RAPPORT DU JEU

PICO PARK

Réalisé par :

Imane Elaoufi

Houda El Asri

Encadré par :

Abdelouahab Ikrame

Mr. Lotfi El Aachak

Le document que vous avez sous vos yeux représente le rapport du projet final pour le module du POO en C++ qui été sous forme d'un jeu vidéo nommé « PICO PARK » créé en appliquant les connaissances requisse tout le long du module via le moteur COCOS2D-X.

Avant de commencer, nous tenons à remercier notre profs Ben Abdelouahab Ikrame et Mr.El Aachak Lotfi.

sjhcv

Table de contenu :

Préparation :

Avant d'entamer la partie saisie du code, il fallait faire quelques modifications au niveau du code source donné par défaut lors de la création d'un nouveau projet sur 'Cocos2d-x'.
La 1ier chose à modifier était la résolution de la fenêtre du jeu à travers le fichier AppDelegate.cpp a la ligne suivante :

```
glv ew->setFrameS ze(620, 800);
```

La résolution d'écran qui sera utilisé lors du jeu sera 6200 x 800 px.
Puis, désactiver l'affichage des statistiques du jeu (nombre des FPS …) sur l'écran à travers la ligne suivante :

```
d rector->setD splayStats(false);
```

Création des scènes :

Le jeu sera composé d'une scène Menu lancée au démarrage du jeu.Cette Scène va permettre de lancer le 1ier niveau etc…

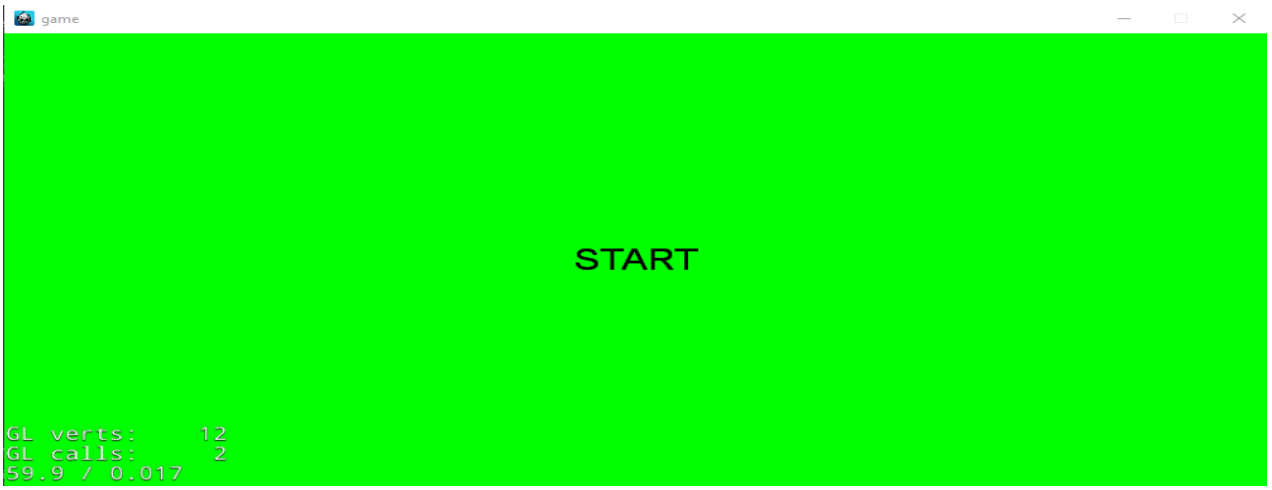
1- menu scène :

Le menu principal du jeu sera composé d'une arriéré plan et d'un seul bouton '**START** ' qui va permettre de commencer le premier Niveau du jeu.
L'arrière-plan est de type Sprite.
Cela est possible à l'aide d'une fonction **GoToLVL1** qui effectue le passagepar référence entre la scène du menu et celle du 1ier niveau :

```
void MainMenu::GoToLVL1(cocos2d::Ref* pSender)
{

    auto scene = MyWorld::createScene();
    Director::getInstance()->replaceScene(TransitionFade::create(0.5, scene));
}
```

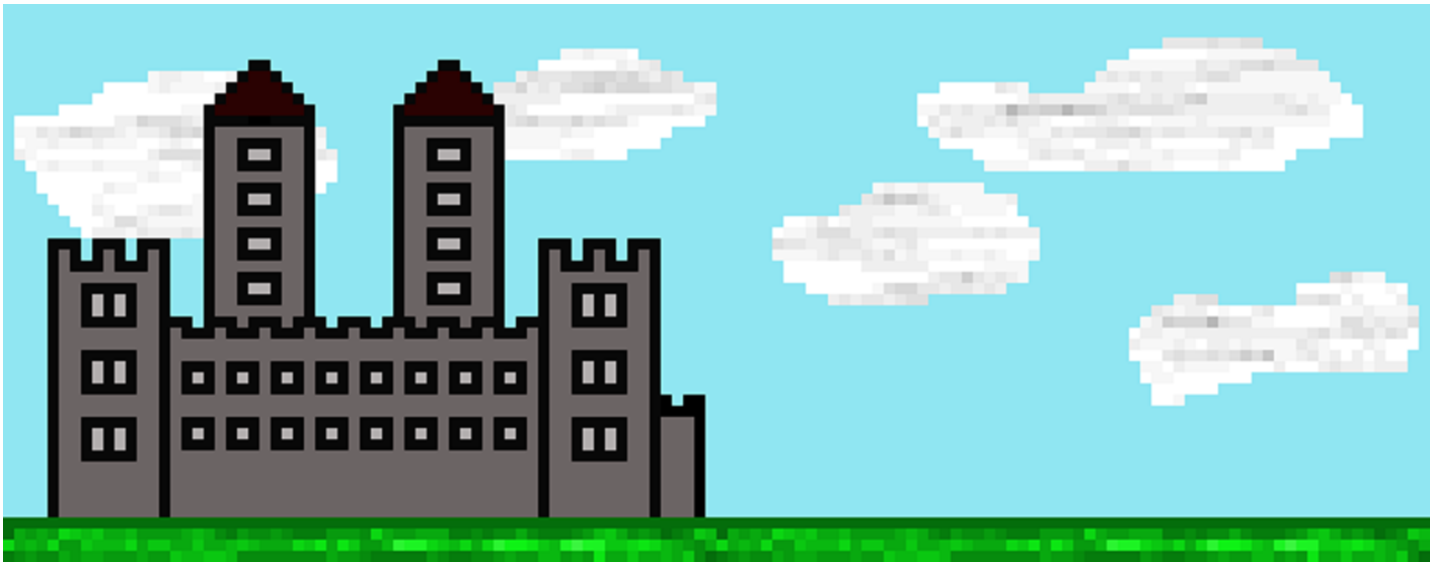
Le passage est effectué avec une animation de transition Fade :
TransitionFade::create(0.5, scene)

