

RAPPORT DE JEU VIDEO 2D



Projet réalisé par : Oumaima Ettassouli

LES LOGICIELS ET PROGRAMMES UTILISÉS



PICO PARK



PRÉSENTATION DU JEU

Let's take a closer look at the game!

Ce projet « **PICO PARK** » a été développé via le moteur de jeu Cocos2D. C'est un Framework C++/Lua disponible en open source pour les systèmes iOS, Android, Windows Phone et beaucoup d'autres plateformes. Il est structuré autour d'un ensemble de classes de base qui fournissent plusieurs fonctionnalités pour construire le jeu.

Concernant l'idée du projet, elle est inspirée de Pico Park (un jeu de coopération de type Platformer Puzzle Game disponible sur Nintendo Switch et PC), mais en ajoutant une touche personnelle avec des fonctionnalités et des méthodes différentes de l'original.

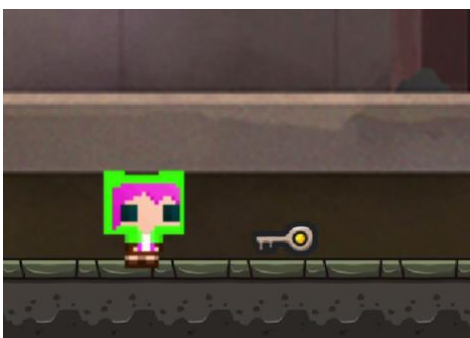
Le jeu est composé de 3 niveaux qui sont de plus en plus difficile au fur et à mesure qu'on avance dans le jeu (plus d'obstacles), alors il faut être prêt à relever de nouveaux défis !

Les map des 3 niveaux ont été créés à travers l'application « Tiled » qui permet de personnaliser et éditer des cartes de tuiles.

Quelles sont les règles du jeu ?



Il faut être attentif à l'environnement du jeu et être capable de repérer les pièges qui se trouvent sur votre chemin. Lorsque vous voyez le panneau signalant un piège, vous devrez utiliser le clavier pour faire sauter votre personnage par-dessus le piège afin de continuer à avancer dans le jeu.



Le joueur gagne la partie s'il parvient à collecter la clé.

SOMMAIRE

- EXPÉRIENCE PERSONNELLE
Difficultés rencontrées
- CONCEPTION
Classes
Ressources
- DÉVELOPPEMENT DU JEU
Code source + Interface Graphique
 - A .Main Menu / Levels Menu
 - B. Level 1/ 2/ 3
 - B.1) Création des cartes
 - B.2) Physics
 - B.3) Détection de la collision
 - C. Game Over scene
 - D. Win scene

Expérience personnelle

Honnêtement, travailler sur ce projet a été une expérience très gratifiante pour moi, cela m'a aidé à comprendre que je pouvais être mon propre soutien et à compter sur moi-même pour atteindre mes objectifs. Même si cela a été un véritable défi à certains moments.


















Parmi les difficultés que j'ai rencontré :

