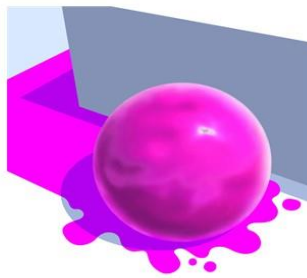


COMPTE RENDU

ROLLER SPLAT



Réaliser Par :

El Ballaoui Abdlghafour

El Houmaini karim

Encadré par :

Mme.Ikram Ben Abdel Ouahab

Licence Génie Informatique
Faculté des sciences et Techniques Tanger
Janvier 1, 2022

TABLE DES MATIERES

INTRODUCTION	1
Résumé	1
Définition	2
MISE EN PLACE DU PROJET	2
EXPLICATION DE CODE	3
Explication du concept	1
AppDelegate	2
IntroScene	3
Levels And Custom Classes	4
CONCLUSION	4
Organisation du travail	1
Conclusion	2

INTRODUCTION :

RESUME

Ce rapport décrit le processus impliqué dans la création d'un jeu informatique graphique RollerSplat en 2D avec C++.

En utilisant la librairie cocos2d-x avec c++, nous avons créé un jeu en 2D où l'objectif pour l'utilisateur est recouvrir le sol d'un labyrinthe avec la peinture.

Ce rapport traite de la vue d'ensemble du jeu, y compris la description et le jeu, puis se concentre sur la conception du jeu, décrivant comment le jeu est mis en œuvre et les fonctions de programmation et les bibliothèques utilisées dans la conception.

Les journaux de projet individuels de chacun des membres sont également inclus en annexe dans ce rapport.

DEFINITION

RollerSplat :

Dans le jeu mobile populaire Roller Splat, le joueur contrôle une balle roulant autour d'un plateau rectangulaire, avec un certain nombre des obstacles .

Lors d'un mouvement donné, la balle se déplace dans l'une des quatre directions cardinales jusqu'à ce qu'il heurte un obstacle.

Cocos2D-x :

Cocos2d-x est l'un des moteurs de jeu open source le plus populaire au monde. Cocos2d-x intègre deux langages de programmation principaux, C++ et Lua. Ce projet se concentrera sur l'implémentation C++. Il existe également une version JavaScript appelée Cocos2d-JS, qui prend également en charge le développement Web. Pour mettre les choses en perspective, quiconque a joué à un jeu mobile aura joué à un jeu qui a très probablement été construit à l'aide de Cocos2d-x. Les capacités du moteur de jeu s'étendent au-delà du développement de jeu, avec fonctionnalités pour le développement d'applications générales. Cependant, l'aspect le plus important qui rend Cocos2d-x phénoménal est sa nature multiplateforme, permettant le développement pour toutes les principales plates-formes mobiles et de bureau.


MISE EN PLACE DU PROJET :

Ces étapes vous guideront tout au long du processus de configuration.


1-Télécharger le dossier archivé RollerSplat.rar depuis GitHub.

 RollerSplat.rar	1/3/2022 4:25 PM	Archive WinRAR	8,475 KB
---	------------------	----------------	----------

2-ouvrir et extraire

 Classes.rar	22,725	22,725	Archive WinRAR	1/3/2022 4:25 ...	71A16415
 Resources.rar	8,654,834	8,654,834	Archive WinRAR	1/3/2022 1:45 ...	278E577B

3-extraire les sous dossier classes et Ressources

 Classes	1/3/2022 4:29 PM	File folder
 Resources	1/3/2022 4:29 PM	File folder

4-Remplacer le dossier classes par le dossier classes qui se trouve dans le répertoire de projet cocos2d-x « Game\MyCppGame\Classes »