Lors de l'appui sur le bouton, la fonction précédente sera invoquée et permettra l'accès au niveau à l'aide de la commande :
CC_CALLBACK_1(MainMenu::GoToLVL1, this)



Niveau1 : Lvl1

Cette scène représente le premier niveau du jeu. La structure du niveau est la suivante :



Au niveau du fichier **player .cpp**,une classe nommée OurPlayer qui représente un personnage joueur dans un jeu ou une application. La classe a plusieurs méthodes qui permettent de contrôler le mouvement, la position et la rotation du joueur.

```
OurPlayer::OurPlayer(cocos2d::Layer* layer, bool jump, bool move, bool stand) {

        Pico = Sprite::create("person.png");
        Pico->setScale(0.05);
        Pico->setPosition(Point(origin.x + 100, origin.y + 100));

        auto  Playerbody = PhysicsBody::createBox(Size(Pico->getContentSize().width-200, Pico-
>getContentSize().height), PhysicsMaterial(0, 0, 0));
        Playerbody->setCollisionBitmask(1);
        Playerbody->setContactTestBitmask(true);
        Playerbody->setRotationEnable(false);
        Pico->setPhysicsBody(Playerbody);
        layer->addChild(Pico, 1);
};
```

Il définit ensuite la position du sprite à un point situé 100 points vers la droite et 100 points vers le bas par rapport à l'angle supérieur gauche de l'écran.

On a aussi trois méthodes, La méthode getposition renvoie la position actuelle du sprite du joueur sous forme d'objet Vec2. La méthode setposition définit la position du sprite sur le point spécifié. La méthode getrect renvoie la boîte englobante du sprite, qui est un rectangle qui englobe l'image du sprite. La méthode turnLeft fait pivoter le sprite autour de l'axe y selon l'angle spécifié. La méthode turnRight fait pivoter le sprite de nouveau vers sa position d'origine s'il est actuellement tourné vers la gauche.

```
cocos2d::Vec2 OurPlayer::getposition() {
        return Pico->getPosition();}
```

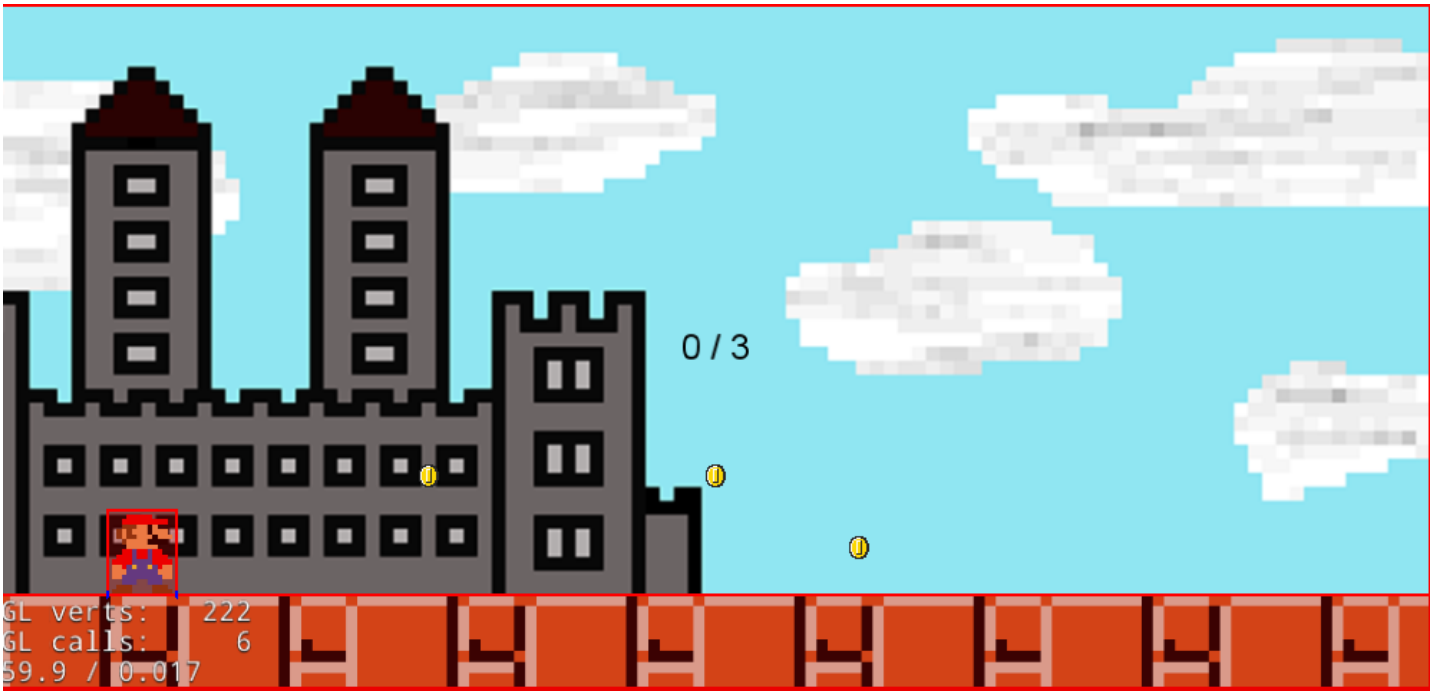
```
void  OurPlayer::setposition(float x, float y) {
	Pico->setPosition(Point(x, y));
}
cocos2d::Rect OurPlayer::getrect() {
	return Pico->getBoundingBox();
}
void OurPlayer::turnLeft(float z) {
	Pico->setRotation3D(Vec3(0,z,0));
}

void OurPlayer::turnRight()
{
	int x = Pico->getRotation3D().y ;
	if (x == 180)
	{
		Pico->setRotation3D(Vec3(0, 0, 0));
```

