Université Abdelmalek Essaadi Faculté
ses Sciences et techniques de Tanger
Département Génie Informatique

LST GI S5

Programmation Orientée Objet en C++

Projet : Jeux 2D PICO PARK,

Réalisé par :

Najjout Ayoub

Zaoui Hanane

Encadré par :

Pr . ELAACHAk LOTFI
Pr . Benabdelouahab Ikram

Objectif du projet :

L'objectif principal de ce projet est de maitriser la programmation orientée objet par la mise en place d 'un jeu vidéo 2D, le jeu proposé s'appelle pico park, c'est un jeu qui a connu un grand succès dans les plateformes mobile.

Table des matières

I. Introduction

1- Programmation Orientée Objet

La programmation orientée objet est une méthode de programmation informatique de plus en plus plébiscitée, que ce soit dans le développement logiciel ou la data science. Organisée autour des objets, ou données, la programmation orientée objet offre de nombreux avantages.

La programmation orientée objet (POO) est un paradigme informatique consistant à définir et à faire interagir des objets grâce à différentes technologies, notamment les langages de programmation (Python, Java, C++, Ruby, Visual Basic.NET, Simula…). On appelle objet, un ensemble de variables complexes et de fonctions, comme par exemple un bouton ou une fenêtre sur l'ordinateur, des personnes (avec les noms, adresse…), une musique, une voiture… Presque tout peut être considéré comme un objet. L'objectif de la programmation orientée objet est de se concentrer sur l'objet lui-même et les données, plutôt que sur la logique nécessaire et les actions à mener pour faire cette manipulation.

Les concepts clés de la POO sont :

- La classe : une classe est un ensemble de code contenant des variables et des fonctions permettant de créer des objets. Une classe peut contenir plusieurs objets.
- Les objets : un objet est un bloc de code mêlant des variables et des fonctions, appelées respectivement attributs et méthodes. Les attributs définissent les caractéristiques d'un objet d'une classe, les méthodes définissent quant à elles les fonctions propres aux instances d'une classe.
- L'encapsulation : l'encapsulation permet d'enfermer dans une capsule les données brutes afin d'éviter des erreurs de manipulation ou de corruptions des données. L'encapsulation permet ainsi de cacher des méthodes et des attributs à l'extérieur de la classe.
- L'héritage : le concept d'héritage signifie qu'une classe B va hériter des mêmes attributs et méthodes qu'une classe A. Lorsqu'une instance de la classe B est créée, on peut alors

appeler les méthodes présentes dans la classe A par la classe B. Cela va permettre de faire gagner du temps au programmeur.

- Le polymorphisme : lorsqu'une classe hérite des méthodes d'une classe parent, il est possible de surcharger une méthode, qui consiste à redéfinir la méthode de la classe parent pour que les deux classes ne fassent pas les mêmes tâches.

2- Cocos2d-x

Cocos2d-x est un Framework de développement de jeux multiplateforme open source mature qui prend en charge la création de jeux 2D et 3D. Le moteur fournit des fonctions riches telles que le rendu graphique, l'interface graphique, l'audio, le réseau, la physique, la saisie utilisateur, etc., et est largement utilisé dans le développement de jeux et la construction d'applications interactives. Son noyau est écrit en C++ et prend en charge le développement en C++, Lua ou JavaScript.
Cocos2d-x se déploie sur les systèmes iOS, Android, HTML5, Windows et Mac avec des fonctionnalités axées sur les plates-formes mobiles natives.