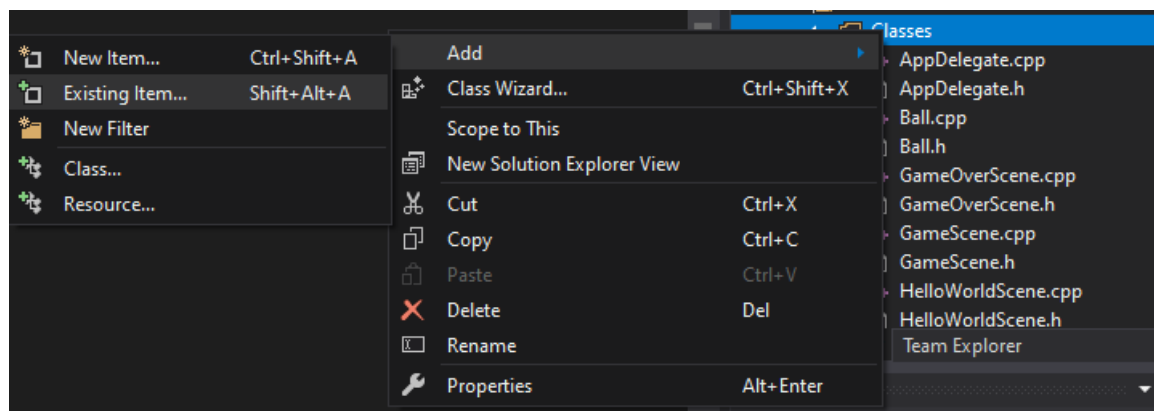
Refaire la même chose pour le dossier ressources.

Classes	1/3/2022 4:25 PM	File folder	
cocos2d	12/4/2021 12:35 AM	File folder	
proj.android	12/4/2021 12:34 AM	File folder	
proj.ios_mac	12/4/2021 12:34 AM	File folder	
proj.linux	12/4/2021 12:34 AM	File folder	
proj.win32	1/3/2022 3:28 PM	File folder	
Resources	1/3/2022 1:45 PM	File folder	
.cocos-project.json	12/4/2021 12:46 AM	JSON File	1 KB
CMakeLists.txt	12/30/2021 11:12 PM	Text Document	6 KB

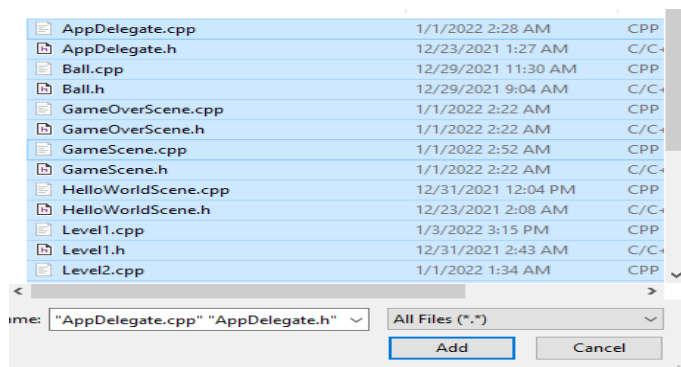
5- Si vous avez des fichiers dans le dossier proj.win32 avec le même nom que les fichiers de notre dossier classes, les déplacer ou les supprimer .

6- Maintenant ouvrir notre solution du projet avec Visual studio Community.

7- sur la barre Solution Explorer, sélectionner notre dossier classes, supprimer tous les fichiers existants puis cliquer sur ADD Existing items.



8-Naviguer notre dossier classes et sélectionner toutes les fichiers.

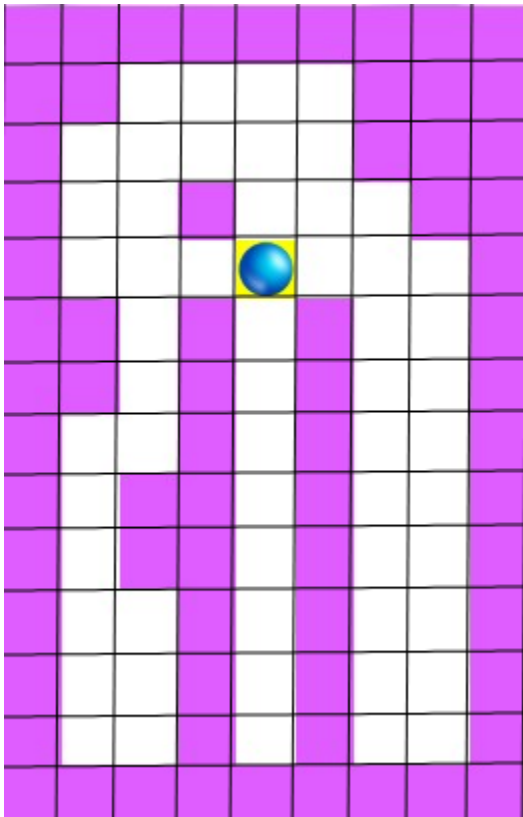


9-Refaire la même chose pour le dossier ressources.