Les fonctions de ce code suivant sont utilisées pour déplacer le personnage joueur dans différentes directions. moveright et moveleft déplacent le personnage joueur horizontalement, tandis que movebot et movetop déplacent le personnage joueur verticalement. onContactBegin est une fonction qui est appelée chaque fois que deux corps de physique entrent en collision. Elle vérifie lesquels sont en train de collider et prend des mesures appropriées en fonction des masques de bits des corps en collision. update est une fonction qui est appelée de manière répétée et est utilisée pour mettre à jour l'état du jeu. Il gère les entrées de l'utilisateur et met à jour la position et l'orientation du personnage joueur en fonction de ces entrées. Il vérifie également les collisions entre le personnage joueur et d'autres objets et met à jour l'état du jeu en conséquence. Enfin, il vérifie si le joueur a gagné le jeu en collectant toutes les pièces et, dans ce cas, il passe au niveau suivant

```
void MyWorld::moveright(float dt) {
    Vec2 playerPos = player->getposition();
    player->setposition(playerPos.x + 80* dt, playerPos.y);
}
void MyWorld::moveleft(float dt) {
    Vec2 ballpos = player->getposition();
    player->setposition(ballpos.x - 80 * dt, ballpos.y);
}
void MyWorld::movebot(float dt) {
    Vec2 ballpos = player->getposition();
    player->setposition(ballpos.x, ballpos.y -80 * dt);
}
void MyWorld::movetop(float dt) {
    Vec2 ballpos = player->getposition();
    player->setposition(ballpos.x, ballpos.y + 160 * dt);
}

bool MyWorld::onContactBegin(cocos2d::PhysicsContact& contact)
{
    PhysicsBody* a = contact.getShapeA()->getBody();
    PhysicsBody* b = contact.getShapeB()->getBody();
    auto p = contact.getContactData()->points;

    // check if the bodies have collided
    if ((1 == a->getCollisionBitmask() && 2 == b->getCollisionBitmask()) || (2 == a->getCollisionBitmask() && 1 == b-
>getCollisionBitmask()))
    {
        ifSpacePressed = false;
        isOnGround = true;
    }
    if ((1 == a->getCollisionBitmask() && 3 == b->getCollisionBitmask()) || (3 == a->getCollisionBitmask() && 1 == b-
>getCollisionBitmask())) {
        if (ifDpressed)
        {
            this->unschedule(SEL_SCHEDULE(&MyWorld::moveright));
            player->setposition(player->getposition().x - 10, player->getposition().y);
        }
        else if (ifApressed)
        {
            this->unschedule(SEL_SCHEDULE(&MyWorld::moveleft));
            player->setposition(player->getposition().x + 10, player->getposition().y);
        }
    }



    return true;
}



void MyWorld::update(float dt) {
```

```
CCLOG("%i", count);

Rect rect1 = player->getrect();

Rect rect5 = wall->getBoundingBox();
auto v = Director::getInstance()->getWinSize();
Layer::setAnchorPoint(Vec2(player->getposition().x / v.width, player->getposition().y /v.height));
for (auto sprite : coins)
{
   Rect rect2 = sprite->getBoundingBox();

   if (rect1.intersectsRect(rect2))
   {
      sprite->removeFromParent();
      if (!sprite->isClaimed)
      {
         cocos2d::AudioEngine::preload("audio/key.mp3");
         cocos2d::AudioEngine::play2d("audio/key.mp3", false, 0.3f);
         sprite->isClaimed = true;
         count++;
         scoreLabel->setString(StringUtils::format("%d / 3", count));
       }
    }
}
```

Voici la scène complète :



Niveau2 : Lvl2

Cette scène représente le 2eme niveau du jeu. La structure du niveau est la suivante :

Dans cette scène, et au niveau du fichier Lvl2.cpp, ona une fonction pour crée et renvoie une nouvelle scène Cocos2d-x avec la physique activée. La gravité du monde de physique est définie sur (0, -400), ce qui signifie que les objets de la scène tombent vers le bas de l'écran. La fonction crée une nouvelle instance de la classe Lvl2 et l'ajoute en tant que couche enfant à la scène. La nouvelle scène est ensuite renvoyée.

```
Scene* Lvl2::createScene()
{
   auto scene = Scene::createWithPhysics();
   scene->getPhysicsWorld()->setGravity(Vec2(0, -400));
   auto layer = Lvl2::create();
   scene->addChild(layer);
   return scene;
 }
```

Dans la fonction Lvl2::createScene(), une nouvelle scène est créée avec un monde de physique. La gravité du monde de physique est définie sur (0, -400). Un nouveau niveau, Lvl2, est créé et ajouté à la scène. La scène est ensuite retournée.

Dans la fonction Lvl2::init(), si la couche de base ne s'initialise pas correctement, la fonction retourne false. Sinon, la couche crée un arrière-plan et un bord autour de la zone de jeu. Elle initialise également un compteur et un label affichant ce compteur. Ensuite, la couche crée un sol en utilisant un sprite et un corps physique. Elle crée également un deuxième sol de la même manière. Un nouvel objet de la classe OurPlayer est créé et ajouté à la couche. Trois pièces sont également créées et ajoutées à la couche. Enfin, la couche démarre une mise à jour périodique et un écouteur d'événement clavier.