II. Etapes création du jeu

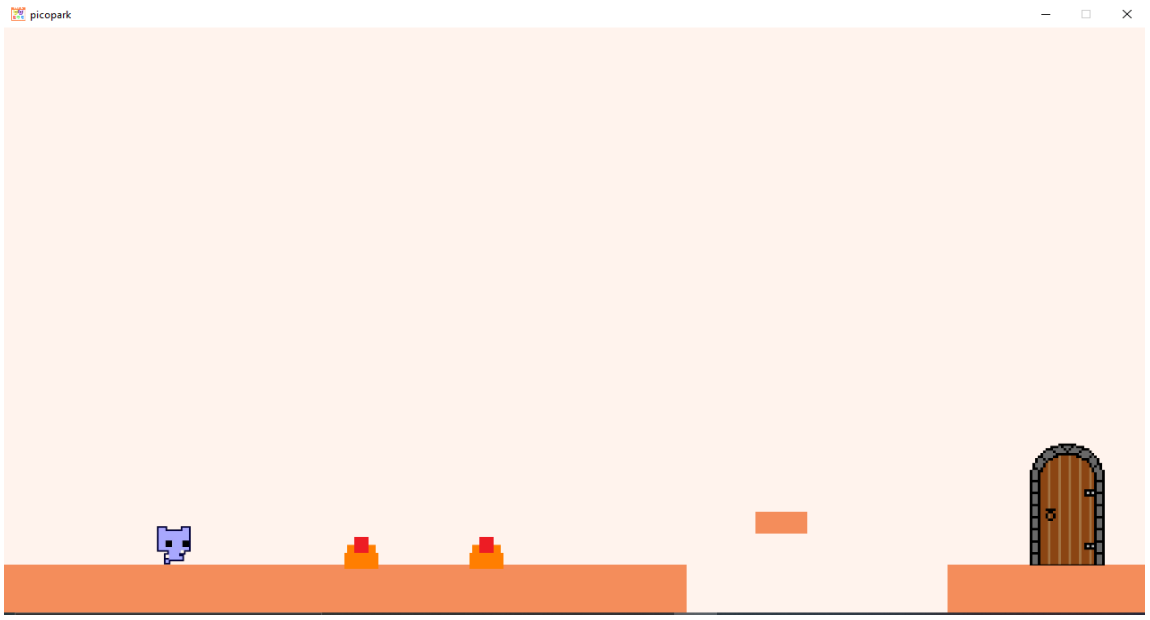
La création du jeu consiste en plusieurs étapes :

- La création de la page du menu qui compose de : le nom du jeu et un menu qui contient deux boutons : PLAY, qui nous dirige vers la scène de transition pour le niveau 1, et EXIT, qui ferme le jeu.

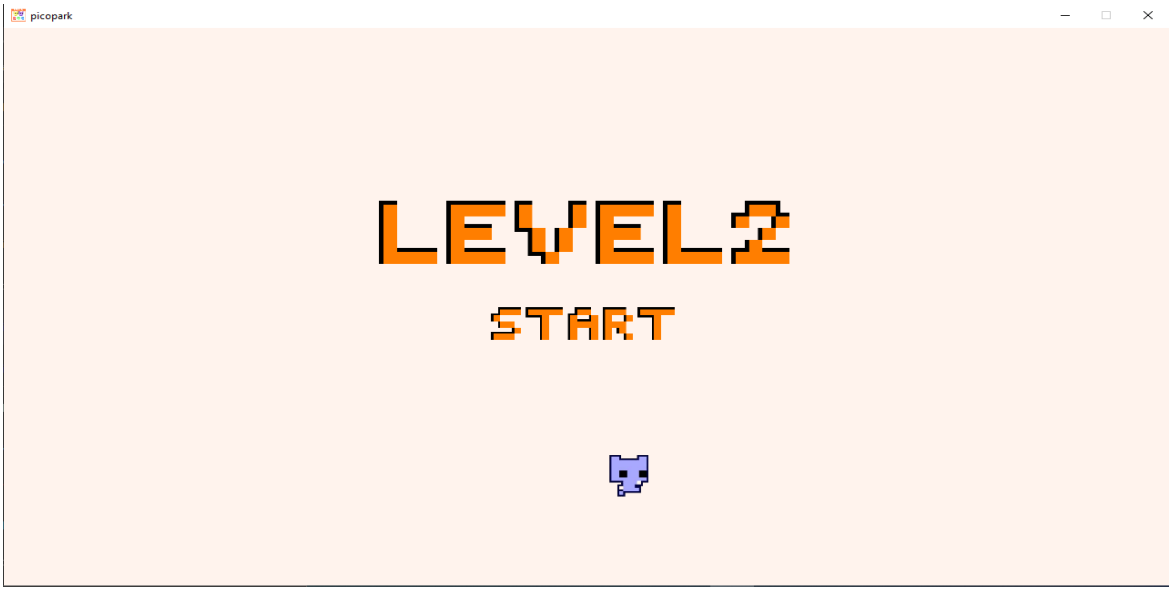
- La création de l'interface affiche une scène de transition pour le niveau 1 et qui compose de : le nom du niveau et un bouton START qui nous dirige vers le niveau 1. Il y a également un sprite qui se déplace jusqu'à un point puis revient à son point de départ.


