Rapport du projet
(Jeu Vidéo 2D)

Réalisé par :

- BOULBEN Firdaous (Grp 1)
- CHIBANI Fahd (Grp 2)

Encadré par :

- Pr. ELAACHAk Lotfi
- Pr. BENABDELOUAHAB Ikram

SOMMAIRE

Sommaire 1

Liste des figures 2

Introduction 3

Outils 3

Le processus du développement et les options développées 4

1. Main Menu Scene 4
2. World Mode 6
   - Select Level Scene 6
   - Levels Scenes 7
   - Level Up Scenes 12
3. Endless Mode 13
4. Pause Scenes 16
5. Game Over Scenes 18

Conclusion 19

Difficultés rencontrées 19

Répartition du travail entre le groupe 19

Annexes 20

Code source du jeu 20

Vidéo démonstrative du jeu 20

LISTE DES FIGURES

Figure 1: Main Menu Scene 4

Figure 2: Select Level Scene 6

Figure 3: Level 1 Scene 9

Figure 4: Level 2 Scene 10

Figure 5: Level 3 Scene 11

Figure 6: Level 4 Scene 11

Figure 7: Level Up 1 Scene 13

Figure 8: Level Up 2 Scene 13

Figure 9: Level Up 3 Scene 13

Figure 10: Level Up 4 Scene 13

Figure 11: Game Scene 13

Figure 12: Pause Scene 16

Figure 13: Game Over Scene 18

INTRODUCTION

L'objectif principal de ce projet est de maitriser la programmation orientée objet par la mise en place d'un jeu vidéo en utilisant Cocos2d-x, un moteur de jeu open source qui permet le développement de jeux en 2D pour les appareils mobiles Android, iOS et Windows Phone, il compile sur Windows, Mac et Linux. Il gère notamment les sprites, les actions, les animations, les particules, les transitions, les timers, les événements (toucher, clavier, accéléromètre, souris), et le son.

Le jeu développé s'appelle PICO PARK. C'est un jeu de type Platformer Puzzle Game, constitué par deux modes unique de jeu :

- World Mode
- Endless Mode

et

Le premier mode est un mode de niveaux (4 niveaux développés) qui consiste à atteindre un but précis (la porte) en évitant les nombreux obstacles qui se dressent sur la route, afin de compléter le niveau et passer au niveau suivant.

Le second mode est un mode sans fin qui consiste tout simplement à s'échapper des bombes en mouvement afin de pouvoir continuer le jeu.

Outils

- C++ avec le respect du paradigme POO.
- Moteur de jeu Cocos2D.

LE PROCESSUS DU DÉVELOPPEMENT ET LES OPTIONS DÉVELOPPÉES

Afin de développer notre jeu vidéo, on avait besoin de créer plusieurs scènes, qui sont les différents statuts que le jeu peut avoir, et ajouter ensuite des méthodes permettant la navigation d'une scène vers une autre. Les scènes créées sont les suivants :

1. Main Menu Scene

Figure 1: Main Menu Scene

C'est la première scène affichée lorsque le jeu est démarré, qui représente le menu principal du jeu et est composée principalement de :

Menu :
C'est une technique utilisée pour afficher les différents boutons permettant la navigation entre les scènes. Dans cette scène on crée un menu composé de trois élément : le premier permet l'affichage du nom de jeu, le deuxième permet la navigation vers la mode du jeu "World Mode", et le troisième vers le mode du jeu "Endless Mode". Et ce en utilisant le code suivant :

```
auto menuTitle = MenuItemImage::create("Game_Title.png", "Game_Title.png");

auto mode1 = MenuItemImage::create("World_Mode.png", "World_Mode.png",
CC_CALLBACK_1(MainMenu::GoToSelectLevelScene, this));
auto mode2 = MenuItemImage::create("Endless_Mode.png", "Endless_Mode.png",
CC_CALLBACK_1(MainMenu::GoToGameScene, this));

auto menu = Menu::create(menuTitle, mode1, mode2, NULL);
menu->setPosition(visibleSize.width / 4, visibleSize.height / 4);
menuTitle->setPosition(Point(visibleSize.width / 4, visibleSize.height / 2));
mode1->setPosition(Point(visibleSize.width / 12, visibleSize.height / 6));
mode2->setPosition(Point(visibleSize.width / 2.4, visibleSize.height / 6));

this->addChild(menu);
```

Sprites :
La seule sprite qu'on a besoin dans la scène de menu principale est celle de l'arrière-plan, pour cela, on crée d'abord une variable sprite nommée "backgroundSprite", on l'initialise à partir d'une image PNG, on la ensuite positionne dans le centre de l'écran visible, et on l'ajoute enfin à notre scène afin de l'afficher à l'écran. Et ce en ajoutant le code suivant dans la fonction init():

```
auto backgroundSprite = Sprite::create("Background.png");
backgroundSprite->setPosition(Point((visibleSize.width / 2) + origin.x,
(visibleSize.height / 2) + origin.y));
this->addChild(backgroundSprite, -1);
```

