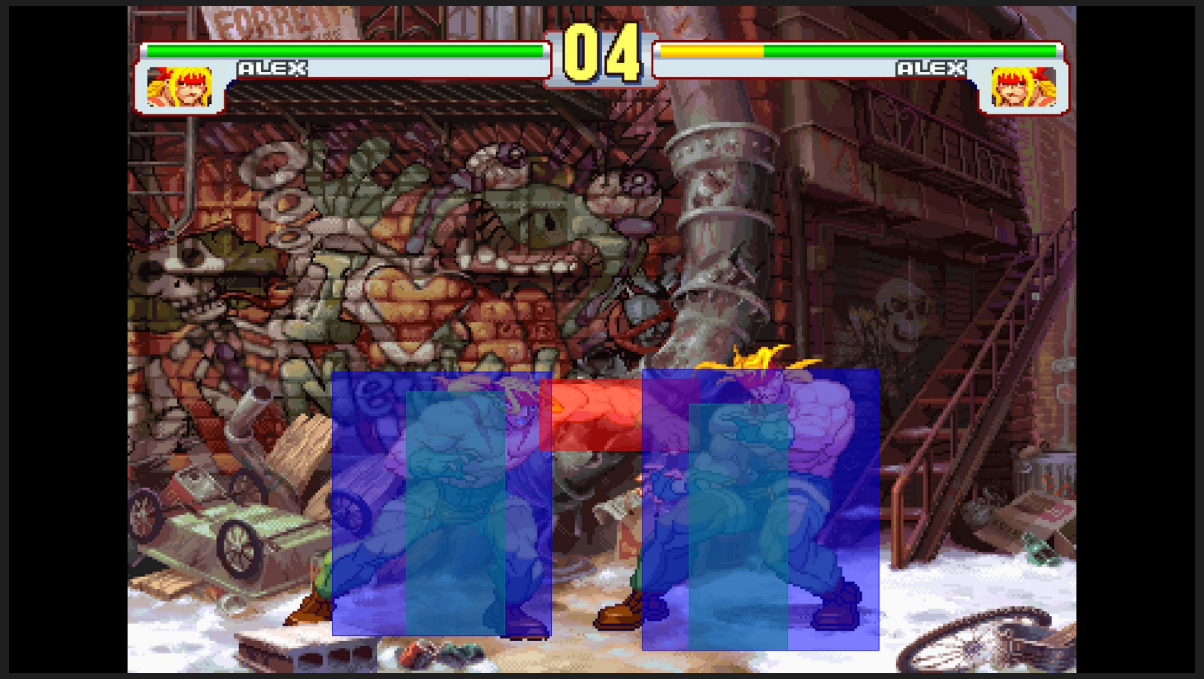

BattleScene
+m_stage: Stage
+m_camera: Camera
+m_hud: Hud
+m_fighters: std::vector<Fighter>
+m_endOfRound: bool
+BattleScene()
+~BattleScene()
+initialize(fighterName1:std::string,fighterName2:std::string)
+update(deltaTime:float ,keycodeF1:std::string,
keycodeF2:std::string)
+getFighter(index:int): Fighter&
+getStage()
+getHud(): Hud&
+getCamera(): Camera&
+regressionTest(): static void
LoadingScene
+m_img1Path: const char*
+m_img2Path: const char*
+m_img3Path: const char*
+m_img4Path: const char*
+m_img5Path: const char*
+m_img6Path: const char*
+m_img7Path: const char*
+m_img8Path: const char*
+m_quit: bool
+LoadingScene()
+~LoadingScen()
+update(keycode:int)
+getImg1Path(): const char*
+getImg3Path(): const char*
+getImg4Path(): const char*
+getImg5Path(): const char*
+getImg6Path(): const char*
+getImg7Path(): const char*
+getImg8Path(): const char*
+getQuit(): const bool&
+regressionTest(): static void
Menu
+ m_menuItems[10]: std::vector<std::string>
+m_menuCurrentItems: std::vector<std::string>
+m_menuIndex: int
+m_quit: bool
+m_inCharacterSelection: bool
+m_exitGame: bool
+m_currentPath: const char*
+Menu()
+~Menu()
+initialize()
+updateEvent(event:int)
+updatePath()
+update()
+setMenuIndex(index:int)
+getMenuCurrentItems(): std::vector<std::string
+getMenuIndex(): int
+getQuitMenu(): const bool&
+getInCharacterSelection(): const bool&
+getExitGame(): const bool&
+getCurrentPath(): const char*
+regressionTest(): static void
SelectScene
-m_atlas: const char*
-m_backGround: const char*
-m_sceneElement: std::unordered_map<std::string, frameData>
-m_currentAnimationName: std::string
-m_currentFrame: int
-m_currentFrameData: mutable const frameData*
-m_frameTime: int
-m_p2: bool
-m_dudley: bool
-m_alex: bool
-m_p1Choose: bool
-m_p2Choose: bool
-m_quit: bool
+m_p1: bool
+SelectScene()
+~SelectScene()
+initialize()
+update(deltaTime:float)
+updateEvent(keycode:int)
+getAtlasPath(): const char
+getBackGroundPath(): const char*
+getFrameData(): const frameData
+getCurrentAnimationName(): const std::string&
+getBoolDudley(): const bool&
+getBoolAlex(): const bool&
+getBoolP1(): const bool&
+getBoolP2(): const bool&
+getBoolP1Choose(): const bool&
+getBoolP2Choose(): const bool&
+getQuitSelect(): const bool&
Game
-m_gameState: GameState
-m_battleScene: BattleScene
-m_menu: Menu
-m_select: SelectScene
-m_loadingScene: LoadingScene
+m_fighterNames: std::vector<std::string>
+Game(initialState:GameState)
+~Game()
+update(deltaTime:float ,keycodeF1:std::string,
keycodeF2:std::string,keycodeMenu:int)
+changeState(newState:GameState)
+getCurrentGameState(): const GameState&
+getBattleScene(): BattleScene&
+getMenu(): Menu&