10-Vous êtes prêt, Exécutez le projet et l'écran du jeu devrait apparaître.

EXPLICATION DU CONCEPT

Afin de réaliser ce projet, nous diviserons la scène en carreaux de taille similaire.



Nous allons ensuite créer deux carrés dans Photoshop, un rose pour l'obstacle et un blanc pour le chemin, suivi de notre Boule aux mêmes dimensions que le carré. Avec l'aide des fonctions de Cocos et Les boucles, nous pourrons créer notre propre scène.

Le carré sera coloré une fois la collision survenue, et on comptera le nombre de carrés colorés pour déterminer la fin de chaque niveau.

AppDelegate:

Avec notre appDelegate, nous avons inclus l'en-tête appDelegate.h et l'en-tête introScene.h .

Notre résolution s'appelle "phoneResolution", a été attribué à GLView avec « the Resolution policy » "SHOW_ALL" afin d'afficher la scène entière. De plus, nous avons choisi la résolution 1100x2200 pour simuler les écrans de téléphones d'aujourd'hui.

Enfin, nous avons déclaré une variable et l'avons affectée IntroScene, pour l'exécuter comme notre première scène.

```
#include "AppDelegate.h"
#include "IntroScene.h"

static cocos2d::Size phoneResolution = cocos2d::Size(1100, 2200);
```

```
// Set the design resolution
glview->setDesignResolutionSize(phoneResolution.width, phoneResolution.height, ResolutionPolicy::SHOW_ALL);

register_all_packages();

// create a scene. it's an autorelease object
auto scene = IntroScene::createScene();
// run
director->runWithScene(scene);
```

IntroScene :

IntroScene.h :

Nous avons déclaré la classe héritant avec des méthodes publiques dans le fichier d'en-tête IntroScene.h .

```
#ifndef __INTRO_SCENE_H__
#define __INTRO_SCENE_H__

#include "cocos2d.h"

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

    virtual bool init();

    void GoToGameScene(Ref* pSender);

    CREATE_FUNC(IntroScene);
};

#endif
```

IntroScene.cpp :

Nous avons tout d'abord créé notre scène.

Ensuite, nous incluons l'en-tête IntroScene, Level1.h puisque ce sera la prochaine scène, et AudioEngine.h qui nous permet d'utiliser le son.

Après nous avons créé la scène avec la méthode createScene.

```
#include "IntroScene.h"
#include "Level1.h"
#include "AudioEngine.h"

USING_NS_CC;

Scene* IntroScene::createScene()
{
    auto scene = Scene::create();
    auto layer = IntroScene::create();
    scene->addChild(layer);
    return scene;
}
```

Dans la méthode « init », nous avons implémenté la musique de fond, l'image de fond, et nous avons créé un bouton de démarrage qui déclenche une fonction (GoToGameScene) qui fera une transition en douceur vers le niveau 1.

```

// On init you have to initialize your instance
bool IntroScene::init()
{
    if ( !Layer::init() )
    { return false; }

    cocos2d::AudioEngine::preload("arcade sound.mp3");
    cocos2d::AudioEngine::play2d("arcade sound.mp3", true);

    auto mainmenu = Sprite::create("mainmenu.png");
    this->addChild(mainmenu,0);
    mainmenu->setPosition(550,1100);

    auto playItem =MenuItemImage::create("Start.png", "Start.png",
                                         CC_CALLBACK_1(IntroScene::GoToGameScene, this));
    auto menu = Menu::create(playItem, NULL);
    menu->setPosition(550, 700);
    this->addChild(menu);

    return true;
}

void IntroScene::GoToGameScene(Ref* pSender)
{
    auto scene = Level1::createScene();
    Director::getInstance()->replaceScene(TransitionFade::create
    (1.0, scene));
}

```

Levels And Custom Classes:

Custom classes:

Tout d'abord, examinons certaines de nos classes personnalisées.

Dans notre jeu, il existe 3 classes personnalisées pour les 3 composants les plus importants.

La class Ball:

Ball.h :

```
#include "cocos2d.h"
#include <string>

using namespace std;

class Ball : public cocos2d::Sprite
{
public:
    char direction;
    static Ball* create(const std::string& filename);
};

#endif
```

Afin de donner au joueur le contrôle du Balle , nous avons besoin d'un Sprite avec quelques propriétés supplémentaires, nous avons donc créé la classe Ball qui hérite de la classe "Sprite".