Transitions :
On ajoute à notre scène une transition qui permet de jouer une animation lorsqu'on passe à l'une des deux scènes après le clic sur les boutons de menu afin de rendre le jeu plus dynamique. La transition qu'on a choisie est la transition "Fade" et on l'a appliqué en utilisant le code suivant :

```
void MainMenu::GoToSelectLevelScene(Ref* pSender)
{
    auto scene = SelectLevel::createScene();
    Director::getInstance()->replaceScene(TransitionFade::create(1.0, scene));
}
```

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

Audio :
On ajoute un effet sonore pour lire un fichier court, après avoir cliqué sur les deux boutons qui nous transmettre vers les autres scènes, en utilisant la line de code suivante dans les deux dernières fonctions :

```
CocosDenshion::SimpleAudioEngine::getInstance()->playEffect("audios/Play.mp3");
```

2. World Mode

- **Select Level Scene**

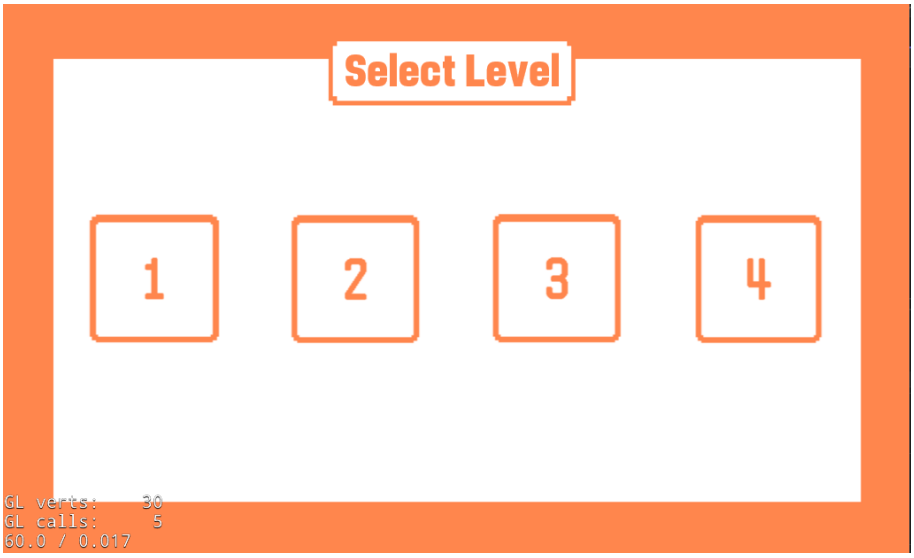

Figure 2: Select Level Scene

C'est la scène affichée lorsque le joueur choisi de démarrer le jeu en "World Mode" en lui permettant de choisir un niveau parmi les 4 niveaux disponibles. Elle est constituée d'une sprite de l'arrière-plan et un menu composé de 4 éléments, chacun d'eux amène à la scène de niveau correspondant:

```
auto Item1 = MenuItemImage::create("Level1.png", "Level1(Click).png",
CC_CALLBACK_1(SelectLevel::GoToLevel1Scene, this));
auto Item2 = MenuItemImage::create("Level2.png", "Level2(Click).png",
CC_CALLBACK_1(SelectLevel::GoToLevel2Scene, this));
auto Item3 = MenuItemImage::create("Level3.png", "Level3(Click).png",
CC_CALLBACK_1(SelectLevel::GoToLevel3Scene, this));
auto Item4 = MenuItemImage::create("Level4.png", "Level4(Click).png",
CC_CALLBACK_1(SelectLevel::GoToLevel4Scene, this));

auto menu = Menu::create(Item1, Item2, Item3,Item4, NULL);
menu->alignItemsHorizontallyWithPadding(visibleSize.height / 8);
this->addChild(menu);
```

Ci-dessous un exemple de la fonction responsable de cette navigation entre la scène actuelle et celle de niveau choisi, il suffit de changer le nom de la fonction et de la classe pour développer les autres fonctions.

```
void SelectLevel::GoToLevel1Scene(cocos2d::Ref* pSender)
{
    CocosDenshion::SimpleAudioEngine::getInstance()->
playEffect("audios/SelectLevel.mp3");
    auto scene = Level1::createScene();
    Director::getInstance()->replaceScene(scene);
}
```

- **Levels Scenes**

Chaque niveau est constitué d'une série d'obstacles à surmonter (avec des degrés de difficultés différentes) pour atteindre l'objectif et passer au niveau supérieur. Pour chacun, on a développé les options suivantes:

Menu :
Un menu composé de deux éléments, l'un joue le rôle d'un libellé indiquant le numéro de niveau courant, et l'autre est un bouton permettant de mettre en pause la scène du jeu en appelant la fonction suivante :

```
void Level1::GoToPause1Scene(cocos2d::Ref* pSender)
{
    auto scene = Pause1Menu::createScene();
    Director::getInstance()->pushScene(scene);
}
```

On utilise la méthode pushScene() qui pousse une scène particulière (la scène de pause) tout en conservant la scène en cours (la scène du jeu), mais interrompant son exécution.

