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

RAPPORT DE PROJET – POO C++

PICO PARK

«Jeux vidéo 2D »

Projet réalisé par

Jabir Khadiri

Bilal Chaair

Projet encadré par

Pr . ELAACHAk LOTFI

Année 2022-2023

Table des matières

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 desktop.

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 : 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 taches.

- 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 n

II. Etapes création du jeu

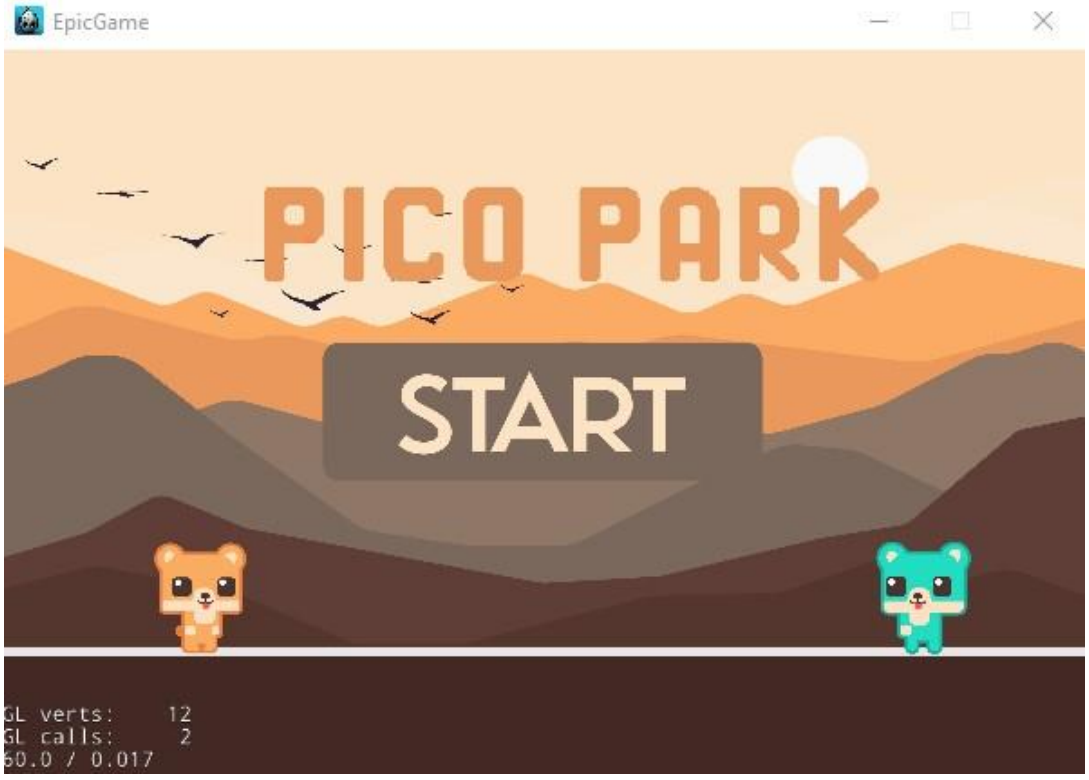
La création du jeu consiste en plusieurs étapes :

- Etape 1:

La création de la page de chargement du jeu :

