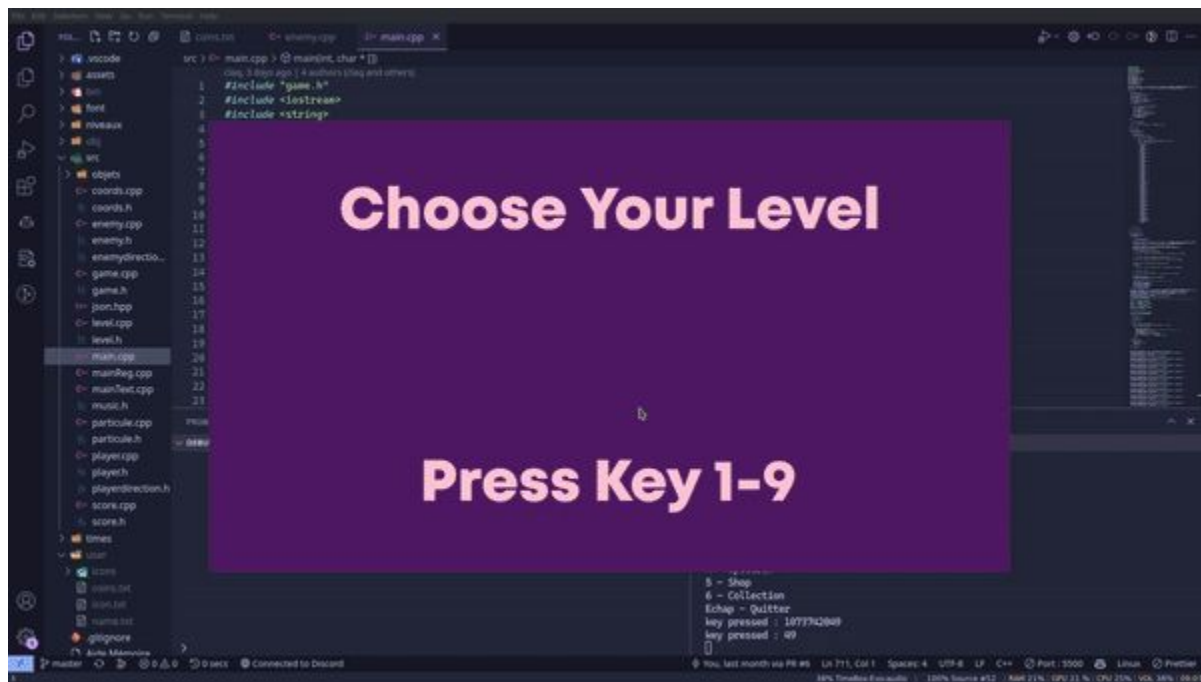


Miso Mania

Par Jules Ginhac, Romain Rochebloine et Mathieu Ponton

Miso Mania



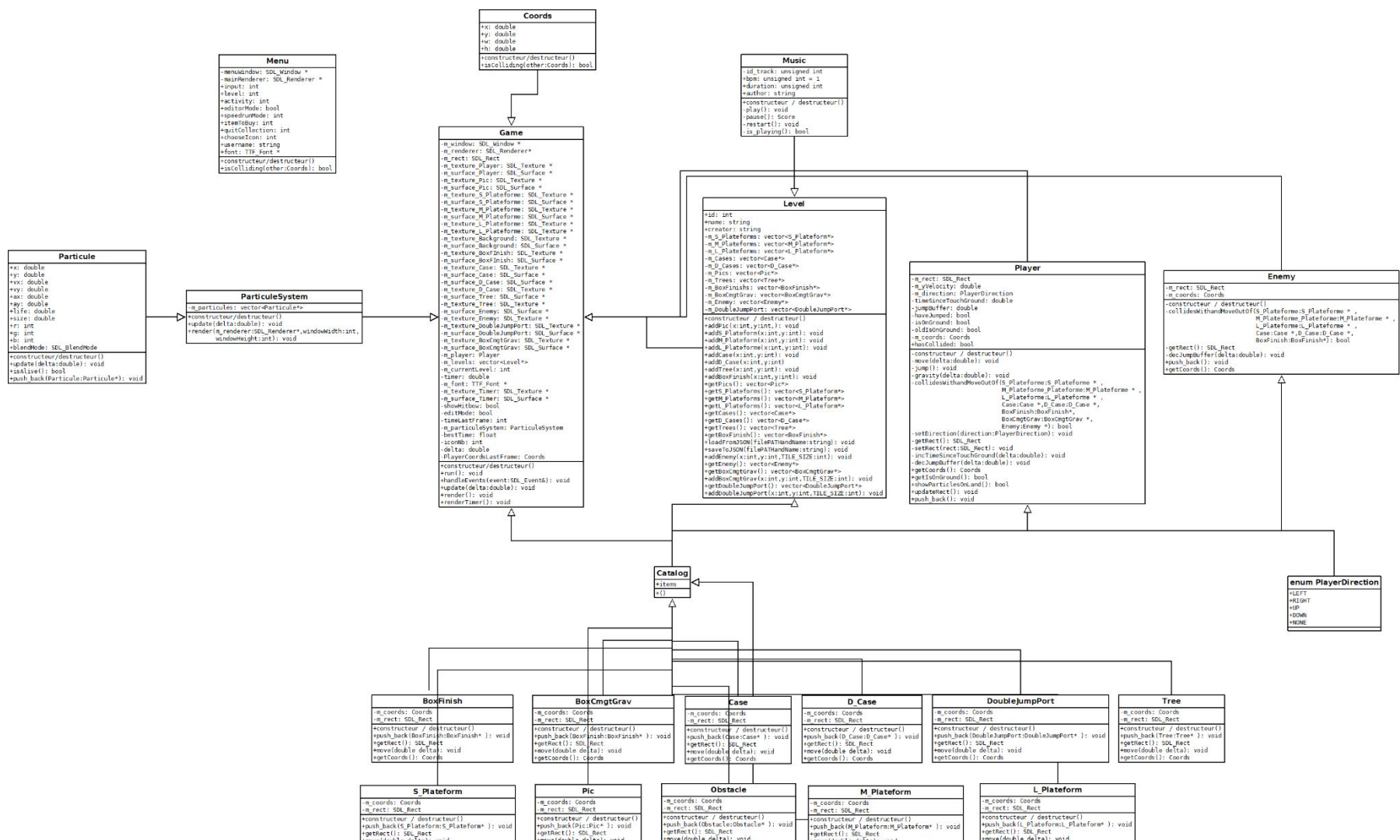
Miso Mania

Le Jeu

- **Type : Jeu de plateforme à niveaux courts**
- **Objectif : Terminer le niveau le plus rapidement possible**
- **Graphismes : 2D (vue de côté)**
- **Textures : “Pixel Art”, style 64 bits**
- **Musique : Douce, style “lofi”**

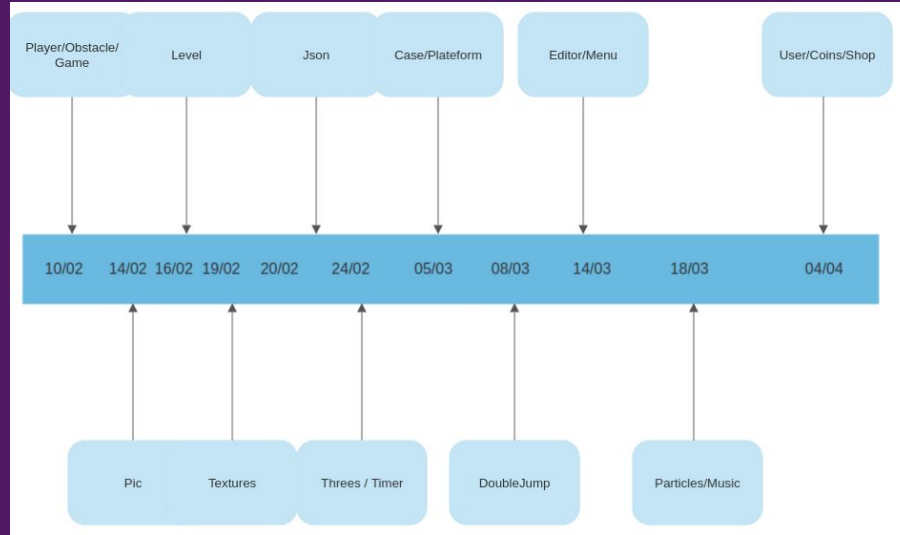
Particularités

- **Créateur de niveaux**
- **Classements**



Miso Mania

Timeline de l'implémentation des classes



Miso Mania

Éléments intéressants du code

Miso Mania

Classe Player

Fonctions indispensables pour pouvoir jouer

- **move, stopMove, moveTo**
- **jump, doubleJump**
- **gravity, stopGravity**

Fonctions collisions

- **collidesWith, moveOutOfCoords, moveOutOf**

Getters et Setters

- **setDirection, setJumpBuffer, setRect, etc.**
- **getDirection, getRect, getGravity, etc.**