picopark
LEVEL1
START

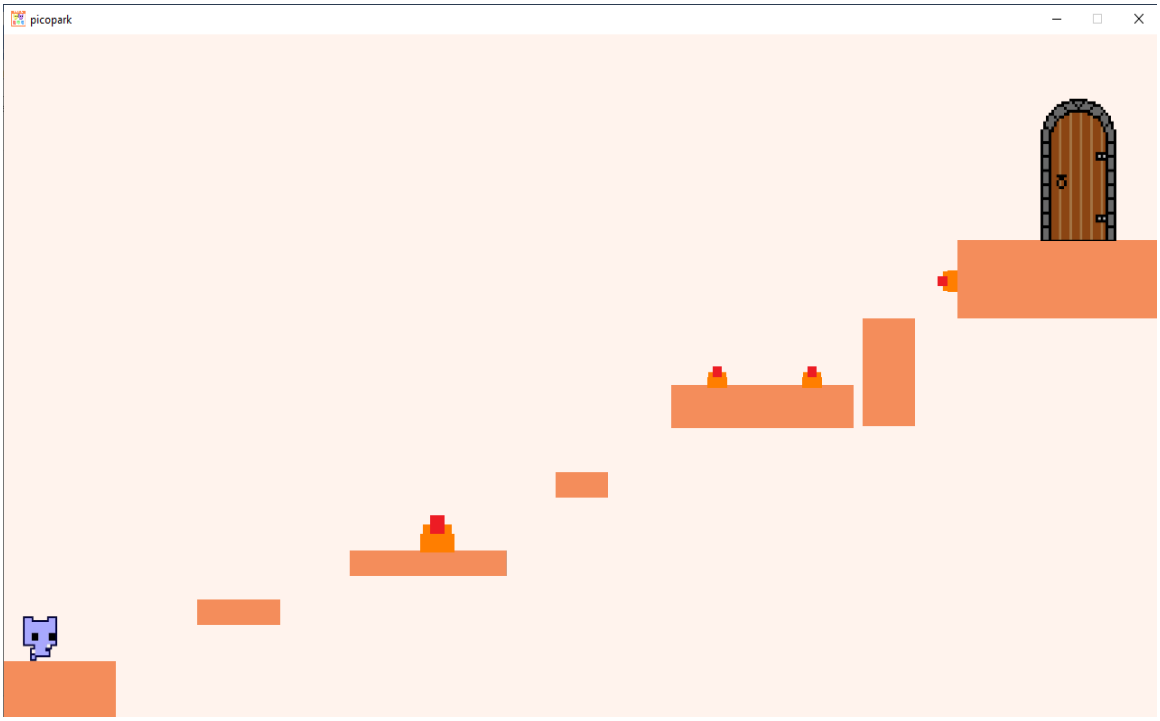
- La création de l'interface affiche le niveau 2, qui compose de : un sprite "avatar" qui peut se déplacer et sauter, des sprites qui déterminent le sol et des sprites qui sont des boutons : si l'avatar les touche, le niveau recommence. Finalement, il y a un sprite porte : si l'avatar le touche, il passe à la scène de transition du niveau 2.



- La création de l'interface affiche une scène de transition pour le niveau 2 et qui compose de : le nom du niveau et un bouton START qui nous dirige vers le niveau 2. Il y a également un sprite qui se déplace jusqu'à un point puis revient à son point de départ.