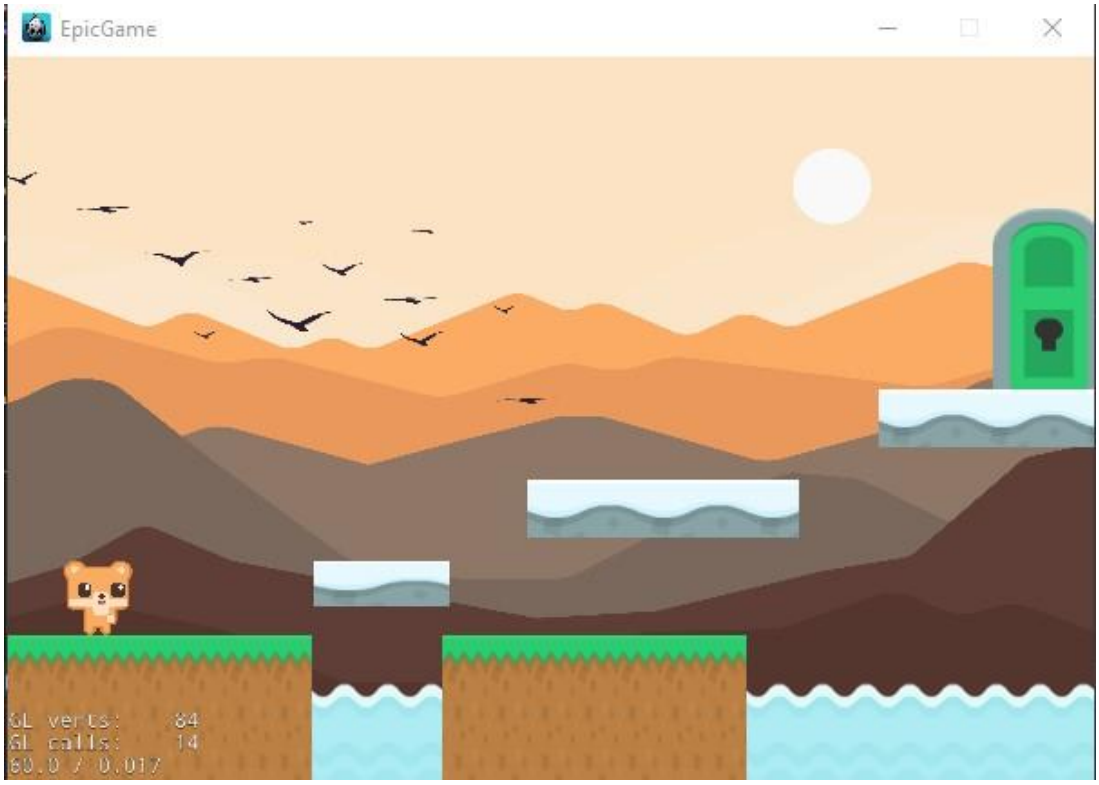
- Etape 2 :

La création de la page du menu avec le nom du jeu et un bouton START qui va nous diriger à une autre interface du niveau 1.



- Etape 3 :

Niveau 1



III. Techniques utilisées

- **Splash Scene:**

La création de la page de chargement est faite par la classe “SplashScene", qui contient le code suivant incluant une image <HelloWorld.png> en tant que sprite et sa résolution qui a été configurée par les fonctions setPosition et setScale.

```
if ( !Scene::init() )
{
    return false;
}

auto visibleSize = Director::getInstance()->getVisibleSize();
Vec2 origin = Director::getInstance()->getVisibleOrigin();

this->scheduleOnce(CC_SCHEDULE_SELECTOR( SplashScene::GoToMainMenuScene ), DISPLAY_TIME_SPLASH_SCENE);

auto backgroundSprite = Sprite::create("HelloWorld.png");
backgroundSprite->setPosition(Point(visibleSize.width / 2 , visibleSize.height / 2 ));
backgroundSprite->setScale(visibleSize.width / backgroundSprite->getContentSize().width, visibleSize.height / backgroundSprite->getContentSize().height);
this->addChild(backgroundSprite);

return true;
```

Ainsi il existe un header “SplashScene.h” qui permet de cree la function SplashScene.

```
#ifndef __SPLASH_SCENE_H__
#define __SPLASH_SCENE_H__

#include "cocos2d.h"

class SplashScene : public cocos2d::Scene
{
public:
    static cocos2d::Scene* createScene();

    virtual bool init();


    // implement the "static create()" method manually
       CREATE_FUNC(SplashScene);

private :
    void GoToMainMenuScene(float dt);
};

#endif // __SPLASH_SCENE_H__
```

- **MainMenuScene:**

Le passage de la page de chargement au menu du jeu effectué par la fonction replacescene comme dans le code source suivant :

```
void SplashScene::GoToMainMenuScene(float dt) {
    auto scene = MainMenuScene::createScene();
     Director::getInstance()->replaceScene(TransitionFade::create(TRANSITION_TIME, scene));
}
```