Miso Mania

Architecture

```
#include "player.h"
#include <iostream>
#include <assert.h>
Player::Player(): hasCollided(false), m_coords(1, 23, 1.44, 1.44),
h_rect(100, 600, 30, 30), m_velocity(0),
m_direction(PlayerDirection::NONE), timeSinceTouchGround(0),
jumpBuffer(0), isOnGround(false), oldIsOnGround(false) {}

Player::~Player() {}

void Player::move(double delta) {
    if (m_direction == PlayerDirection::LEFT) {
        m_coords.x -= 5 * delta;
    } else if (m_direction == PlayerDirection::RIGHT) {
        m_coords.x += 5 * delta;
    }
}

void Player::jump() {
    if ((haveJumped && timeSinceTouchGround < 0.2) {
        m_velocity = -13;
        haveJumped = true;
        isOnGround = false;
    } else {
        jumpBuffer = 0.1;
    }
}

void Player::gravity(double delta) {
    m_velocity += 25 * delta;
    m_coords.y += m_velocity * delta;
}

bool Player::collidesWith(Obstacle *obstacle) {
    return m_coords.isColliding(obstacle->getCoords());
}

bool Player::collidesWith(S_Plateform *S_Plateform) {
    return m_coords.isColliding(S_Plateform->getCoords());
}

bool Player::collidesWith(M_Plateform *M_Plateform) {
    return m_coords.isColliding(M_Plateform->getCoords());
}

bool Player::collidesWith(L_Plateform *L_Plateform) {
    return m_coords.isColliding(L_Plateform->getCoords());
}

bool Player::collidesWith(Pic *Pic) {
    return m_coords.isColliding(Pic->getCoords());
}
```

```
bool Player::collidesWith(DoubleJumpPort *DoubleJumpPort){
    return m_coords.isColliding(DoubleJumpPort->getCoords());
}

bool Player::collidesWith(Case *Case) {
    return m_coords.isColliding(Case->getCoords());
}

bool Player::collidesWith(D_Case *D_Case) {
    return m_coords.isColliding(D_Case->getCoords());
}

bool Player::collidesWith(BoxFinish *BoxFinish) {
    return m_coords.isColliding(BoxFinish->getCoords());
}

bool Player::collidesWith(BoxCmgGrav *BoxCmgGrav) {
    return m_coords.isColliding(BoxCmgGrav->getCoords());
}

bool Player::collidesWith(Enemy *Enemy) {
    return m_coords.isColliding(Enemy->getCoords());
}

void Player::moveOutOfCoords(Coords coords){
    double intoTop = m_coords.y + m_coords.h - coords.y;
    double intoBottom = coords.y + coords.h - m_coords.y;
    double intoLeft = m_coords.x + m_coords.w - coords.x;
    double intoRight = coords.x + coords.w - m_coords.x;
    if (intoTop < intoBottom && intoTop < intoLeft && intoTop < intoRight) {
        m_coords.y = intoTop;
        stopGravity();
        haveJumped = false;
        timeSinceTouchGround = 0;
        if (jumpBuffer > 0) {
            jump();
            jumpBuffer = 0;
        }
        isOnGround = true;
        hasCollided = true;
    } else if (intoBottom < intoTop && intoBottom < intoLeft && intoBottom < intoRight) {
        m_coords.y = intoBottom;
        stopGravity();
    } else if (intoLeft < intoTop && intoLeft < intoBottom && intoLeft < intoRight) {
        m_coords.x = intoLeft;
    } else if (intoRight < intoTop && intoRight < intoBottom && intoRight < intoLeft) {
        m_coords.x = intoRight;
    }
}

void Player::moveOutOf(Obstacle *obstacle){
    moveOutOfCoords(obstacle->getCoords());
}
```