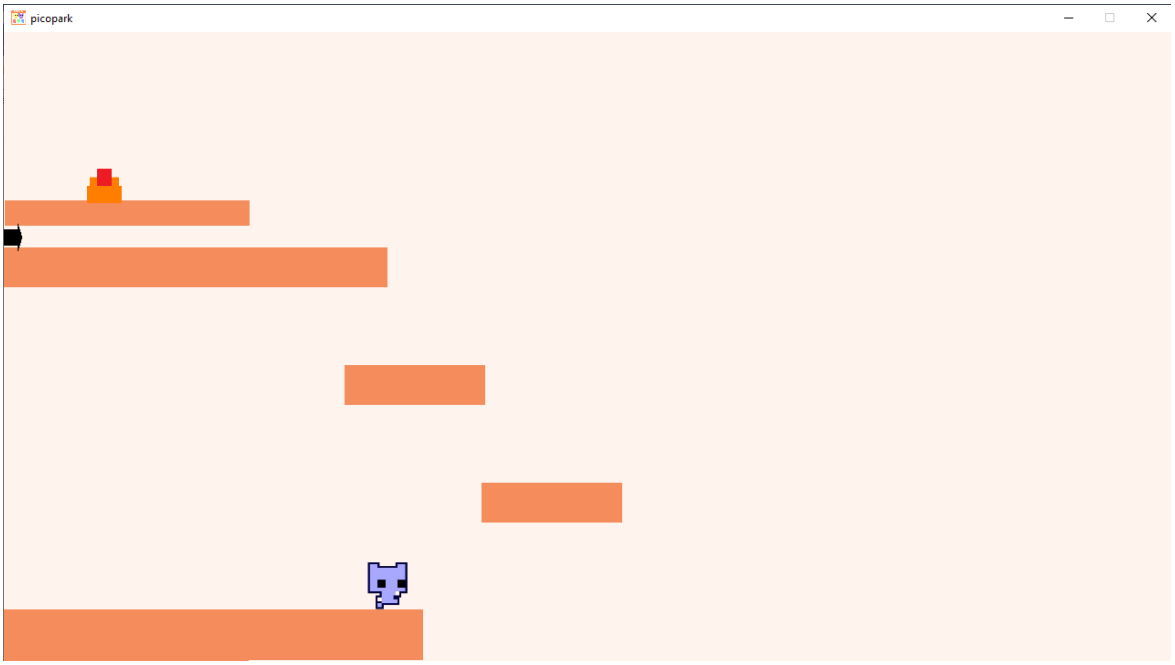


- La création de l'interface affiche le niveau 2, qui compose de : un sprite "avatar" qui peut se déplacer et sauter, des sprites qui déterminent le sol et des sprites qui sont des boutons : si l'avatar les touche, le niveau recommence. Finalement, il y a un sprite porte : si l'avatar le touche, il passe à la scène de transition du niveau 3.