+getSelectScene(): SelectScene&
+getLoadingScene(): LoadingScene&
+setFighterNames(fighterName1:std::string,
fighterName2:std::string)
+regressionTest(): static void
enum GameState
+LoadingScene
+MainMenu
+CharacterSelection
+BattleScene
+ExitGame

```
{ "StandingFarHK-0"  , {{1500, 3499, 209, 150}, {-23, 38}, {-73, 38}, 20 , {69 , 50, 40, 100}, {42, 43, 76, 103}} },
{ "StandingFarHK-1"  , {{1710, 3499, 209, 150}, {-23, 38}, {-73, 38}, 40 , {53 , 57, 40, 92 }, {21, 47, 87, 103}} },
{ "StandingFarHK-2"  , {{1920, 3499, 209, 150}, {-23, 38}, {-73, 38}, 40 , {23 , 58, 40, 92 }, {0 , 45, 89, 105}} },
{ "StandingFarHK-3"  , {{2130, 3499, 209, 150}, {-23, 38}, {-73, 38}, 50 , {23 , 58, 40, 92 }, {0 , 43, 89, 107}} },
{ "StandingFarHK-4"  , {{2340, 3499, 209, 150}, {-23, 38}, {-73, 38}, 70 , {23 , 46, 40, 104}, {2 , 34, 83, 116}} },
{ "StandingFarHK-5"  , {{2550, 3499, 209, 150}, {-23, 38}, {-73, 38}, 70 , {29 , 43, 40, 107}, {19, 27, 73, 123}} },
{ "StandingFarHK-6"  , {{2760, 3499, 209, 150}, {-23, 38}, {-73, 38}, 70 , {72 , 34, 43, 116}, {63, 19, 57, 131}, {118, 43, 91, 40}} },
{ "StandingFarHK-7"  , {{1500, 3650, 209, 150}, {-23, 38}, {-73, 38}, 140, {74 , 34, 43, 116}, {63, 19, 57, 131}, {118, 43, 91, 40}} },
{ "StandingFarHK-8"  , {{1710, 3650, 209, 150}, {-23, 38}, {-73, 38}, 70 , {74 , 43, 44, 106}, {62, 19, 63, 131}, {118, 43, 91, 40}} },
{ "StandingFarHK-9"  , {{1920, 3650, 209, 150}, {-23, 38}, {-73, 38}, 70 , {74 , 43, 44, 106}, {61, 31, 73, 119}, {118, 43, 91, 40}} },
{ "StandingFarHK-10" , {{2130, 3650, 209, 150}, {-23, 38}, {-73, 38}, 70 , {88 , 53, 44, 97 }, {64, 41, 78, 109}} },
{ "StandingFarHK-11" , {{2340, 3650, 209, 150}, {-23, 38}, {-73, 38}, 100, {107, 70, 40, 80 }, {66, 61, 90, 89 }} },
{ "StandingFarHK-12" , {{2550, 3650, 209, 150}, {-23, 38}, {-73, 38}, 70 , {107, 55, 40, 95 }, {74, 44, 82, 106}} },
{ "StandingFarHK-13" , {{2760, 3650, 209, 150}, {-23, 38}, {-73, 38}, 20 , {107, 49, 40, 101}, {80, 37, 76, 113}} },
```



```cpp
// Function refactored
bool Fighter::checkCollision(const BoundingBoxInfo& boxInfo) const {
    BoundingBox ownAtlas = m_currentSpriteRect->atlas;
    BoundingBox oppAtlas = m_opponent->getSpriteRect().atlas;

    int ownBoxPositionX, oppBoxPositionX;

    if(m_reversed) {
        ownBoxPositionX =
            m_position.x + (ownAtlas.width - boxInfo.ownBox.posx)*m_scale;
        oppBoxPositionX =
            m_opponent->m_position.x + boxInfo.oppBox.posx*m_scale;
    } else {
        ownBoxPositionX = m_position.x + boxInfo.ownBox.posx*m_scale;
        oppBoxPositionX =
            m_opponent->m_position.x
            + (oppAtlas.width - boxInfo.oppBox.posx)*m_scale;
    }

    bool horizontalCollision = (m_reversed ?