Il était nécessaire de redéfinir la méthode « create » dans cocos2d::Sprite afin de rendre une balle capable d'être déclarée comme un Sprite normal, voilà l'exemple :

```
auto sprite = Sprite::create("sprite.png");  
  
auto Ball = Ball::create("ball.png");  
auto obstacle = Obstacle::create("Obstacle.png");  
auto underSquar = UnderSquar::create("Obstacle.png");
```

Et à l'intérieur de la méthode create, nous avons implémenté la physique de la balle et le calque où elle serait affichée.

La redéfinition de la méthode create :

```
Sprite* Sprite::create(const std::string& filename)  
{  
    Sprite *sprite = new (std::nothrow) Sprite();  
    if (sprite && sprite->initWithFile(filename))  
    {  
        sprite->autorelease();  
        return sprite;  
    }  
    CC_SAFE_DELETE(sprite);  
    return nullptr;  
}
```

Ball.cpp:

```
#include "Ball.h"
USING_NS_CC;

Ball* Ball::create(const std::string& filename)
{
    Ball* ball = new (std::nothrow) Ball();
    if (ball && ball->initWithFile(filename))
    {
        ball->autorelease();

        auto ballBody = PhysicsBody::createBox(ball->getContentSize(), PhysicsMaterial(0, 0, 0));
        ballBody->setDynamic(false);
        ballBody->setContactTestBitmask(1);
        ballBody->setCollisionBitmask(1);

        ball->setPhysicsBody(ballBody);
        ball->setTag(1);

        return ball;
    }
    CC_SAFE_DELETE(ball);
    return nullptr;
}
```

La classe UnderSquar:

On a fait la même chose pour la classe « UnderSquar »:

UnderSquar.h:

```
#ifndef __UNDER_SQUAR_H__
#define __UNDER_SQUAR_H__

#include "cocos2d.h"
#include <string>

using namespace std;

class UnderSquar : public cocos2d::Sprite
{
public:
    static UnderSquar* create(const std::string& filename);
};

#endif
```

UnderSquar.cpp:

```
#include "UnderSquar.h"
USING_NS_CC;

UnderSquar* UnderSquar::create(const std::string& filename)
{
    UnderSquar* squar = new (std::nothrow) UnderSquar();
    if (squar && squar->initWithFile(filename))
    {
        squar->autorelease();

        auto Body = PhysicsBody::createBox(squar->getContentSize(), PhysicsMaterial(0, 0, 0));
        Body->setDynamic(false);
        Body->setContactTestBitmask(2);
        Body->setCollisionBitmask(2);
        squar->setPhysicsBody(Body);
        squar->setTag(3);

        return squar;
    }
    CC_SAFE_DELETE(squar);
    return nullptr;
}
```


La classe Obstacle:

On a fait la même chose pour la classe « Obstacle »:

Obstacle.h:

```
#ifndef __OBSTACLE_H__
#define __OBSTACLE_H__

#include "cocos2d.h"
#include <string>

using namespace std;

class Obstacle : public cocos2d::Sprite
{
public:
    static Obstacle* create(const std::string& filename);
};

#endif
```

Obstacle.cpp:

```
#include "Obstacle.h"
USING_NS_CC;

Obstacle* Obstacle::create(const std::string& filename)
{
    Obstacle* squar = new (std::nothrow) Obstacle();
    if (squar && squar->initWithFile(filename))
    {
        squar->autorelease();

        auto Body = PhysicsBody::createBox(squar->getContentSize(), PhysicsMaterial(0, 0, 0));
        Body->setDynamic(false);
        Body->setContactTestBitmask(2);
        Body->setCollisionBitmask(2);
        squar->setPhysicsBody(Body);
        squar->setTag(2);

        return squar;
    }
    CC_SAFE_DELETE(squar);
    return nullptr;
}
```

Level 1

Le fichier Level1.h:

Comme vous pouvez le voir dans le fichier nommé level1.h, nous avons déclaré notre classe héritant avec des méthodes et variables qui seront utilisées dans le fichier `level1.cpp`.

```
#include "cocos2d.h"
#include "Ball.h"
#include "UnderSquar.h"
#include "Obstacle.h"

class Level1 : public cocos2d::Scene
{
public:
    static Scene* createScene();

    virtual bool init();

    void createItems();

    void mouvUp(float dt);
    void mouvDown(float dt);
    void mouvLeft(float dt);
    void mouvRight(float dt);

    void stoping(cocos2d::Vec2 vvv);
    void winStoping();
    void levelComplete();

    void goTONextLevel(Ref* pSender);
};
```

```

    Ball* ball;

    bool won = false;
    bool onMouv = false;
    float speed = 1800;
    int coloredCount;
    CREATE_FUNC(Level1);

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

    bool onContactBegin(cocos2d::PhysicsContact& contact);
};

#endif

```

Le fichier Level1.cpp:

Nous avons inclus nos en-têtes de niveau 1 et 2, ainsi que nos en-têtes de composants principaux, après nous avons créé notre scène de niveau 1 avec le Physique implémenté.

```
#include "Level1.h"
#include "Level2.h"
#include "Ball.h"
#include "UnderSquar.h"
#include "Obstacle.h"

USING_NS_CC;

Scene* Level1::createScene()
{
    auto level1 = Scene::createWithPhysics();
    level1->getPhysicsWorld()->setGravity(Vec2(0, 0));
    auto layer = Level1::create();
    layer->SetPhysicsWorld(level1->getPhysicsWorld());
    level1->addChild(layer);
    return level1;
}
```

The Background:

Afin de s'assurer que tous les autres composants apparaissent au-dessus de l'image d'arrière-plan du niveau 1, nous avons créé un Sprite et l'avons placé au fond du niveau (layer =0).

```
Sprite* background = Sprite::create("back.png");
background->setPosition(Vec2(origin.x + visibleSize.width / 2,
                             origin.y + visibleSize.height / 2));
this->addChild(background);
```

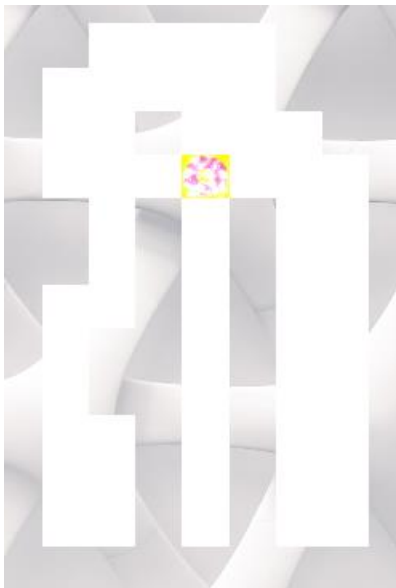
Main Objects creation and placement:

La méthode appelée "createItems" créera et positionnera chaque UnderSquar et Obstacle dans la scène.

```
createItems();
```

Nous avons un tableau de positions pour les Undersquar, puis une boucle qui parcourra le tableau et créera un UnderSquar à chaque point.