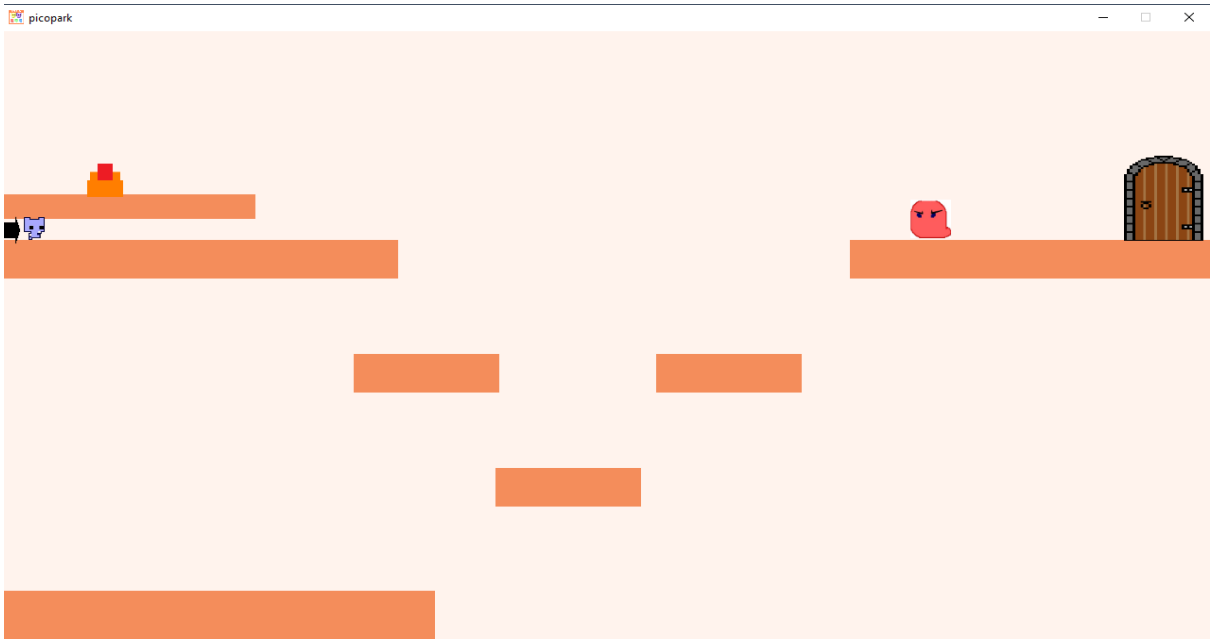


- La création de l'interface affiche une scène de transition pour le niveau 3 qui compose de : le nom du niveau et un bouton START qui nous dirige vers le niveau 3. Il y a également un sprite qui se déplace jusqu'à un point puis revient à son point de départ.

- La création de la scène qui affiche le niveau 3, qui est divisé en deux parties.
    - La première partie contient un sprite avatar qui se déplace et saute, des sprites qui déterminent le sol , un sprite bouton: si l'avatar le touche, sa taille diminue, et si il le touche une autre fois, sa taille revient à sa taille initiale et un sprite ROW: si l'avatar le touche, le reste du niveau 3 est affiché.



    - La deuxième partie contient des sprites qui déterminent le sol, un sprite MONSTER: si l'avatar le touche, le niveau recommence, et un sprite DOOR: si l'avatar le touche, il passe à la dernière scène, GAME OVER.


picopark

- La création de la scène affiche que le joueur a gagné.

III. Techniques utilisées

- Pour les éléments du menu principale on a commencé par créer l'attribut PLAY afin d'ajouter le bouton avec une image « f.png » et qui appelle la fonction play qui va nous diriger vers la scène de transition pour le niveau 1 ,et l'attribut closeItem afin d'ajouter le bouton avec une image « exit.png » et qui ferme le jeu.
  Pour créer PLAY et closeItem on a utilisé la fonction prédéfinit create de la classe MenuItemImage, on lui a donner sa position grâce a la fonction setPosition, et finalement on a créé menu avec la fonction create de la classe menu et on lui a attribuer PLAY et EXIT .

```
    auto play = MenuItemImage::create(
                                        "f.png",
                                        "f1.png",
        CC_CALLBACK_1(HelloWorld::GoTolevel1CutScene, this));


    if (play == nullptr ||
        play->getContentSize().width <= 0 ||
        play->getContentSize().height <= 0)
    {
        problemLoading("'f.png' and 'f.png'");
    }
    else
    {
        play->setPosition(Point(visibleSize.width/2 + origin.x, visibleSize.height -
380 + origin.y));
        play->setScale(2);
    }
    auto closeItem = MenuItemImage::create(
        "exit.png",
        "exit1.png",
        CC_CALLBACK_1(HelloWorld::menuCloseCallback, this));

    if (closeItem == nullptr ||
        closeItem->getContentSize().width <= 0 ||
        closeItem->getContentSize().height <= 0)

    {
        problemLoading("'exit.png' and 'exit.png'");
```

```
    }
    else
    {
        closeItem->setPosition(Point(visibleSize.width / 2 + origin.x,
visibleSize.height - 480 + origin.y));
        closeItem->setScale(2);
    }

    // create menu, it's an autorelease object
    auto menu = Menu::create(closeItem,play, NULL);
    menu->setPosition(Vec2::ZERO);
    this->addChild(menu,1);
```

- Pour le bouton Play on a ajouté la fonction `GoTolevel1CutScene` de la classe `level1CutScene` et après on a utilisé la méthode Director::getInstance()-> pushScene() pour afficher la scene `level1CutScene`.

```
    void HelloWorld::GoTolevel1CutScene(Ref* pSender)
{
    auto scene = level1CutScene::createScene();
    Director::getInstance()->pushScene(TransitionFade::create(0.01, scene));


}
```

- Pour la bouton EXIT on a ajouté la fonction `menuCloseCallback` et après on a utilisé la méthode `Director::getInstance()->end()` pour quitter le jeux .

```
void HelloWorld::menuCloseCallback(Ref* pSender)
{
    //Close the cocos2d-x game scene and quit the application
    Director::getInstance()->end();

}
```