        ownBoxPositionX - boxInfo.ownBox.width*m_scale
        < oppBoxPositionX + boxInfo.oppBox.width*m_scale :

        ownBoxPositionX + boxInfo.ownBox.width*m_scale
        > oppBoxPositionX - boxInfo.oppBox.width*m_scale);

    bool verticalCollision =
        m_position.y + ownAtlas.height*m_scale
        > m_opponent->m_position.y + boxInfo.oppBox.posy*m_scale
        &&
        m_position.y + boxInfo.ownBox.posy*m_scale
        < m_opponent->m_position.y + oppAtlas.height*m_scale;

    return horizontalCollision && verticalCollision;
}
```





```cpp
const double dt = 16.0;
double accumulator = 0.0;
auto startTime = std::chrono::high_resolution_clock::now();

while (getQuitStatus()) {
    auto currentTime = std::chrono::high_resolution_clock::now();
    double frameTime =
        std::chrono::duration_cast<std::chrono::milliseconds>
            (currentTime - startTime).count();

    startTime = currentTime;

    accumulator += std::min(frameTime, 160.0);
    while(accumulator >= dt) {
        // Update area
        accumulator -= dt;
    }

    // Render area

    // Waiting time calculation
    double sleepTime =
        dt - std::chrono::duration_cast<std::chrono::milliseconds>(
                std::chrono::high_resolution_clock::now() - startTime
            ).count();

    if (sleepTime > 0.0) {
        std::this_thread::sleep_for(
            std::chrono::milliseconds(static_cast<long long>(sleepTime))
            );
    }

    // We release the OS of our program
    std::this_thread::sleep_for(std::chrono::milliseconds(1));
}
```

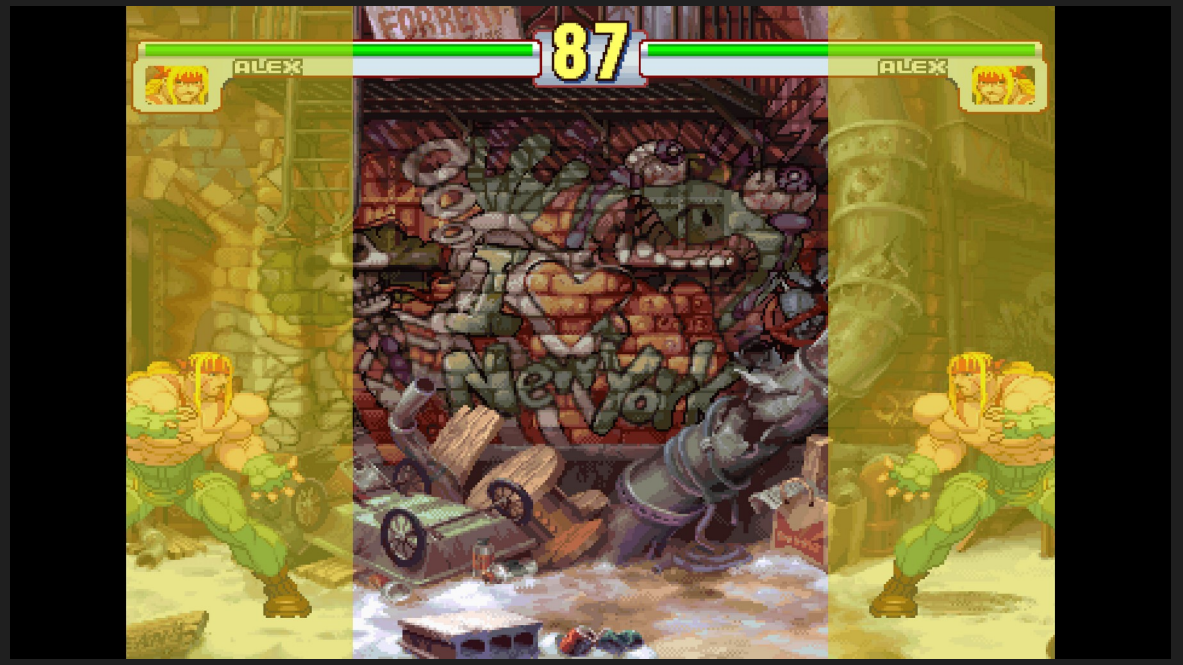
```cpp
// Method to update the game state based on the elapsed time and input
void Game::update(float deltaTime, std::string keycodeF1, std::string keycodeF2, int keycodeMenu) {
    // Update based on the current game state
    switch (gameState) {
        case GameState::LoadingScene:
            loadingScene.update(keycodeMenu);
            break;
        case GameState::MainMenu:
            menu.updateEvent(keycodeMenu);
            break;
        case GameState::CharacterSelection:
            select.updateEvent(keycodeMenu);
            break;
        case GameState::BattleScene:
            // Mettre à jour la scène de combat
            battleScene.update(deltaTime, keycodeF1,keycodeF2);
            break;
        case GameState::ExitGame:
            break;
    }
}
```

```cpp
void update(float deltaTime, bool playerId, std::string keycode);