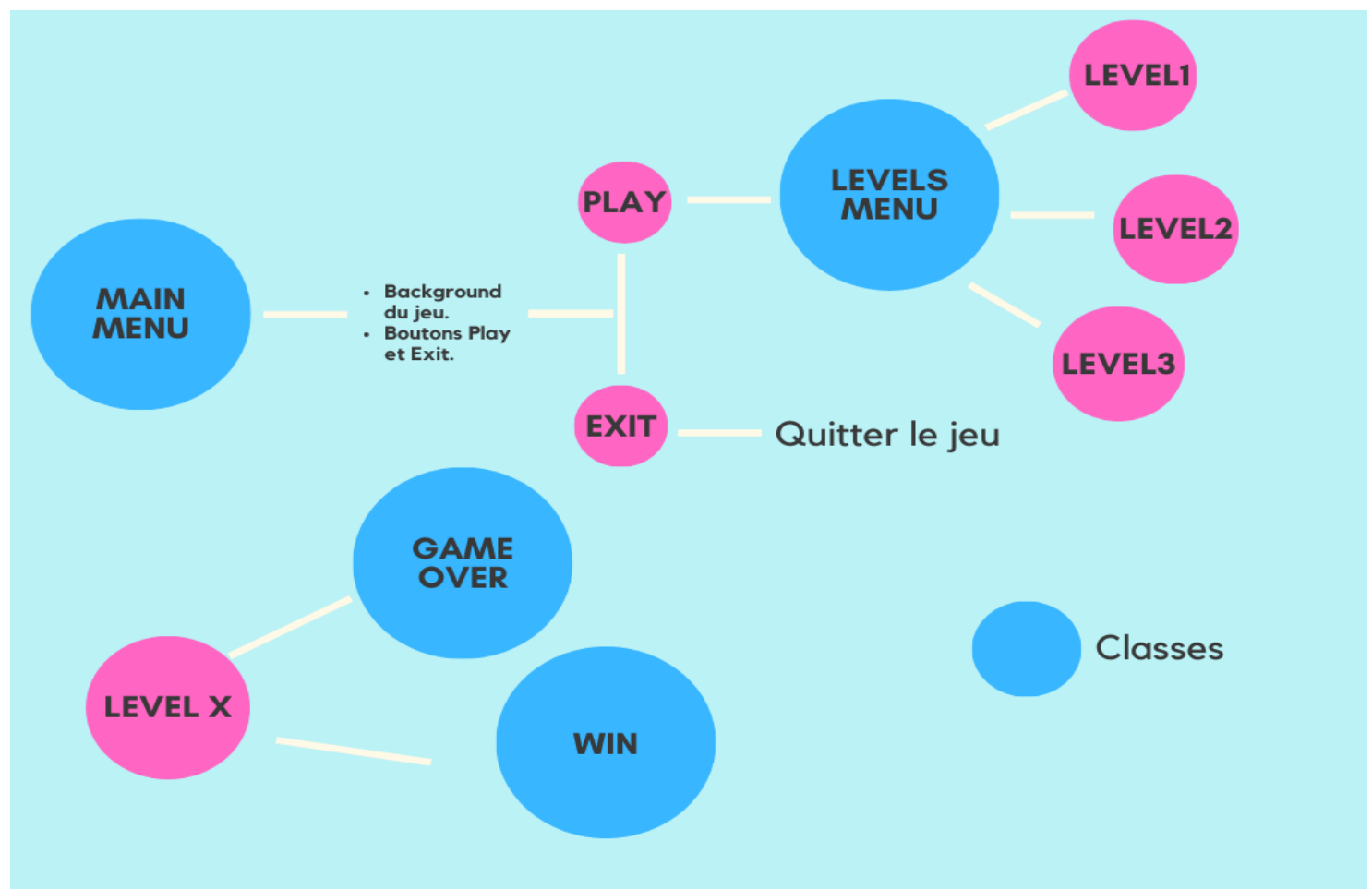
- La gestion de la Physique qui était d'ailleurs la partie la plus difficile. Utiliser le moteur physique Box2D peut prendre beaucoup de temps en soi car il nécessite une compréhension approfondie de comment il fonctionne. Par exemple, je devais déterminer comment appliquer une gravité appropriée au personnage, et veiller à ce que ses mouvements soient cohérents et réalistes (qui était loin d'être le cas au début)
- Gérer les collisions et les interactions entre les objets du jeu : Il était difficile de s'assurer que les collisions entre le personnage et les plateformes soient détectées et gérées de manière appropriée afin que le personnage ne passe pas à travers les objets, ou reste coincé dessus.
- Gérer les entrées du clavier : après plusieurs tentations et des heures de recherche, j'étais finalement capable de déterminer les différentes touches du clavier qui seront utilisées et le mouvement relatif à chaque touche (faire bouger le personnage, sauter...)
- Travailler avec des cartes Tiled : c'était compliqué de gérer les différents éléments de la carte, et puis l'importer dans cocos2Dx (en utilisant la classe `TMXTiledMap`) surtout que c'était mon tout premier contact avec l'application.
- Les erreurs et les bugs qui ne s'arrêtent pas : c'était trop frustrant de passer des heures devant l'écran essayant de résoudre toutes ces erreurs qui empêchent le jeu de fonctionner.

En général, malgré toutes les difficultés qu'on peut rencontrer dans n'importe quel projet, il faut rester positif, persévérant et surtout ne pas abandonner.





