Le menu contient une image <background 3.png> qui a été créée en tant que sprite, et le bouton <start> au milieu de l'écran:

```
auto backgroundSprite = Sprite::create("background 3.png");
backgroundSprite->setPosition(Point(visibleSize.width / 2 + origin.x, visibleSize.height / 2 + origin.y));
backgroundSprite->setScale(visibleSize.width / backgroundSprite->getContentSize().width, visibleSize.height / backgroundSprite->getContentSize().height);

this->addChild(backgroundSprite);

auto playItem = MenuItemImage::create("start.png", "start.png", CC_CALLBACK_1(MainMenuScene::GoToGameScene, this));
playItem->setPosition(Point(visibleSize.width / 2 , visibleSize.height / 2 ));
playItem->setScale(visibleSize.width / backgroundSprite->getContentSize().width, visibleSize.height / backgroundSprite->getContentSize().height);
auto menu = Menu::create(playItem, NULL);
menu->setPosition(Point::ZERO);
this->addChild(menu);
```

Ainsi il existe un header « GameScene.h » qui permet de créer la fonction MainMenuScene, et la classe <<definition>> qu'ont definie le temps du passage d'une scene au autre :

```
#ifndef __MAIN_MENU_SCENE_H__
#define __MAIN_MENU_SCENE_H__

#include "cocos2d.h"

class MainMenuScene : public cocos2d::Scene
{
public:
    static cocos2d::Scene* createScene();

    virtual bool init();


    // implement the "static create()" method manually
    CREATE_FUNC ( MainMenuScene );

private :
    void GoToGameScene(cocos2d::Ref* sender);
};

#endif // __MAIN_MENU_SCENE_H__
```

```
#ifndef __DEFINITION_H__
#define __DEFINITION_H__

#define DISPLAY_TIME_SPLASH_SCENE 2
#define TRANSITION_TIME 0.5


#endif // __DEFINITION_H__
```

- **GameScene:**

Le passage du menu au Le niveau 1 du jeu effectué par la fonction replacescene comme dans le code source suivant:

```
void MainMenuScene::GoToGameScene(cocos2d::Ref* sender) {
    auto scene = GameScene::createScene();
    Director::getInstance()->replaceScene(TransitionFade::create(TRANSITION_TIME, scene));
}
```

Le premier niveau est un groupe de sprites indépendants comme les obstacles (obs41, obs2, obs3,door2), le sol(backgrounsprite2),le personnage(caractere) et l'arrière-plan(backgrounsprite1) qui ont des positions physiques et des corps différents :