- On a utilisé la même méthode pour démarrer un niveau. Exemple de la scène de transition du niveau 1 :

```
    void level1CutScene::GoToLevel1(Ref* pSender)
{
    auto scene = level1::createScene();
    Director::getInstance()->pushScene(TransitionFade::create(0.01, scene));


}
```

- Maintenant, passons à la création des niveaux. Tout d'abord, nous mettons l'arrière-plan et nous plaçons le terrain dans lequel nous avons utilisé des sprites pour déterminer le sol et des sprites pour déterminer les boutons, ainsi que des sprites pour ajouter la porte et enfin des sprites pour ajouter les monstres:
  - ➢ Exemple du niveau 1:
  - ○ Arrière-plan :

```
auto background = Sprite::create("background.png");
    if (background == nullptr)
    {
        problemLoading("'background.png'");
    }
    else
    {
        // position the sprite on the center of the screen

        background->setPosition(Point(visibleSize.width / 2 + origin.x,
visibleSize.height / 2 + origin.y));
        background->setScale(2);
        this->addChild(background, 0);

        // add the sprite as a child to this layer

    }
```

  - ○ Terrain :

```
//fixed land 2

    auto land1 = Sprite::create("land.png");
    land1->getPhysicsBody()->setMass(0);
    auto landBody1 = PhysicsBody::createBox(land1->getContentSize(),
PHYSICSBODY_MATERIAL_DEFAULT);
    landBody1->setDynamic(false);
```

```
    land1->setPosition(Point(visibleSize.width / 2 + origin.x + 800,
visibleSize.height / 2 + origin.y - 320));
    land1->setPhysicsBody(landBody1);
    addChild(land1);



    //mini land

    {
        auto sprite1 = Sprite::create("sprite1.png");
        auto sprite1Body = PhysicsBody::createBox(sprite1->getContentSize(),
PHYSICSBODY_MATERIAL_DEFAULT);
        sprite1->setPosition(Point(visibleSize.width / 2 + origin.x + 230,
visibleSize.height / 2 + origin.y - 230));
        sprite1Body->setDynamic(false);
        sprite1->setScale(0.2, 0.35);
        sprite1->setPhysicsBody(sprite1Body);
        addChild(sprite1);

        // void replay

        auto sprite12 = Sprite::create("sprite1.png");
        auto sprite12Body = PhysicsBody::createBox(sprite1->getContentSize(),
PHYSICSBODY_MATERIAL_DEFAULT);
        sprite12->setPosition(Point(visibleSize.width / 2 + origin.x + 230,
visibleSize.height / 2 + origin.y - 530));
        sprite12Body->setDynamic(false);
        sprite12->setScale(1, 1);
        sprite12->setPhysicsBody(sprite12Body);
        addChild(sprite12);
        sprite12Body->setCollisionBitmask(1);
        sprite12Body->setCategoryBitmask(1);
        sprite12Body->setContactTestBitmask(1);

        //btn dont touch

        auto btn = Sprite::create("btn.png");
        auto btnBody = PhysicsBody::createBox(btn->getContentSize(),
PHYSICSBODY_MATERIAL_DEFAULT);
        btn->setPosition(Point(visibleSize.width / 2 + origin.x - 100,
visibleSize.height / 2 + origin.y - 266));
        btnBody->setDynamic(false);
        btn->setScale(0.2, 0.35);
        btnBody->setCollisionBitmask(1);
        btnBody->setCategoryBitmask(1);
        btnBody->setContactTestBitmask(1);
        btn->setPhysicsBody(btnBody);
        addChild(btn);

        auto btn1 = Sprite::create("btn.png");
```

```
        auto btnBody1 = PhysicsBody::createBox(btn->getContentSize(),
PHYSICSBODY_MATERIAL_DEFAULT);
        btn1->setPosition(Point(visibleSize.width / 2 + origin.x - 240,
visibleSize.height / 2 + origin.y - 266));
        btnBody1->setDynamic(false);
        btn1->setScale(0.2, 0.35);
        btnBody1->setCollisionBitmask(1);
        btnBody1->setCategoryBitmask(1);
        btnBody1->setContactTestBitmask(1);
        btn1->setPhysicsBody(btnBody1);
        addChild(btn1);
    }
     //door for level2

    auto sprite10 = Sprite::create("door.png");
    auto sprite1Body10 = PhysicsBody::createBox(sprite10->getContentSize(),
PHYSICSBODY_MATERIAL_DEFAULT);
    sprite10->setPosition(Point(visibleSize.width / 2 + origin.x + 550,
visibleSize.height / 2 + origin.y - 208));
    sprite1Body10->setDynamic(false);
    sprite10->setScale(0.3, 0.3);
    sprite10->setPhysicsBody(sprite1Body10);
    addChild(sprite10);
    sprite1Body10->setContactTestBitmask(1);
    sprite1Body10->setCollisionBitmask(2);
    sprite1Body10->setCategoryBitmask(1);
```