CONCEPTION

Les classes

 AppDelegate	 Level3
 AppDelegate	 Level3
 Definition	 LevelsMenu
 GameOver	 LevelsMenu
 GameOver	 MainMenu
 Level1	 MainMenu
 Level1	 Win
 Level2	 Win
 Level2	



Ressources

 fonts	 res	 back1 Fichier PNG 3,98 Mo	 back1 Fichier source TypeScript 234 octet(s)
 back2 Fichier JPG 110 Ko	 back2 Fichier source TypeScript 330 octet(s)	 back3 Fichier JPG 28,7 Ko	 back3 Fichier source TypeScript 281 octet(s)
 exit Fichier PNG 32,3 Ko	 Gameover Fichier JPG 308 Ko	 home Fichier PNG 9,53 Ko	 Level1test Tiled map file 152 Ko
 Level2 Tiled map file 221 Ko	 Level3 Tiled map file 220 Ko	 levels background Fichier JPG 216 Ko	 lvl1 Fichier PNG 10,1 Ko
 lvl2 Fichier PNG 11,9 Ko	 lvl3 Fichier PNG 15,4 Ko	 menu Fichier PNG 33,3 Ko	 PICO PARK WALLPAPER Fichier JPG 162 Ko
 player2 Fichier PNG 4,76 Ko	 start Fichier PNG 33,9 Ko	 tileForProjectCPP Fichier PNG 78,4 Ko	 tileForProjectCPP Fichier source TypeScript 11,0 Ko
 tileForProjectCPP2 Fichier source TypeScript 250 octet(s)	 tileForProjectCPP3 Fichier source TypeScript 250 octet(s)	 try again Fichier PNG 47,9 Ko	 win Fichier PNG 930 Ko

Tous les designs du jeu ont été créés de manière manuelle et personnelle. (Les cartes des 3 niveaux, les scènes...)

DÉVELOPPEMENT DU JEU

A) Main Menu

C'est la première scène qui s'affiche lors du lancement du jeu.

Pour ajouter un background au jeu, j'ai utilisé la classe `Sprite` de cocos2D qui permet de créer un objet en lui attribuant l'image souhaité.

Ensuite, j'ai configuré l'objet pour qu'il occupe toute la largeur et la hauteur de l'écran.

```
//Création du background du jeu + son positionnement

auto background = Sprite::create("PICO PARK WALLPAPER.jpg");
background->setPosition(Point((visibleSize.width / 2) + origin.x, (visibleSize.height / 2) + origin.y));
this->addChild(background);

float rX = visibleSize.width / background->getContentSize().width;
float rY = visibleSize.height / background->getContentSize().height;

background->setScaleX(rX);
background->setScaleY(rY);
```

De la même manière, j'ai ajouté les boutons PLAY et EXIT en utilisant les classes `Menu` et `MenuItemImage`.

```
//Création des boutons PLAY et EXIT

auto menu_item_1 = MenuItemImage::create("start.png", "start.png", CC_CALLBACK_1(MainMenu::GotToLevelsMenu, this));
auto menu_item_2 = MenuItemImage::create("exit.png", "exit.png", CC_CALLBACK_1(MainMenu::menuCloseCallback, this));

menu_item_1->setPosition(Vec2(405, 63));
menu_item_2->setPosition(Vec2(70, 65));

auto* menu = Menu::create(menu_item_1, menu_item_2, NULL);
menu->setPosition(Point(0, 0));
this->addChild(menu);

return true;
```

Pour le fonctionnement de chacun de ces deux boutons :

La fonction de rappel `CALLBACK` permet de basculer les scènes du jeu. Pour le bouton Play, une fois cliqué dessus, cela nous mènera vers LevelsMenu (avec une transition) où l'utilisateur choisit le niveau qu'il veut jouer. Sinon, quitter le jeu en utilisant la méthode `end()` de la classe `Director` en cliquant sur Exit.


```

//clic sur PLAY -> LevelsMenu

void MainMenu::GotToLevelsMenu(cocos2d::Ref* pSender)
{
    auto scene = LevelsMenu::create();
    Director::getInstance()->replaceScene(TransitionFade::create(TRANSITION_TIME, scene));
}

//clic sur EXIT -> Quitter le jeu

void MainMenu::menuCloseCallback(Ref* pSender)
{
    Director::getInstance()->end();
}
////////////////////////////////////

```

Capture d'écran de la scène MainMenu



Remarque: Même chose pour LevelsMenu (Ajout du background + boutons)