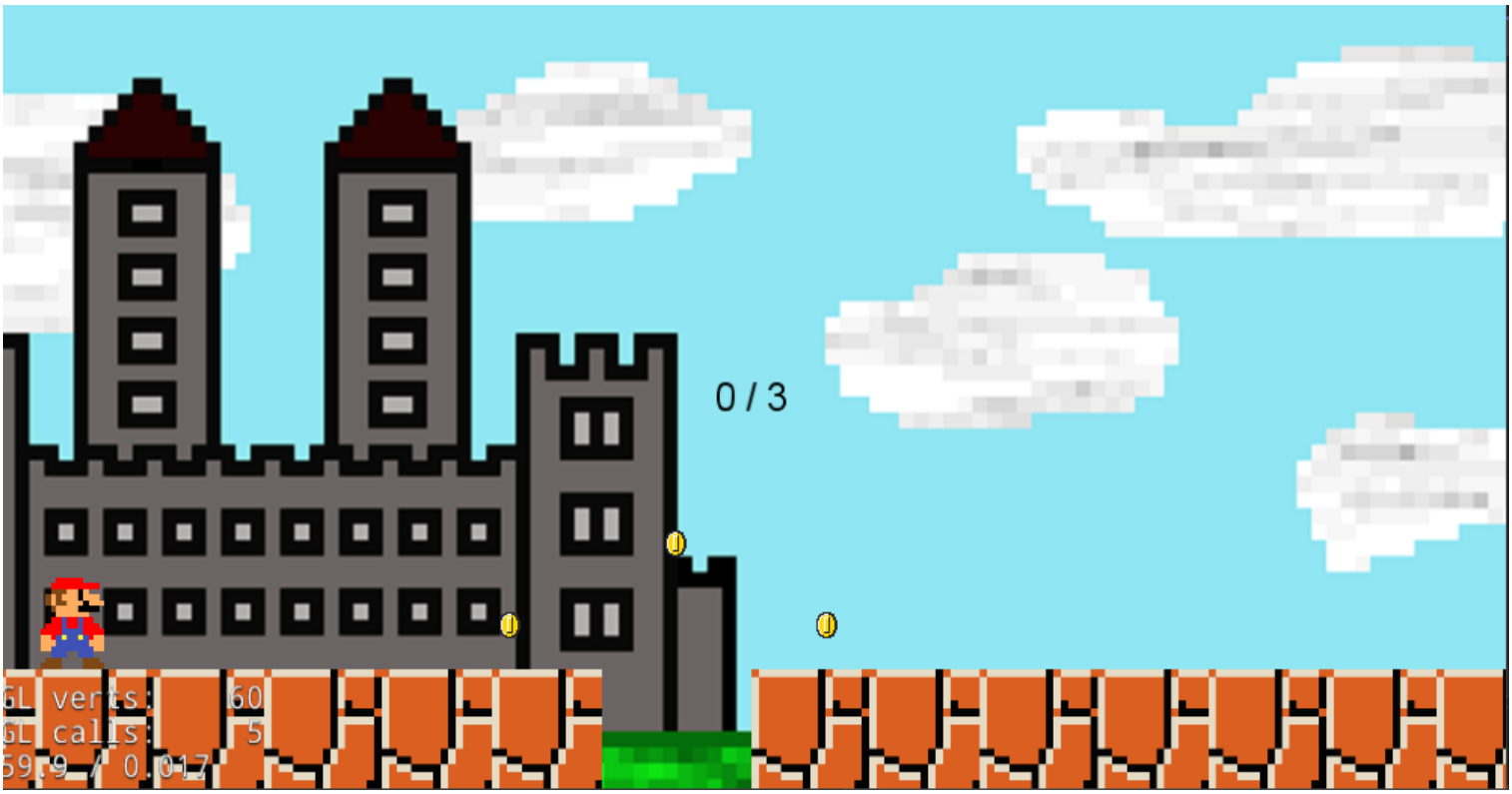
```
bool Lvl2::init()
{
   if (!Layer::init())
   {
      return false;
   }

   auto visibleSize = Director::getInstance()->getWinSize();

   Vec2 origin = Director::getInstance()->getVisibleOrigin();
   auto background = Sprite::create("Background.png");
   background->setScale(1);
   background->setPosition(visibleSize.width / 2, visibleSize.height / 2);
   this->addChild(background, -10);
```

```
    text = StringUtils::format("%d / 3", count);

    scoreLabel = Label::createWithTTF(text, "fonts/arial.ttf", 24);
    scoreLabel->setTextColor(Color4B::BLACK);
    scoreLabel->setPosition(visibleSize.width / 2 + origin.x, visibleSize.height / 2 + origin.y);
    this->addChild(scoreLabel, 10000);
    auto edgebody = PhysicsBody::createEdgeBox(visibleSize, PHYSICSBODY_MATERIAL_DEFAULT, 3);
    edgebody->setCollisionBitmask(3);
    edgebody->setContactTestBitmask(true);
    auto edgeNode = Node::create();
    edgeNode->setPosition(Point((visibleSize.width / 2) , (visibleSize.height / 2)));
    edgeNode->setPhysicsBody(edgebody);
    this->addChild(edgeNode);
    wall = Sprite::create("ground.png");
    wall->setScale(0.5);
    auto Frame4body = PhysicsBody::createBox(Size(wall->getContentSize().width, wall->getContentSize().height), PhysicsMaterial(0, 0,
0));
    Frame4body->setCollisionBitmask(2);
    Frame4body->setContactTestBitmask(true);
    wall->setAnchorPoint(Vec2());
    wall->setPosition(Point( origin.x-580,  origin.y));
    if (Frame4body != nullptr)
    {
        Frame4body->setDynamic(false);
        wall->setPhysicsBody(Frame4body);
    }
    this->addChild(wall);

    auto gound2 = Sprite::create("ground.png");
    gound2->setScale(0.5);
    auto gound2body = PhysicsBody::createBox(Size(gound2->getContentSize().width, gound2->getContentSize().height),
PhysicsMaterial(0, 0, 0));
    gound2body->setCollisionBitmask(2);
    gound2body->setContactTestBitmask(true);
    gound2->setAnchorPoint(Vec2());
    gound2->setPosition(Point(origin.x + 500, origin.y ));
    if (gound2body != nullptr)
    {
        gound2body->setDynamic(false);
        gound2->setPhysicsBody(gound2body);
    }
    this->addChild(gound2);

    player = new OurPlayer(this,!OnGround,imoving,!imoving);
    player->setposition(origin.x+50,origin.y+100);

    for (int j = 1; j <= 3; j++)
    {
        auto _sprite = coin::coininit();

        {
            if (j == 1)
            {
                log("%d", j);
                _sprite->setPosition(Vec2(origin.x + 240 + j * 100, origin.y + 100));
            }
            else if(j==2)
            {
                _sprite->setPosition(Vec2(origin.x + 250 + j * 100, origin.y + 150));
            }
            else
            {
                _sprite->setPosition(Vec2(origin.x + 250 + j * 100, origin.y + 100));
            }
        }
        this->addChild(_sprite, 1);
        coins.push_back(_sprite);
    }
this->scheduleUpdate();
    auto eventListener = EventListenerKeyboard::create();
```

```
  eventListener->onKeyPressed = [=](EventKeyboard::KeyCode keyCode, Event* event) {
   switch (keyCode) {
            //case wich key is pressed
          case EventKeyboard::KeyCode::KEY_LEFT_ARROW:
          case EventKeyboard::KeyCode::KEY_A:
         isApressed = true;
               this->schedule(SEL_SCHEDULE(&Lvl2::moveleft), 0.01);

            break;
          case EventKeyboard::KeyCode::KEY_RIGHT_ARROW:
          case EventKeyboard::KeyCode::KEY_D:

            isDpressed = true;
               this->schedule(SEL_SCHEDULE(&Lvl2::moveright), 0.01);

            break;
          case EventKeyboard::KeyCode::KEY_SPACE:
          case EventKeyboard::KeyCode::KEY_W:
            if (isSpacePressed == false)
            {
               cocos2d::AudioEngine::preload("audio/jump.mp3");
               cocos2d::AudioEngine::play2d("audio/jump.mp3", false, 0.3f);
               this->schedule(SEL_SCHEDULE(&Lvl2::movetop), 0.01);
               isSpacePressed = true;
               OnGround = false;
            }
           break;
          case EventKeyboard::KeyCode::KEY_UP_ARROW:

            isUpPressed = true;

            break;

         }
};
  eventListener->onKeyReleased = [=](EventKeyboard::KeyCode keyCode, Event* event)
  {

         switch (keyCode) {
            //case wich key is pressed
          case EventKeyboard::KeyCode::KEY_LEFT_ARROW:
          case EventKeyboard::KeyCode::KEY_A:

            isApressed = false;
               this->unschedule(SEL_SCHEDULE(&Lvl2::moveleft));
               cocos2d::AudioEngine::pauseAll();

            break;
          case EventKeyboard::KeyCode::KEY_RIGHT_ARROW:
          case EventKeyboard::KeyCode::KEY_D:

            isDpressed = false;
               this->unschedule(SEL_SCHEDULE(&Lvl2::moveright));
               cocos2d::AudioEngine::pauseAll();

            break;
          case EventKeyboard::KeyCode::KEY_SPACE:
          case EventKeyboard::KeyCode::KEY_W:

               this->unschedule(SEL_SCHEDULE(&Lvl2::movetop));

            break;
          case EventKeyboard::KeyCode::KEY_DOWN_ARROW:
          case EventKeyboard::KeyCode::KEY_S:

               this->unschedule(SEL_SCHEDULE(&Lvl2::movebot));

            break;
          case EventKeyboard::KeyCode::KEY_UP_ARROW:
```

```
            isUpPressed = false;

            break;

        }
};

this->_eventDispatcher->addEventListenerWithSceneGraphPriority(eventListener, this);
//we create a contact listener to detect the collision of the ball with te walls and the obstacles
auto contactListener = EventListenerPhysicsContact::create();
//make a call back to the function everytime a contact happen
contactListener->onContactBegin = CC_CALLBACK_1(Lvl2::onContactBegin, this);
this->getEventDispatcher()->addEventListenerWithSceneGraphPriority(contactListener, this);
return true;
```