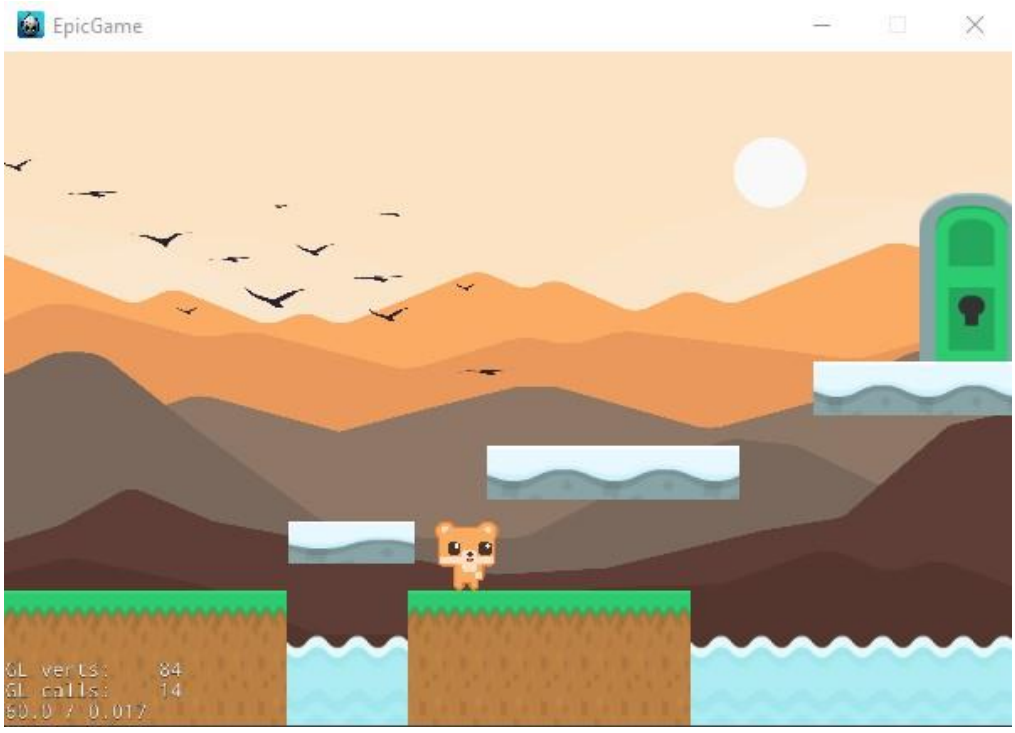
```
auto backgroundSprite1 = Sprite::create("background.png");
backgroundSprite1->setPosition(Point(visibleSize.width / 2 + origin.x, visibleSize.height / 2 + origin.y));
backgroundSprite1->setScale(visibleSize.width / backgroundSprite1->getContentSize().width, visibleSize.height / backgroundSprite1->getContentSize().height);
this->addChild(backgroundSprite1);

    auto backgroundSprite2 = Sprite::create("level1-1.png");
    backgroundSprite2->setPosition(Point(visibleSize.width / 2 + origin.x, visibleSize.height / 2 + origin.y));
    backgroundSprite2->setScale(visibleSize.width / backgroundSprite2->getContentSize().width, visibleSize.height / backgroundSprite2->getContentSize().height);
    backgroundSprite2->setName("backgroundSprite2");
    auto physicsBody = PhysicsBody::createBox(backgroundSprite2->getContentSize() / 1.5, PhysicsMaterial(1.0f, 0.0f, 20.0f));
    physicsBody->setDynamic(false);
    backgroundSprite2->setPhysicsBody(physicsBody);
   this->addChild(backgroundSprite2);

    auto caractere = Sprite::create("movein.png");
    caractere->setContentSize(Size(70, 60));
    caractere->setPosition(40,80);
    auto physicsBody2 = PhysicsBody::createBox(caractere->getContentSize(), PhysicsMaterial(100.0f, 0.0f, 20.0f));
    physicsBody2->setDynamic(false);
    physicsBody2->setContactTestBitmask(false);
    physicsBody2->setCollisionBitmask(1);
    physicsBody2->setCategoryBitmask(1);
    caractere->setRotation(0.0f);
    caractere->setPhysicsBody(physicsBody2);
    this->addChild(caractere);
```

```
auto obs1 = Sprite::create("obs1.png");
obs1->setContentSize(Size(60,20));
obs1->setPosition(165,87);
auto physicsBody3 = PhysicsBody::createBox(obs1->getContentSize() / 1.5, PhysicsMaterial(1.0f, 0.0f, 20.0f));
physicsBody3->setGravityEnable(false);
physicsBody3->setDynamic(false);
physicsBody3->setContactTestBitmask(1);
physicsBody3->setCollisionBitmask(1);
physicsBody3->setCategoryBitmask(1);
obs1->setRotation(0.0f);
obs1->setPhysicsBody(physicsBody3);
this->addChild(obs1);

auto obs2 = Sprite::create("obs2.png");
obs2->setContentSize(Size(120, 25));
obs2->setPosition(290, 120);
auto physicsBody4 = PhysicsBody::createBox(obs2->getContentSize() / 1.5, PhysicsMaterial(1.0f, 0.0f, 20.0f));
physicsBody4->setGravityEnable(false);
physicsBody4->setDynamic(false);
physicsBody4->setContactTestBitmask(1);
physicsBody4->setCollisionBitmask(1);
physicsBody4->setCategoryBitmask(1);
obs2->setRotation(0.0f);
obs2->setPhysicsBody(physicsBody4);
this->addChild(obs2);

auto obs3 = Sprite::create("obs3.png");
obs3->setContentSize(Size(130, 25));
obs3->setPosition(450, 160);
this->addChild(obs3);

auto door = Sprite::create("door2.png");
door->setContentSize(Size(50, 80));
door->setPosition(460, 213);
this->addChild(door);
```