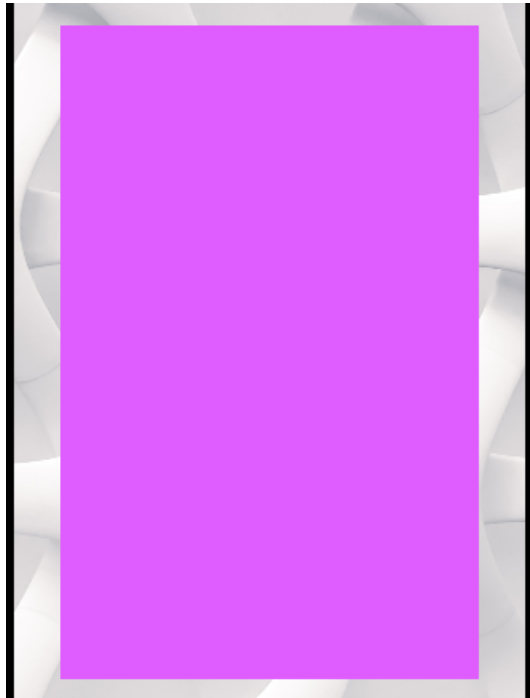
```
void Level1::createItems() {  
    Point positions[58] = {  
        Point(250, 550),Point(250, 650),Point(250, 750),Point(250, 850),Point(250, 950),Point(250, 1050),Point(250, 1350),  
        Point(250, 1450), Point(250, 1550),  
        Point(350, 550), Point(350, 650), Point(350, 750),Point(350, 1050),Point(350, 1150),Point(350, 1250),Point(350, 1350),  
        Point(350, 1450),Point(350, 1550),Point(350, 1650), Point(450, 1350),Point(450, 1550),Point(450, 1650),  
        Point(550, 550),Point(550, 650),Point(550, 750),Point(550, 850),Point(550, 950),Point(550, 1050),Point(550, 1150),  
        Point(550, 1250),Point(550, 1350),Point(550, 1450),Point(550, 1550),Point(550, 1650),  
        Point(650, 1350),Point(650, 1450), Point(650, 1550),Point(650, 1650),  
        Point(750, 550),Point(750, 650),Point(750, 750),Point(750, 750),Point(750, 850),Point(750, 950),Point(750, 1050),  
        Point(750, 1150),Point(750, 1250), Point(750, 1350),Point(750, 1450),  
        Point(850, 550), Point(850, 650),Point(850, 750),Point(850, 850),Point(850, 950),Point(850, 1050),Point(850, 1150),  
        Point(850, 1250), Point(850, 1350) };  
  
    for (Point p : positions)  
    {  
        auto Usqr = UnderSquar::create("whiteSquar.png");  
        this->addChild(Usqr, 1);  
        Usqr->setPosition(p);  
    }  
}
```



Au cours de la boucle suivante, des obstacles sont créés, positionnés et ajoutés à un vecteur Obstacles, les obstacles vont prendre une grande partie de la scene et vont couvrir les underSquars.

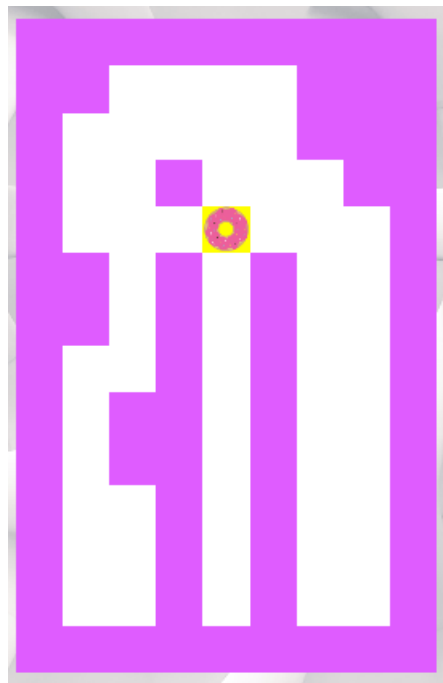
```
cocos2d::Vector <Obstacle*> Obstacles;

int count = 0;
for (float col = 450; col < 1800; col += 100) {
    for (float row = 150; row < 1000; row += 100) {
        auto obstacle = Obstacle::create("100x100.png");
        this->addChild(obstacle, 2);
        obstacle->setPosition(row, col);
        Obstacles.pushBack(obstacle);
    }
}
```



La dernière boucle de la méthode createItems supprime tous les obstacles es positions UnderSquare, de sorte que tous les composants soient en place.

```
for (auto op : positions)
{
    for (auto obst : Obstacles) {
        if (obst->getPosition() == op)
        {
            this->removeChild(obst);
        }
    }
}
```



The Input listener and Movement Dynamics:

La méthode "Init" crée EventListener qui détectera si le joueur a cliqué sur l'une flèche, et pendant que la balle ne bouge pas et que le niveau n'est pas terminé, elle attribuera une valeur au membre de la classe de direction de la balle (« l » fait référence à la gauche ; « r » fait référence à la droite ; « u » fait référence au haut ; « d » fait référence au bas), et il programmera des méthodes pour déplacer la balle 100 fois par seconde.

```
auto listener1 = EventListenerKeyboard::create();
listener1->onKeyPressed = [=](EventKeyboard::KeyCode keyCode, Event* event) {

    if (!onMouv && !won)
    {
        onMouv = true;
        switch (keyCode) {
            case EventKeyboard::KeyCode::KEY_LEFT_ARROW:
            case EventKeyboard::KeyCode::KEY_A:

                ball->direction = 'l';
                this->schedule(SEL_SCHEDULE(&Level1::mouvLeft), 0.01);
                break;

            case EventKeyboard::KeyCode::KEY_RIGHT_ARROW:
            case EventKeyboard::KeyCode::KEY_D:

                ball->direction = 'r';
                this->schedule(SEL_SCHEDULE(&Level1::mouvRight), 0.01);
                break;

            case EventKeyboard::KeyCode::KEY_UP_ARROW:
            case EventKeyboard::KeyCode::KEY_W:

                ball->direction = 'u';
                this->schedule(SEL_SCHEDULE(&Level1::mouvUp), 0.01);
                break;

            case EventKeyboard::KeyCode::KEY_DOWN_ARROW:
            case EventKeyboard::KeyCode::KEY_S:

                ball->direction = 'd';
                this->schedule(SEL_SCHEDULE(&Level1::mouvDown), 0.01);
                break;
        }
    }
};

_eventDispatcher->addEventListenerWithSceneGraphPriority(listener1, this);
```


Comme exemple de mouvement dynamique, nous pouvons prendre la méthode "mouvLeft": elle décrémente la position x de la balle sur chaque 'frame' (100 images par seconde = 100fps), de sorte que la balle continue à se déplacer vers la gauche . Cette méthode est similaire à la méthode "update".

On multiplie la vitesse par delta time pour lisser le mouvement.

```
void Level1::mouvLeft(float dt) {  
    auto position = ball->getPosition();  
    position.x -= speed * dt;  
    ball->setPosition(position);  
}
```

Collision detection and Stopping Dynamics:

Ensuite nous avons créé Contact Listener dans la méthode "Init" qui écoute toutes les occurrences de contact.

```
auto collisionListener = EventListenerPhysicsContact::create();  
collisionListener->onContactBegin = CC_CALLBACK_1(Level1::onContactBegin, this);  
_eventDispatcher->addEventListenerWithSceneGraphPriority(collisionListener, this);  
  
return true;
```

onContactBegin partie 1:

S'il y a une collision, le "onContact Begin" affectera au hasard un objet du SpriteA à l'autre objet du SpriteB.

Nous avons donné à chaque composant un tag (Ball :1, Obstacle :2, Undersquare :3) pour filtrer les collisions. La première instruction if déterminera si la balle entre en collision avec un obstacle alors que le niveau n'est pas terminé. Si cela se produit, la méthode "stopping" sera exécutée.

```

bool Level1::onContactBegin(cocos2d::PhysicsContact& contact) {

    auto spriteA = contact.getShapeA()->getBody()->getOwner();
    auto spriteB = contact.getShapeB()->getBody()->getOwner();

    Color3B clr(255, 255, 0);
    auto act = TintTo::create(0, clr);

    if (!won) {

        if (spriteA->getTag() == 1 && spriteB->getTag() == 2)
        {
            stoping(spriteB->getPosition());
        }
        else if (spriteA->getTag() == 2 && spriteB->getTag() == 1)
        {
            stoping(spriteA->getPosition());
        }
    }
}

```

La méthode "stopping":

La méthode Stopping est responsable de l'arrêt de la balle lorsqu'une collision se produit. Puisqu'il utilise la position de l'obstacle comme paramètre, il ajuste la position de la balle par rapport à l'obstacle en tenant compte également de la direction de la balle.

```

void Level1::stopping(cocos2d::Vec2 vvv) {

    onMouv = false;
    auto position = ball->getPosition();
    switch (ball->direction)
    {
        case 'l':
            unschedule(SEL_SCHEDULE(&Level1::mouvLeft));
            ball->setPosition(vvv.x + 100, position.y);
            break;

        case 'r':
            unschedule(SEL_SCHEDULE(&Level1::mouvRight));
            ball->setPosition(vvv.x - 100, position.y);
            break;

        case 'u':
            unschedule(SEL_SCHEDULE(&Level1::mouvUp));
            ball->setPosition(position.x, vvv.y - 100);
            break;

        case 'd':
            unschedule(SEL_SCHEDULE(&Level1::mouvDown));
            ball->setPosition(position.x, vvv.y + 100);
            break;
    }
}

```

onContactBegin partie 2:

Après avoir détecté la collision entre la balle et les underSquars, le reste de la méthode exécute l'action "act" pour changer la couleur des underSquars.