Voici la scene complete:



Niveau3 : Lvl3

Le code suivant crée une nouvelle scène dans un moteur de jeu et configure ses propriétés de physique. La scène est créée avec un monde de physique, et la gravité est réglée sur (0, -400) ce qui signifie que les objets de la scène ressentiront une force descendante de 400 unités par seconde au carré. Le masque de dessin de débogage est également défini sur DEBUGDRAW_ALL, ce qui signifie que le moteur de physique affichera des informations de débogage pour tous les objets de physique de la scène.

Ensuite, la classe Lvl3 est créée et ajoutée en tant qu'enfant à la scène. Enfin, la scène est renvoyée à l'appelant.

```
Scene* Lvl3::createScene()
{
    auto scene = Scene::createWithPhysics();
    scene->getPhysicsWorld()->setDebugDrawMask(PhysicsWorld::DEBUGDRAW_ALL);
    scene->getPhysicsWorld()->setGravity(Vec2(0, -400));
    //scene->getPhysicsWorld()->setDebugDrawMask(0xffff);
    auto layer = Lvl3::create();
    scene->addChild(layer);
    return scene;
}
```

Ce code initialise une couche dans un moteur de jeu. La fonction init initialise la couche et configure divers éléments du jeu tels qu'une image de fond, une boîte de bordure pour le monde de jeu, un label de score et un sprite de mur.

La fonction init vérifie d'abord si la fonction init de la classe de base a réussi, et si ce n'est pas le cas, elle renvoie false. Sinon, elle continue à configurer la couche.

La taille visible de la fenêtre de jeu est déterminée et un sprite de fond est créé et ajouté à la couche. Ensuite, une boîte de bordure est créée à l'aide d'un corps physique et ajoutée à la couche en tant que nœud. La boîte de bordure sert de limites au monde de jeu.

Un label de score est créé à l'aide d'un label avec une police TrueType et ajouté à la couche. Enfin, un sprite de mur est créé et son échelle est définie.
Ce code configure divers éléments de jeu tels que des sprites et des corps de physique, et les ajoute à la couche en tant qu'enfants. Il met également en place des écouteurs d'événements pour l'entrée au clavier et planifie des fonctions de mise à jour pour être appelées à des intervalles réguliers.