- Apres on crée l'avatar:

```
  //Avatar

    auto avatar = Sprite::create("avatar1.png");
    auto avatarBody = PhysicsBody::createBox(avatar->getContentSize(),
PhysicsMaterial(0, 0, 0));
    avatarBody->setDynamic(true);
    avatar->setPosition(Point(visibleSize.width / 2 + origin.x - 450,
visibleSize.height / 2 + origin.y - 266));
    avatar->setScale(0.35, 0.35);
    avatarBody->setMass(0.1);
    avatarBody->setRotationEnable(false);
    avatar->setPhysicsBody(avatarBody);
    avatarBody->setContactTestBitmask(1);
    avatarBody->setCollisionBitmask(1);
    avatarBody->setCategoryBitmask(1);
    addChild(avatar);
```

- Maintenant on a ajouter une méthode pour faire déplacer et sauter l'avatar par clavier .

```
              // Create a keyboard event listener

    auto keyboardListener = EventListenerKeyboard::create();
    keyboardListener->onKeyPressed = [avatar](EventKeyboard::KeyCode KeyCode, Event*
event)
    {

        if (KeyCode == EventKeyboard::KeyCode::KEY_UP_ARROW) {
            auto action1 = JumpBy::create(0.7f, Vec2(50, 50), 80.0f, 1);
            avatar->runAction(action1);
        }
        if (KeyCode == EventKeyboard::KeyCode::KEY_RIGHT_ARROW) {
            auto jump = JumpBy::create(0.7f, Vec2(100, 100), 100.0f, 1);
            MoveBy* moveAction = MoveBy::create(1, Vec2(100, 0));
            RepeatForever* repeatAction = RepeatForever::create(moveAction);
            avatar->runAction(repeatAction);
        }
        if (KeyCode == EventKeyboard::KeyCode::KEY_LEFT_ARROW) {
            auto jump = JumpBy::create(0.7f, Vec2(100, 100), 100.0f, 1);
            MoveBy* moveAction = MoveBy::create(1, Vec2(-70, 0));
            RepeatForever* repeatAction = RepeatForever::create(moveAction);
            avatar->runAction(repeatAction);

        }

    };
```

```
    keyboardListener->onKeyReleased = [avatar](EventKeyboard::KeyCode KeyCode, Event*
event)
    {
        if (KeyCode == EventKeyboard::KeyCode::KEY_RIGHT_ARROW) {
            avatar->stopAllActions();
        }
        if (KeyCode == EventKeyboard::KeyCode::KEY_LEFT_ARROW) {
            avatar->stopAllActions();
        }

    };

    this->getEventDispatcher()-
>addEventListenerWithSceneGraphPriority(keyboardListener, avatar);
```

- Finalement, on a créer une fonction de détection des collisions.
    - Une fonction de détection des collisions entre l'avatar et le sol et les boutons qui recommence le niveau.
    - Une fonction de détection des collisions entre l'avatar et le Monster qui recommence le niveau.
    - Une fonction de détection des collisions entre l'avatar et la porte qui nous dirige vers la scène de transition au la scène de Game over.