B) LEVELS

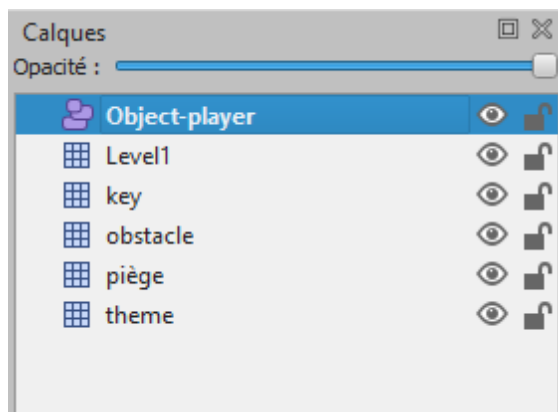
Création des cartes

Le jeu comprend 3 niveaux, chacun composé d'une carte unique créé avec l'application Tiled. Parmi les propriétés de la carte on trouve :

- **Calques** : une carte peut contenir plusieurs calques, chacun contenant des tuiles ou des objets.
- **Tuiles** : des éléments graphiques utilisés pour composer les niveaux du jeu.
- **Objets** : des éléments placés sur la carte qui peuvent être utilisés pour définir des zones de collision...



Tuiles



Calques

Pour intégrer les cartes créées avec Tiled dans le jeu en cocos2Dx, on utilise les classes suivantes :

- **TMXTiledMap** : permet de charger et gérer les cartes. Elle permet d'accéder aux calques, aux tuiles et aux objets de la carte.
- **TMXLayer** : représente un calque de la carte.
- **TMXObjectGroup** : représente un groupe d'objets de la carte.

Level 1 : Ajout de la carte et des calques

```
//Ajout de Tile Map + Positionnement
map_level1 = TMXTiledMap::create("Level1test.tmx");

float XX = map_level1->getPosition().x;
float YY = map_level1->getPosition().y;

//Ajouter les calques
_background1 = map_level1->getLayer("Level1");
theme = map_level1->getLayer("theme");
piege = map_level1->getLayer("piège");
obstacle = map_level1->getLayer("obstacle");
key = map_level1->getLayer("key");
addChild(map_level1);
```

Physics

La bibliothèque de physique intégrée de cocos2dx, basée sur Box2D, offre une variété d'outils pour créer des mouvements et des comportements réalistes en utilisant des concepts tel que la gravité, la friction et la collision.

```
//Introduction de PhysicsBody au joueur
PhysicsBody* player_body;

//The Physic Material is used to adjust friction and bouncing effects of colliding objects.
player_body = PhysicsBody::createBox(_player->getContentSize(), PhysicsMaterial(0.0f, 0.0f, 0.7f));

player_body->setDynamic(true);
player_body->setRotationEnable(false);
player_body->addMass(10);
_player->setPosition(x, y);
_player->setPhysicsBody(player_body);

this->addChild(_player);

//The speed limit module is used to set the speed of the particles to gradually slow down over the life cycle.
float limit = player_body->getVelocityLimit();
```

```

PhysicsBody* tilePhysics;

//Largeur de la map
for (int x = 0; x < 70; x++)
{
    //Hauteur de la map
    for (int y = 0; y < 32; y++)
    {
        auto spriteTile = _background1->getTileAt(Vec2(x, y));
        if (spriteTile != NULL)
        {
            tilePhysics = PhysicsBody::createBox(Size(24, 24), PhysicsMaterial(0.0f, 0.0f, 2.0f));
            tilePhysics->setDynamic(false);
            tilePhysics->setGravityEnable(false);
            tilePhysics->addMass(100);
            spriteTile->setPhysicsBody(tilePhysics);
        }
    }
}

```

Pour simuler un comportement physique réaliste pour les tuiles du jeu, on doit ajouter un corps physique à chaque tuile.

Cela va être appliqué à toutes les tuiles de la carte tant qu'ils existent.

```
spriteTile != NULL
```



Comment savoir si le joueur a perdu ou gagné?

DETECTION DE LA COLLISION

J'ai utilisé les méthodes de détection de collision par `BoundingBox` et `intersectsRect` de l'objet `CCRect` pour déterminer lorsque 2 objets entrent en contact.

Cas 1 : contact avec le piège -> Affichage de GameOver scene.