Tout d'abord, trois sprites de mur sont créés et leur échelle est définie. Chaque sprite de mur est doté d'un corps physique en forme de boîte, et les corps de physique sont définis pour ne pas être dynamiques (c'est-à-dire non affectés par des forces). Les sprites de mur sont ensuite positionnés et ajoutés à la couche.

Ensuite, un objet joueur et deux objets ennemis sont créés et ajoutés à la couche. Un vecteur de sprites de pièces est également initialisé, et trois sprites de pièces sont créés et ajoutés au vecteur et à la couche.

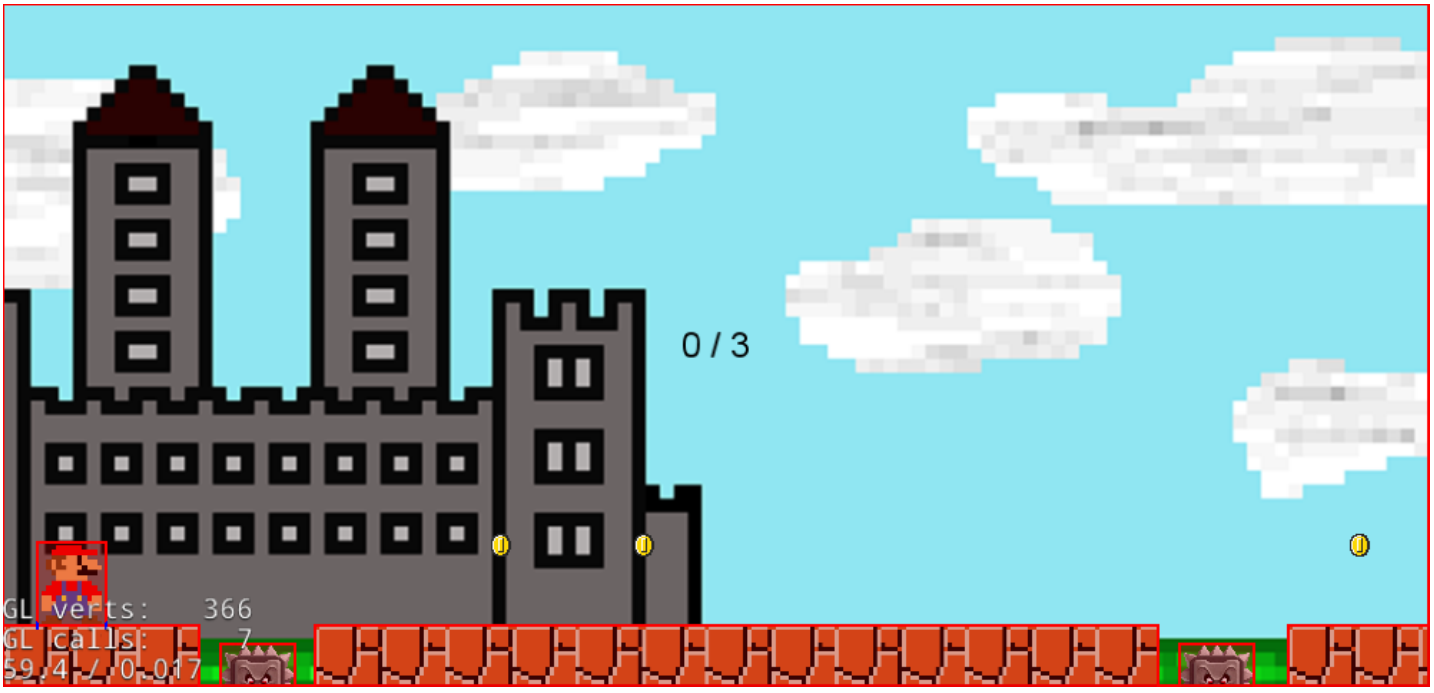
```
bool Lvl3::init()
{
    if (!Layer::init())
    {
        return false;
    }



    //----------------------------------------------create variables for positioning our instance-----------------------------------------------------

    auto visibleSize = Director::getInstance()->getWinSize();

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

    //-------------------------------------------------------------create backgroud color-------------------------------------------------

    auto background = Sprite::create("Background.png");
    background->setScale(1);
    background->setPosition(visibleSize.width / 2, visibleSize.height / 2);
    this->addChild(background, -10);
    //-------------------------------------------------create an edge box for our game-----------------------------------------------------

    auto edgebody = PhysicsBody::createEdgeBox(visibleSize, PHYSICSBODY_MATERIAL_DEFAULT, 3);
    edgebody->setCollisionBitmask(3);
    edgebody->setContactTestBitmask(true);
    auto edgeNode = Node::create();
    edgeNode->setPosition(Point((visibleSize.width / 2) , (visibleSize.height / 2)));
    edgeNode->setPhysicsBody(edgebody);
    this->addChild(edgeNode);


    text = StringUtils::format("%d / 3", count);

    scoreLabel = Label::createWithTTF(text, "fonts/arial.ttf", 24);
    scoreLabel->setTextColor(Color4B::BLACK);
    scoreLabel->setPosition(visibleSize.width / 2 + origin.x, visibleSize.height / 2 + origin.y);
    this->addChild(scoreLabel, 10000);


    wall = Sprite::create("ground.png");
    wall->setScale(0.3);
    auto Frame4body = PhysicsBody::createBox(Size(wall->getContentSize().width, wall->getContentSize().height), PhysicsMaterial(0, 0,
0));
    Frame4body->setCollisionBitmask(2);
    Frame4body->setContactTestBitmask(true);
    wall->setAnchorPoint(Vec2());
    wall->setPosition(Point( origin.x-450,  origin.y));
    if (Frame4body != nullptr)
    {
        Frame4body->setDynamic(false);
        wall->setPhysicsBody(Frame4body);
    }
```