Miso Mania

Architecture

```
void Player::moveOutOf(S_Plateform *S_Plateform){
    moveOutOfCoords(S_Plateform->getCoords());
}

void Player::moveOutOf(M_Plateform *M_Plateform){
    moveOutOfCoords(M_Plateform->getCoords());
}

void Player::moveOutOf(L_Plateform *L_Plateform){
    moveOutOfCoords(L_Plateform->getCoords());
}

void Player::moveOutOf(Case *Case){
    moveOutOfCoords(Case->getCoords());
}

void Player::moveOutOf(D_Case *D_Case){
    moveOutOfCoords(D_Case->getCoords());
}

void Player::moveOutOf(BoxCmgtGrav *BoxCmgtGrav){
    moveOutOfCoords(BoxCmgtGrav->getCoords());
}

void Player::moveOutOf(Enemy *Enemy){
    moveOutOfCoords(Enemy->getCoords());
}

void Player::incTimeSinceTouchGround(double delta) {
    timeSinceTouchGround += delta;
}

void Player::decJumpBuffer(double delta) {
    if (jumpBuffer > 0) jumpBuffer -= delta;
}

void Player::stopGravity() {
    m_yVelocity = 0;
}

void Player::stopMove() {
    m_direction = PlayerDirection::NONE;
}

void Player::doubleJump() {
    if (haveJumped) {
        m_yVelocity = -13;
        haveJumped = false;
    }
}

void Player::setDirection(PlayerDirection direction) {
    m_direction = direction;
}
```

```
SDL_Rect Player::getRect() {
    return m_rect;
}

void Player::setRect(SDL_Rect rect) {
    m_rect = rect;
}

double Player::getGravity() {
    return m_yVelocity;
}

void Player::setGravity(double gravity) {
    m_yVelocity = gravity;
}

void Player::setJumpBuffer(double jumpBuffer) {
    this->jumpBuffer = jumpBuffer;
}

void Player::setTimeSinceTouchGround(double timeSinceTouchGround) {
    this->timeSinceTouchGround = timeSinceTouchGround;
}

void Player::moveTo(double x, double y){
    m_coords.x = x;
    m_coords.y = y;
}

void Player::updateRect(){
    m_rect.x = m_coords.x * 40;
    m_rect.y = m_coords.y * 40;
    m_rect.w = m_coords.w * 40;
    m_rect.h = m_coords.h * 40;
}

Coords Player::getCoords() const {
    return m_coords;
}

bool Player::getIsOnGround() const {
    return isOnGround;
}

bool Player::showParticlesOnLand() {
    bool r = !oldIsOnGround && isOnGround;
    if (!hasCollided) {
        isOnGround = false;
    }
    oldIsOnGround = isOnGround;
    return r;
}
```

Miso Mania

Export/import des niveaux en JSON.

- Utilisation de la librairie [nlohmann/json](#) (nécessaire car la création du c++ précède celle du json de 22 ans !).
- Permet de partager les niveaux à ses amis facilement, il suffit d'envoyer un fichier de quelques Kilobytes.
- Permet d'assurer que tous les niveaux créés ont exactement la même nomenclature / format.

Deux fonctions : *loadFromJSON* et *saveToJSON*, toutes deux fonctions membres de la classe *level*.

Miso Mania

Architecture

```
void Level::loadFromJSON(string filename, int TILE_SIZE) {
    cout << "Loading level from " << filename << endl;
    ifstream file(filename);
    json j;
    file >> j;

    id = j["id"];
    name = j["nom du niveau"];
    creator = j["createur"];
    for (auto& element : j["obstacles"]) {--
    for (auto& element : j["pics"]) {
        int x = element["x"];
        int y = element["y"];
        addPic(x, y, TILE_SIZE);
        cout << "Pic: " << x << ", " << y << endl;
    }
    for (auto& element : j["BoxFinish"]) {--
    for (auto& element : j["trees"]) {--
    for (auto& element : j["DoubleJumpPort"]) {--
    for (auto& element : j["Case"]) {--
    for (auto& element : j["D_Case"]) {--
    for (auto& element : j["S_Plateform"]) {--
    for (auto& element : j["M_Plateform"]) {--
    for (auto& element : j["L_Plateform"]) {--
    for (auto& element : j["BoxCmgtGrav"]) {--
    for (auto& element : j["Enemy"]) {--
}
```

```
void Level::saveToJSON(string filename) {
    json j;

    j["id"] = id;
    j["nom du niveau"] = name;
    j["createur"] = creator;
    for (Obstacle* obstacle : m_obstacles) {--
    for (Pic* pic : m_pics) {
        json picJSON;
        Coords picCoords = pic->getCoords();
        picJSON["x"] = picCoords.x;
        picJSON["y"] = picCoords.y;
        j["pics"].push_back(picJSON);
    }
    for (BoxFinish* box : m_BoxFinish) {--
    for (Tree* tree : m_trees) {--
    for (DoubleJumpPort* doublejumpport : m_DoubleJumpPort) {--
    for (Case* Case : m_Case) {--
    for (D_Case* D_Case : m_D_Case) {--
    for (S_Plateform* S_Plateform : m_S_Plateform) {--
    for (M_Plateform* M_Plateform : m_M_Plateform) {--
    for (L_Plateform* L_Plateform : m_L_Plateform) {--
    for (BoxCmgtGrav* BoxCmgtGrav : m_BoxCmgtGrav) {--

    ofstream file(filename);
    file << j;
}
```