Cas 2 : contact avec la clé -> Affichage de Win scene.

```
//collision entre le piège et le joueur

for (int x = 0; x < 70; x++)
{
    for (int y = 0; y < 40; y++)
    {
        auto spriteTilePiege = piege->getTileAt(Vec2(x, y));

        if (spriteTilePiege != NULL)
        {
            //En cas de contact avec le piège---->GameOverScene
            if (spriteTilePiege->getBoundingBox().intersectsRect(_player->getBoundingBox())) {
                auto scene = GameOver2::create();
                Director::getInstance()->replaceScene(scene);
            }
        }
    }
}
```

```
//collision entre KEY et le joueur

for (int x = 0; x < 70; x++)
{
    for (int y = 0; y < 40; y++)
    {
        auto spriteTilePiege = key->getTileAt(Vec2(x, y));

        if (spriteTilePiege != NULL)
        {
            //En cas de contact avec le piège---->GameOverScene
            if (spriteTilePiege->getBoundingBox().intersectsRect(_player->getBoundingBox())) {
                auto scene = Win::create();
                Director::getInstance()->replaceScene(scene);
            }
        }
    }
}
```

C) Game Over

C'est la scène qui s'affichera lorsque le joueur perd dans une partie du jeu. Le joueur a la possibilité de recommencer le niveau en appuyant sur le bouton rejouer affiché à l'écran.

En utilisant la classe `Sprite` de `cocos2D`, j'ai créé un background du jeu et son Positionnement pour qu'il prend l'intégralité de l'écran. D'autre part, j'ai ajouté les boutons **Try Again** et **Home** en utilisant les classes `Menu` et `MenuItemImage`.

Les 2 boutons sont implémentés en utilisant un événement de toucher de `cocos2dx` qui Active une fonction de callback pour redémarrer le niveau lorsqu'il est appuyé.

Remarque: J'ai créé trois GameOver scenes pour le fonctionnement de chacun des 3 boutons « Rejouer » (chaque scène est utilisée dans un niveau différent).

```
//Création de la scène GameOver + positionnement

auto background = Sprite::create("Gameover.jpg");
background->setPosition(Point((visibleSize.width / 2) + origin.x, (visibleSize.height / 2) + origin.y));
this->addChild(background);

float rX = visibleSize.width / background->getContentSize().width;
float rY = visibleSize.height / background->getContentSize().height;

background->setScaleX(rX);
background->setScaleY(rY);

//Création du bouton try again pour permettre au joueur de rejouer + bouton home

auto tryagain = MenuItemImage::create("try again.png", "try again.png", CC_CALLBACK_1(GameOver::GotoLevel1, this));
tryagain->setScale(0.4);
tryagain->setPosition(Point((visibleSize.width / 2) + origin.x, (visibleSize.height / 2) + origin.y));
auto home = MenuItemImage::create("home.png", "home.png", CC_CALLBACK_1(GameOver::GoBackToMainMenu, this));

tryagain->setPosition(Vec2(140, 140));
home->setPosition(Vec2(450, 270));

auto menu = Menu::create(tryagain, home, NULL);
menu->setPosition(Point(0, 0));
this->addChild(menu);

return true;
```

D) Win scene

C'est la scène qui s'affichera lorsque le joueur complète le niveau avec succès. Elle est implémentée en utilisant les mêmes méthodes et classes que Game Over scene.

```
//Création de la scène Win + positionnement

auto background = Sprite::create("win.png");
background->setPosition(Point((visibleSize.width / 2) + origin.x, (visibleSize.height / 2) + origin.y));
this->addChild(background);

float rX = visibleSize.width / background->getContentSize().width;
float rY = visibleSize.height / background->getContentSize().height;

background->setScaleX(rX);
background->setScaleY(rY);

//Création des boutons home et menu

auto home = MenuItemImage::create("home.png", "home.png", CC_CALLBACK_1(Win::GoBackToMainMenu, this));
auto levelmenu = MenuItemImage::create("menu.png", "menu.png", CC_CALLBACK_1(Win::GotToLevelsMenu, this));
levelmenu->setScale(1);
levelmenu->setPosition(Vec2(170, 170));

home->setPosition(Vec2(450, 270));

auto menu = Menu::create(levelmenu, home, NULL);
menu->setPosition(Point(0, 0));

this->addChild(menu);

return true;
```



PETITE NOTE

Je tiens à remercier mes professeurs pour l'opportunité et l'idée de ce projet. Cela a été une expérience enrichissante pour moi. Bien que le jeu ne soit pas parfait, j'ai apprécié chaque moment de ce travail.

| Oumaima Ettassouli