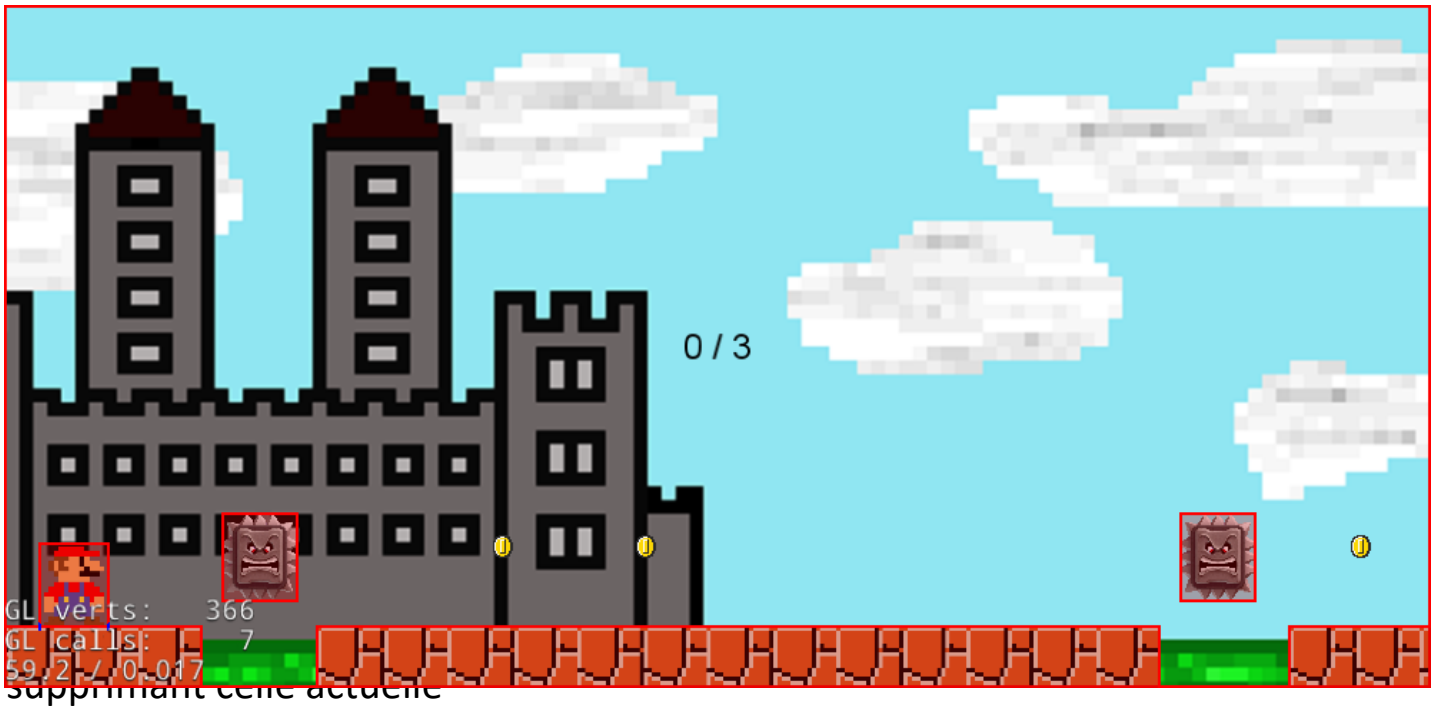
```
    this->addChild(wall);


    wall2 = Sprite::create("ground.png");
    wall2->setScale(0.3);
    auto Frame2body = PhysicsBody::createBox(Size(wall2->getContentSize().width, wall2->getContentSize().height), PhysicsMaterial(0, 0,
0));
    Frame2body->setCollisionBitmask(2);
    Frame2body->setContactTestBitmask(true);
    wall2->setAnchorPoint(Vec2());
    wall2->setPosition(Point(origin.x + 220, origin.y));
    if (Frame2body != nullptr)
    {
        Frame2body->setDynamic(false);
        wall2->setPhysicsBody(Frame2body);
    }
    this->addChild(wall2);

    wall3 = Sprite::create("ground.png");
    wall3->setScale(0.3);

        auto Frame3body = PhysicsBody::createBox(Size(wall3->getContentSize().width, wall3->getContentSize().height),
PhysicsMaterial(0, 0, 0));
    Frame3body->setCollisionBitmask(2);
    Frame3body->setContactTestBitmask(true);
    wall3->setAnchorPoint(Vec2());
    wall3->setPosition(Point(origin.x + 900, origin.y));
    if (Frame3body != nullptr)
    {
        Frame3body->setDynamic(false);
        wall3->setPhysicsBody(Frame3body);
    }
    this->addChild(wall3);


    player = new OurPlayer(this,!Ground , moving,!moving);
    player->setposition(origin.x + 50 , origin.y + 100);

    enemy = new Enemy(this);
    enemy2 = new Enemy(this);
    enemy2->setposition(origin.x + 850, origin.y + 100);

    for (int j = 1; j <= 3; j++)
    {
        auto _sprite = coin::coininit();

        {
            if (j == 1)
            {
                log("%d", j);
                _sprite->setPosition(Vec2(origin.x + 250 + j * 100, origin.y + 100));
            }
            else if (j == 2)
            {
                _sprite->setPosition(Vec2(origin.x + 250 + j * 100, origin.y + 100));
            }
            else
            {
                _sprite->setPosition(Vec2(origin.x + 650 + j * 100, origin.y + 100));
            }
        }



        this->addChild(_sprite, 1);
        coins.push_back(_sprite);
    }

    this->scheduleUpdate();
```

```
auto eventListener = EventListenerKeyboard::create();

eventListener->onKeyPressed = [=](EventKeyboard::KeyCode keyCode, Event* event) {

	switch (keyCode) {
	  //case wich key is pressed
	case EventKeyboard::KeyCode::KEY_LEFT_ARROW:
	case EventKeyboard::KeyCode::KEY_A:

	  Apressed = true;

	    this->schedule(SEL_SCHEDULE(&Lvl3::moveleft), 0.01);

	  break;
	case EventKeyboard::KeyCode::KEY_RIGHT_ARROW:
	case EventKeyboard::KeyCode::KEY_D:

	  Dpressed = true;

	    this->schedule(SEL_SCHEDULE(&Lvl3::moveright), 0.01);

	  break;
	case EventKeyboard::KeyCode::KEY_SPACE:
	case EventKeyboard::KeyCode::KEY_W:
	  if (SpacePressed == false)
	  {
	    this->schedule(SEL_SCHEDULE(&Lvl3::movetop), 0.01);
	    SpacePressed = true;
	    Ground = false;
	    cocos2d::AudioEngine::preload("audio/jump.mp3");
	    cocos2d::AudioEngine::play2d("audio/jump.mp3", false, 0.3f);
	  }




	  break;

	case EventKeyboard::KeyCode::KEY_UP_ARROW:

	  UpPressed = true;

	  break;

	}




};
eventListener->onKeyReleased = [=](EventKeyboard::KeyCode keyCode, Event* event)
{

	switch (keyCode) {
	  //case wich key is pressed
	case EventKeyboard::KeyCode::KEY_LEFT_ARROW:
	case EventKeyboard::KeyCode::KEY_A:

	  Apressed = false;
	    this->unschedule(SEL_SCHEDULE(&Lvl3::moveleft));

	    cocos2d::AudioEngine::pauseAll();

	  break;
	case EventKeyboard::KeyCode::KEY_RIGHT_ARROW:
```

```
        case EventKeyboard::KeyCode::KEY_D:

            Dpressed = false;
                this->unschedule(SEL_SCHEDULE(&Lvl3::moveright));

                cocos2d::AudioEngine::pauseAll();
            break;
        case EventKeyboard::KeyCode::KEY_SPACE:
        case EventKeyboard::KeyCode::KEY_W:

                this->unschedule(SEL_SCHEDULE(&Lvl3::movetop));

            break;
        case EventKeyboard::KeyCode::KEY_DOWN_ARROW:
        case EventKeyboard::KeyCode::KEY_S:

                this->unschedule(SEL_SCHEDULE(&Lvl3::movebot));

            break;
        case EventKeyboard::KeyCode::KEY_UP_ARROW:

            UpPressed = false;

            break;

        }
};

this->_eventDispatcher->addEventListenerWithSceneGraphPriority(eventListener, this);
//we create a contact listener to detect the collision of the ball with te walls and the obstacles
auto contactListener = EventListenerPhysicsContact::create();
//make a call back to the function everytime a contact happen
contactListener->onContactBegin = CC_CALLBACK_1(Lvl3::onContactBegin, this);
this->getEventDispatcher()->addEventListenerWithSceneGraphPriority(contactListener, this);
return true;
```