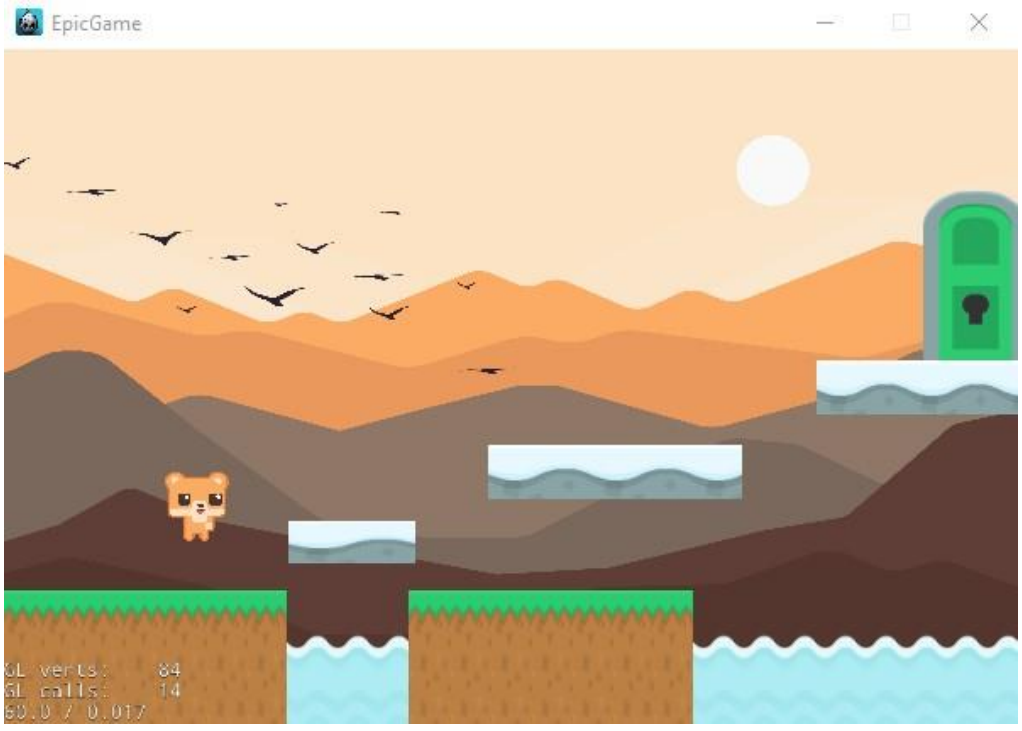
le personnage peut se déplacer sur le sol et sauter en utilisant les touches du clavier (Z,D,S,Q) ou les touches fléchées comme on peut le voir dans le code source suivant:

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

    Vec2 loc = event->getCurrentTarget()->getPosition();
    switch (keyCode) {
    case EventKeyboard::KeyCode::KEY_LEFT_ARROW:
    case EventKeyboard::KeyCode::KEY_A:
        event->getCurrentTarget()->runAction(MoveBy::create(0.01, Vec2(-10, 0)));
        break;
    case cocos2d::EventKeyboard::KeyCode::KEY_RIGHT_ARROW:
    case cocos2d::EventKeyboard::KeyCode::KEY_D:
        event->getCurrentTarget()->runAction(MoveBy::create(0.01, Vec2(10, 0)));
        break;
    case cocos2d::EventKeyboard::KeyCode::KEY_UP_ARROW:
    case cocos2d::EventKeyboard::KeyCode::KEY_W:
        event->getCurrentTarget()->runAction(JumpBy::create(0.5, Vec2(60, 0), 50, 1));
        break;
    case cocos2d::EventKeyboard::KeyCode::KEY_DOWN_ARROW:
    case cocos2d::EventKeyboard::KeyCode::KEY_S:
        event->getCurrentTarget()->runAction(JumpBy::create(0.5, Vec2(-60, 0), 50, 1));
        break;
    }
};
```

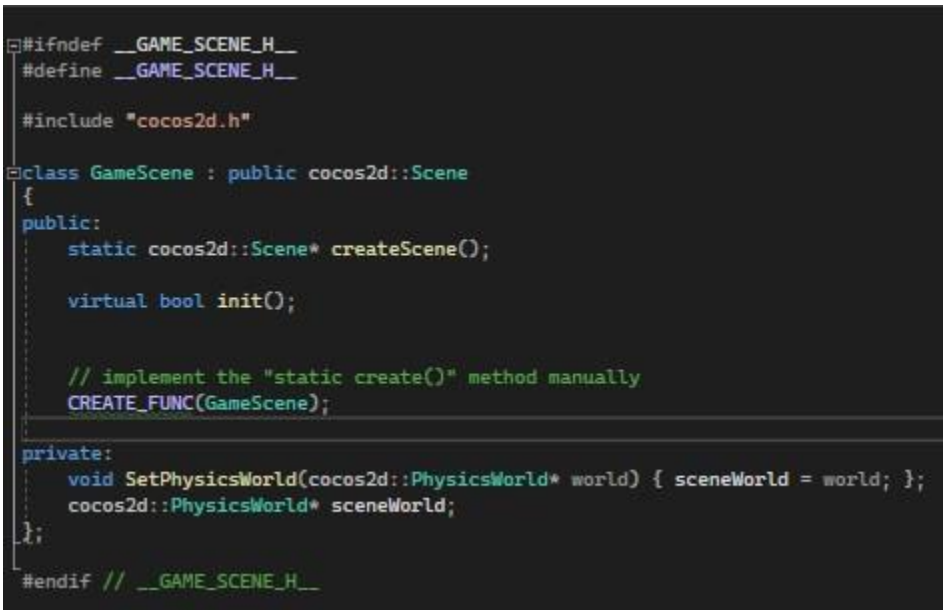
Le Déplacement de personnage:



la capabilité du personnage à sauter:



Ainsi il existe un header « GameScene.h » qui permet de créer la fonction GameScene:



```h
#ifndef __GAME_SCENE_H__
#define __GAME_SCENE_H__

#include "cocos2d.h"

class GameScene : public cocos2d::Scene
{
public:
    static cocos2d::Scene* createScene();

    virtual bool init();


    // implement the "static create()" method manually
    CREATE_FUNC(GameScene);

private:
    void SetPhysicsWorld(cocos2d::PhysicsWorld* world) { sceneWorld = world; };
    cocos2d::PhysicsWorld* sceneWorld;
};

#endif // __GAME_SCENE_H__
```

IV. Conclusion

En conclusion, bien que le développement d'un jeu en utilisant Cocos2dx et C++ ait été une expérience passionnante et éducative, le projet n'a pas été entièrement terminé. Malgré nos efforts et notre dévouement, nous n'avons pas réussi à terminer toutes les fonctionnalités et fonctionnalités prévues du jeu.

Cela a été une issue décevante, mais nous avons tiré des leçons précieuses de ce processus. Nous avons acquis une précieuse expérience de travail avec Cocos2dx et C++, et avons acquis une meilleure compréhension du temps et des ressources nécessaires pour développer un jeu.

En avançant, nous appliquerons ce que nous avons appris de cette expérience à des projets futurs. Nous serons également plus conscients de l'étendue et de la faisabilité de nos plans de projet afin de nous mettre en place pour réussir.

Nous tenons à remercier chaleureusement notre professeur < LOTFI ELAACHAK > pour son soutien et son encadrement tout au long de ce projet de développement de jeu. Sa guidance et ses conseils précieux ont été essentiels à notre réussite et nous sommes très reconnaissants pour son temps et de son énergie investis dans notre réussite.

V. Références

https://docs.cocos.com/cocos2d-x/manual/en/

https://github.com/cocos2d/cocos2d-x-samples

https://gamefromscratch.com/cocos2d-x-c-game-programming-tutorial-series/

https://www.raywenderlich.com/1848-cocos2d-x-tutorial-for-beginners#toc-anchor-001