void setAnimation(std::string keycode, bool playerId);
bool isAnimationInProgress();
```

```cpp
void handleEvents();
void handleKeybinds(const std::string& playerName, SDL_Event& event);
```

```cpp
std::string returnTouchAction(SDL_Event& event,const std::string& name ,const std::string& action);
SDL_Keycode returnValueSDL(const std::string& action);

void setAnimation(const std::string& playerName, const std::string& keycode);
void resetAnimation(const std::string& playerName, const std::string& keycode);
```

```cpp
// Updates the camera based on fighter positions and other parameters
void Camera::update(float deltaTime) {
    // Camera limits
    const int leftCameraLimit = -m_stageLeftEdge;
    const int rightCameraLimit =
        m_stageWidth + m_stageLeftEdge - m_screenWidth - m_cameraZone;

    // Current camera zone limits
    const int currentLeftZoneLimit  =
        m_position.x + m_cameraZone + m_stageLeftEdge;
    const int currentRightZoneLimit =
        m_position.x + m_screenWidth - m_cameraZone - m_stageLeftEdge;

    // Neutral zone
    const int neutralZone =
        m_screenWidth - (m_cameraZone << 1) - (m_stageLeftEdge << 1);

    // Calculate minimum and maximum fighter positions
    const int minPositionX =
        std::min(m_fighterPosition[0], m_fighterPosition[1]);
    const int maxPositionX =
        std::max(m_fighterPosition[0], m_fighterPosition[1]);

    // Updating the camera based on fighter position
    if ((maxPositionX - minPositionX) > neutralZone) {
        const int middlePosition = ((maxPositionX - minPositionX) >> 1);
        m_position.x = minPositionX + middlePosition - (m_screenWidth >> 1);
    } else {
        // If fighter exceeds the current camera zone, move the camera
        for (int i = 0; i < 2; ++i) {
            if ((m_fighterPosition[i] < currentLeftZoneLimit &&
                (m_fighterVelocity[i] * m_fighterDirection[i]) < 0) ||
                (m_fighterPosition[i] > currentRightZoneLimit &&
                (m_fighterVelocity[i] * m_fighterDirection[i]) > 0)) {
                m_position.x +=
                    m_fighterVelocity[i] * m_fighterDirection[i] * deltaTime;
            }
        }
    }
    // Ensure camera does not exceed its limits
    if (m_position.x < leftCameraLimit)
        m_position.x = leftCameraLimit;
    if (m_position.x > rightCameraLimit)
        m_position.x = rightCameraLimit;
}
```




Kastagne