Voici la scene complete:





supprimant celle actuelle

Réessayer :

```
auto retryItem =
        MenuItemImage::create("Retry.png",
                "Retry.png",
                CC_CALLBACK_1(flauseMenu::Retry, th s));
vo d flauseMenu::Retry(cocos2d::Ref* pSender)
{
    auto scene = GameScreen::createScene();
    D rector::getInstance()->popScene();
    D rector::getInstance()->replaceScene(scene); }
```

La fonction précédente permet de revenir relancer le niveau 1 en remplaçant la scène actuelle avec celle du niveau 1

Revenir au menu d'accueil :

```
auto ma nMenuItem =
        MenuItemImage::create("Ma nMenu.png",
                "Ma nMenu.png",
                CC_CALLBACK_1(flauseMenu::GoToMa nMenuScene, th s));
vo d flauseMenu::GoToMa nMenuScene(cocos2d::Ref* pSender)
{
    auto scene = Ma nMenu::createScene();
    D rector::getInstance()->popScene();
```

```
    D rector::getInstance()->replaceScene(scene);
}
Tous ces éléments de menu ont été organ sés avec :
menu->al gnItemsVert callyW thfladd ng(v s bleS ze.he ght / 16);
    th s->addCh ld(menu);
```

Game Over Scène :

Ce code définit une classe de couche pour un écran de fin de jeu dans un jeu. La couleur de fond de la couche est définie sur un bleu pâle, et un sprite avec l'image "gameOver.png" est ajouté au centre de l'écran et redimensionné pour être 1,5 fois plus grand que sa taille originale. Cette couche sera utilisée pour afficher l'écran de fin de jeu au joueur lorsque le jeu est terminé.

```
#include "GameOver.h"
#include "AudioEngine.h"

USING_NS_CC;
Scene* GameOver::createScene()
{

    auto scene = Scene::createWithPhysics();
    //scene->getPhysicsWorld()->setDebugDrawMask(PhysicsWorld::DEBUGDRAW_ALL);
    scene->getPhysicsWorld()->setGravity(Vec2(0, 0));

    auto layer = GameOver::create();
    scene->addChild(layer);
    return scene;
}

// Print useful error message instead of segfaulting when files are not there.
static void problemLoading(const char* filename)
{
    printf("Error while loading: %s\n", filename);
    printf("Depending on how you compiled you might have to add 'Resources/' in front of filenames in HelloWorldScene.cpp\n");
}

// on "init" you need to initialize your instance
bool GameOver::init()
{
    //////////////////////////////
    // 1. super init first
    if ( !Layer::init() )
    {
        return false;
    }
    LayerColor* _bgColor = LayerColor::create(Color4B(206, 248, 252, 255));
    this->addChild(_bgColor, -10);
    auto visibleSize = Director::getInstance()->getVisibleSize();
    Vec2 origin = Director::getInstance()->getVisibleOrigin();
    auto youwin = Sprite::create("gameOver.png");
    youwin->setPosition(Point((visibleSize.width/2) + origin.x, (visibleSize.height / 2) + origin.y));
    youwin->setScale(1.5);
    this->addChild(youwin);
```

Les difficutes :

Développement du code: le développement de ce jeu est complexe et prendre du temps, surtout le jeu est ambitieux et comporte de nombreuses fonctionnalités.

Gestion du temps: créerce jeu a prendre beaucoup de temps, et il est difficile de trouver un équilibre entre le travail sur le jeu et les autres responsabilités.

Gestion de projet: le jeu difficile de gérer efficacement un projet de jeu, en particulier on travaillez en équipe et que on devons coordonner les efforts de plusieurs personnes.

Test et débogage: il est important de tester et de déboguer soigneusement un jeu pour s'assurer qu'il fonctionne de manière fluide et sans erreur, mais cela peut être une tâche chronophage.