Miso Mania

Exemple de fichier JSON généré

```
niveaux > {} level2.json > ...  
You, 3 minutes ago | 2 authors (Jules and others)  
1 {  
2   "id": 2,  
3   "nom du niveau": "level2",  
4   "createur": "Claq",  
5  
6   "BoxFinish": [  
7     {"x": 0.0, "y": 0.0}],  
8  
9   "Case": [  
10    {"x": 24.0, "y": 24.0},  
11    {"x": 28.0, "y": 22.0},  
12    {"x": 33.0, "y": 24.0},  
13    {"x": 35.0, "y": 22.0},  
14    {"x": 40.0, "y": 24.0},  
15    {"x": 19.0, "y": 12.0},  
16    {"x": 17.0, "y": 13.0},  
17    {"x": 21.0, "y": 13.0},  
18    {"x": 23.0, "y": 12.0},  
19    {"x": 39.0, "y": 9.0},  
20    {"x": 15.0, "y": 12.0},  
21    {"x": 0.0, "y": 2.0}],  
22  
23   "D_Case": [  
24     {"x": 10.0, "y": 24.0}],  
25
```

Miso Mania














Données propres à chaque joueur

- Nom d'utilisateur.
- Liste d'icônes possédées par le joueur.
- Icône choisie par le joueur parmi celles qu'il possède.
- Nombre de pièces d'or possédées par le joueur.

Ces 4 informations sont stockées dans le dossier /user. Ce dernier est créé par le makefile au moment de la compilation du jeu. Nous avons ajouté le dossier /user au fichier .gitignore afin de ne pas partager ses données à tout le monde sur Github !

Miso Mania

Architecture

- ▼  icons
 -  0.txt
 -  1.txt
 -  2.txt
 -  3.txt
 -  4.txt
 -  5.txt
 -  6.txt
 -  7.txt
 -  8.txt
 -  coins.txt
 -  icon.txt
 -  name.txt

Miso Mania

Chargement de l'icône du joueur

```
if (iconFile != NULL)
{
    char icon[2];
    fgets(icon, 2, iconFile);
    fclose(iconFile);
    //if null, we load the default icon
    if (icon[0] == '\0')
    {
        m_surface_player = IMG_Load("assets/icons/0.png");
    }
    else{
        char iconPath[20] = "assets/icons/";
        strcat(iconPath, icon);
        strcat(iconPath, ".png");
        m_surface_player = IMG_Load(iconPath);
        iconNb = atoi(icon);
    }
}
else
{
    cout << "icon.txt not found" << endl;
    m_surface_player = IMG_Load("assets/icons/0.png");
}
```

Miso Mania

Types de particules



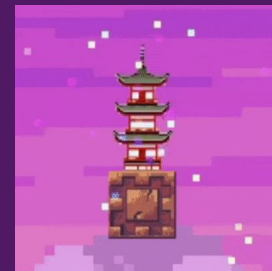
- Blanc transparent
- Position de départ aléatoire en dessous du joueur
- Vitesse de départ, direction, taille, durée de vie aléatoire



- Rose opaque
- Position de départ aléatoire sur l'arbre
- Vitesse et direction constante



- Bleu opaque
- Position de départ au centre
- Direction aléatoire
- Vitesse initiale fixe
- Accélération inverse de la vitesse



- Couleur aléatoire
- Position de départ aléatoire
- Direction et vitesse aléatoire
- Accélération inverse de la vitesse

Miso Mania

Types de particules



Particules derrière le joueur pour les icônes les plus chères.

- Position aléatoire sur le joueur**
- Vitesse nulle**
- Direction de l'accélération aléatoire**

Miso Mania

Système de particules

```
void ParticuleSystem::update(double delta) {  
    for (int i=m_particules.size()-1; i >= 0; i--) {  
        m_particules[i]->update(delta);  
        if (!m_particules[i]->isAlive()) {  
            delete m_particules[i];  
            m_particules.erase(m_particules.begin() + i);  
        }  
    }  
}
```

**-Met à jour chaque
particule et supprime les
particules en fin de vie**

```
void Particule::update(double delta) {  
    vx += ax * delta;  
    vy += ay * delta;  
    x += vx * delta;  
    y += vy * delta;  
    life -= delta;  
}
```

**-Met à jour la vitesse,
position et durée de vie
de la particule**