Dès que tous les underSquars sont colorés, les méthodes "winStopping" et "levelComplete" sont invoquées et la position de la balle est définie sur la même position que le dernier underSquar.

```
if (spriteA->getTag() == 3 && spriteB->getTag() == 1 )
{
    if (spriteA->getColor() != clr) {
        spriteA->runAction(act);
        coloredCount++;
    }
    if (coloredCount == 58)
    {
        won = true;
        ball->runAction(MoveTo::create( 0.03, spriteA->getPosition() ) );
        levelComplete();
    }
    log("%d", coloredCount);
}

else if(spriteA->getTag() == 1 && spriteB->getTag() == 3)
{
    if (spriteB->getColor() != clr) {
        spriteB->runAction(act);
        coloredCount++;
    }
    log("%d", coloredCount);

    if (coloredCount == 58)
    {
        won = true;
        winStopping();
        ball->runAction( MoveTo::create(0.03, spriteB->getPosition() ) );
        levelComplete();
    }
}

return true;
}
```

La méthode « winStoping » :

L'utilisation de cette méthode empêche la planification « sheduling » de toutes les méthodes de mouvement.

```
void Level1::winStoping() {  
    onMouv = false;  
    this->unschedule(SEL_SCHEDULE(&Level1::mouvLeft));  
    this->unschedule(SEL_SCHEDULE(&Level1::mouvRight));  
    this->unschedule(SEL_SCHEDULE(&Level1::mouvUp));  
    this->unschedule(SEL_SCHEDULE(&Level1::mouvDown));  
}
```

La méthode "levelComplete" :

La méthode crée un effet d'explosion de particules (pour célébrer la fin de chaque niveau) et un menu contenant deux flèches, une pour le niveau suivant et une pour le niveau précédent.

```
void Level2::levelComplete() {  
    auto particleSystem = ParticleExplosion::create();  
    this->addChild(particleSystem, 6);  
    particleSystem->setPosition(550, 1100);  
  
    auto menuBack = MenuItemImage::create("menuBack.png", "menuBack.png");  
    auto menu1 = Menu::create(menuBack, NULL);  
    menu1->setPosition(550, 1100);  
    this->addChild(menu1, 3);  
  
    auto congrats = MenuItemImage::create("congratulations.png", "congratulations.png");  
    auto menu2 = Menu::create(congrats, NULL);  
    menu2->setPosition(550, 1200);  
    this->addChild(menu2, 4);  
  
    auto next = MenuItemImage::create("arrowright.png", "arrowright.png",  
        CC_CALLBACK_1(Level2::goTONextLevel, this));  
    auto menu3 = Menu::create(next, NULL);  
    menu3->setPosition(650, 900);  
    this->addChild(menu3, 4);  
  
    auto pre = MenuItemImage::create("arrowleft.png", "arrowleft.png",  
        CC_CALLBACK_1(Level2::goTOPreviousLevel, this));  
    auto menu4 = Menu::create(pre, NULL);  
    menu4->setPosition(450, 900);  
    this->addChild(menu4, 4);  
}
```



Les autres niveaux :

De même, les autres niveaux ont les mêmes déclarations des méthodes dans l'en-tête, sauf que nous n'avons pas besoin de déclarer d'autres variables car "Level2" et "Level3" héritent de la classe "Level1".

La dernière Scène :

À la fin des trois niveaux, une OutroScene apparaît.



GESTION ET ORGANISATION DE TRAVAIL :

Pour réussir ce projet, tous les membres du groupe ont recherché et exploré des idées sur la façon dont nous pouvons atteindre notre objectif, puis nous nous sommes rencontrés pour discuter des idées et choisir les meilleures à utiliser dans le jeu, puis nous avons travaillé ensemble pour le créer.

Conclusion

En guise de conclusion, ce projet nous a permis de mettre en pratique nos connaissances, comme l'implémentation de la programmation orientée objet. Cela nous a également aidé à comprendre le fonctionnement des jeux vidéo, ainsi que l'efficacité de travail en équipe.

Lien de code source GitHub :

Lien Abdlghafour :

<https://github.com/AbdelghafourBoy/RollerSplate-Abd-El-Ghafour-El-Ballaoui-El-Houmaini-karim-.git>

Lien ElHoumaini Karim :

https://github.com/karimelhou/RollerSplat_CodeSource.git