Exemple du level 3 :

```
// Add an event listener for contact events
    static int counter = 0;
    auto parentScene = Director::getInstance()->getRunningScene();
    auto contactListener = EventListenerPhysicsContact::create();
    contactListener->onContactBegin = [&](PhysicsContact& contact) {
        auto avatar = contact.getShapeA()->getBody()->getNode();
        auto cont = contact.getShapeB()->getBody()->getNode();
        if (avatar && cont) {
            if (avatar->getTag() == 1 && cont->getTag() == 2) {

                avatar->removeFromParent();
                auto delay = DelayTime::create(0.4); // Delay for 0.4 second
                auto replaceScene = CallFunc::create([]() {
                    auto levelScene = level3::createScene();
                Director::getInstance()->replaceScene(levelScene);
                    });
                auto sequence = Sequence::create(delay, replaceScene, nullptr);
                this->runAction(sequence);
```

```
        }
        else if (avatar->getTag() == 1 && cont->getTag() == 3) {
            counter++;
            if (counter % 2 == 1) {
                avatar->setScale(0.20, 0.20);
            }
            else if (counter % 2 == 0) {
                avatar->setScale(0.40, 0.40);
            }

        }
        else if (avatar->getTag() == 1 && cont->getTag() == 6) {
            avatar->removeFromParent();
            auto delay = DelayTime::create(0.5); // Delay for 1 second
            auto replaceScene = CallFunc::create([]() {
                auto levelScene = gameover::createScene();
            Director::getInstance()->replaceScene(levelScene);
                });
            auto sequence = Sequence::create(delay, replaceScene, nullptr);
            this->runAction(sequence);

        }
        else if (avatar->getTag() == 1 && cont->getTag() == 5) {
            avatar->removeFromParent();
            auto delay = DelayTime::create(0.4); // Delay for 0.4 second
            auto replaceScene = CallFunc::create([]() {
                auto levelScene = level3::createScene();
            Director::getInstance()->replaceScene(levelScene);
                });
            auto sequence = Sequence::create(delay, replaceScene, nullptr);
            this->runAction(sequence);

        }
        else if (avatar->getTag() == 1 && cont->getTag() == 4) {

            auto delay = DelayTime::create(0.1); // Delay for 0.4 second
            auto replaceScene = CallFunc::create([]() {
                auto levelScene = level3Part2::createScene();
            Director::getInstance()->replaceScene(levelScene);
                });
            auto sequence = Sequence::create(delay, replaceScene, nullptr);
            this->runAction(sequence);
        }


    }
    return true;
};
_eventDispatcher->addEventListenerWithSceneGraphPriority(contactListener, avatar);
```

IV. Les difficultés rencontrer :

Nous avons rencontré pas mal de difficultés :

- Nous n'avons pas trouvé beaucoup de tutoriels expliquant comment travailler avec Cocos2d-x et Visual Studio.
- La documentation de Cocos2d-x a été utile pour se familiariser avec les fonctions prédéfinies, mais ce n'était pas suffisant.
- Nous avons passé beaucoup de temps à comprendre la physique, qui était la partie la plus difficile du projet.
- Lors de nos recherches on trouvait pas mal de fonction quine sont pas applicable vue qu'il appartienne a une ancienne version de cocos2d-x.
- De plus, nous ne savions pas comment repartir le travail entre nous.

V. La répartition du travail :

Nous avons défini toutes les fonctions et les classes dont nous avions besoin pour développer ce jeu et nous nous sommes réparti le travail de recherche. Ensuite, nous avons développé les niveaux ensemble.

Ensuite, nous avons testé le jeu et rédigé le rapport, puis créé une vidéo de démonstration où nous avons testé tous les niveaux et les propriétés que nous avons développés.

VI. Conclusion :

Pour conclure la réalisation de tels projets ce n'était pas une chose facile pour nous et on a rencontré pas mal de difficultés surtout que la majorité des forums était en langue chinoise et encore lors de nos recherches on trouvait pas mal de fonction qui ne sont pas applicable vue qu'il appartienne a une ancienne version de cocos2d-x.
Mais personne ne peut nier que ce projet nous ait appris beaucoup de choses et il porte une valeur spéciale comme c'est notre premier projet de telle façon et c'est le fruit de nos efforts.
Ce projet nous a encore permis de travailler en groupe et partager tous nos savoirs entre nous, et nous a appris même la patience vue qu'après plusieurs essaye-on remarque aucun avancement mais c'était le contraire.
Finalement je tiens a remercier tout particulièrement Monsieur Lotfi ELAACHAK pour tous ses efforts lors du cours et ses prestigieux conseils.

VII. Références

Cocos2d-x Game Development Essentials - Cocos2d-x Cookbook
https://www.raywenderlich.com/ - Cocos2d-x v3 C++ Tutorial - YouTube
https://discuss.cocos2d-x.org/ - https://stackoverflow.com/