Sprites :
On a 5 catégorie de sprite

1. Arrière-plan (background) : L'arrière-plan affichées dans les différentes scènes de jeu.
2. Joueur (player) : Le caractère principal du jeu auquel on attribue le masque `BITMASK_PLAYER 0x0001`
3. Obstacles (obstacles, bombs, enemies) : Des pièges qui tue le joueur et le place en situation d'échec (Game Over) auxquels on attribue le masque `BITMASK_OBSTACLE 0x0002`
4. Objectif (door) : Ce que le joueur doit atteindre après avoir avancé dans le jeu et surmonté toutes les obstacles. On l'attribue le masque `BITMASK_DOOR 0x0004`
5. Plateformes (ground, scale, road) : auxquels on attribue le masque `BITMASK_SCALE 0x0008`

Actions :
On trouve notamment le contrôle de personnage (déplacer et sauter en avant et en arrière) afin qu'il puisse exercer un impact sur le monde du jeu. Et ce en utilisant le code suivant :

```
auto keyboardListener = EventListenerKeyboard::create();
keyboardListener->onKeyPressed = [playerSprite, visibleSize]
(EventKeyboard::KeyCode keyCode, Event* event)
{
    Vec2 loc = event->getCurrentTarget()->getPosition();
    switch (keyCode)
    {
    case EventKeyboard::KeyCode::KEY_UP_ARROW:
        //Vérifier pour empêcher le joueur de sortir de l'écran visible
        if (playerSprite->getPositionX() >= visibleSize.width - (playerSprite->
getContentSize().width / 2)) {
            playerSprite->setPositionX(visibleSize.width - (playerSprite->
getContentSize().width / 2));
        }
        else {
            event->getCurrentTarget()->runAction(JumpBy::create(0.5, Vec2(100,
0), 100, 1));
            playerSprite->setTexture("Jump_right.png");
```

```
        }
        break;
    case EventKeyboard::KeyCode::KEY_LEFT_ARROW:
        if (playerSprite->getPositionX() <= 0 + (playerSprite->
getContentSize().width / 2)) {
            playerSprite->setPositionX(playerSprite->getContentSize().width / 2);
        }
        else {
            event->getCurrentTarget()->runAction(MoveBy::create(0.01, Vec2(-40,
0)));
            playerSprite->setTexture("Left.png");
        }
        break;
    case EventKeyboard::KeyCode::KEY_DOWN_ARROW:
        if (playerSprite->getPositionX() <= 0 + (playerSprite->
getContentSize().width / 2)) {
            playerSprite->setPositionX(playerSprite->getContentSize().width / 2);
        }
        else {
            event->getCurrentTarget()->runAction(JumpBy::create(0.5, Vec2(-100,
0), 100, 1));
            playerSprite->setTexture("Jump_left.png");
        }
        break;
    case EventKeyboard::KeyCode::KEY_RIGHT_ARROW:
        if (playerSprite->getPositionX() >= visibleSize.width - (playerSprite->
getContentSize().width / 2)) {
            playerSprite->setPositionX(visibleSize.width - (playerSprite->
getContentSize().width / 2));
        }
        else {
            event->getCurrentTarget()->runAction(MoveBy::create(0.01, Vec2(40,
0)));
            playerSprite->setTexture("Right.png");
        }
            break;
    }
};
_eventDispatcher->addEventListenerWithSceneGraphPriority(keyboardListener,
playerSprite);
```

Collision Detection :

C’est le processus de déterminer si deux objets ou plus sont entrés en collision. Les détections de collision qui doivent être effectuées sont entre le joueur et les obstacles, et entre le joueur et l’objectif. La première collision passera le jeu à la scène Game Over tandis que la deuxième passera le jeu à la scène Level Up, et ce en utilisant le code suivant (en changeant les noms de classes selon chaque niveau) :

```
EventListenerPhysicsContact* listener = EventListenerPhysicsContact::create();
listener->onContactBegin = [playerBody](PhysicsContact& contact) {
    int categoryA = contact.getShapeA()->getBody()->getCategoryBitmask();
    int categoryB = contact.getShapeB()->getBody()->getCategoryBitmask();

    if (contact.getShapeA()->getBody()==playerBody || contact.getShapeB()->
getBody()==playerBody)
    {
        // Vérifier quelle catégorie le joueur a entrée en collision avec
        if (categoryA == BITMASK_DOOR || categoryB == BITMASK_DOOR)
        {
            // Le joueur a attaint l’objectif
```

```
            CocosDenshion::SimpleAudioEngine::getInstance()->
playEffect("audios/LevelUp.mp3");
            auto scene = LevelUp1::createScene();
            Director::getInstance()->replaceScene(scene);
        }
        else if (categoryA == BITMASK_OBSTACLE || categoryB == BITMASK_OBSTACLE)
        {
            // Le joueur est entrée n collision avec une obstacle
            CocosDenshion::SimpleAudioEngine::getInstance()->
playEffect("audios/GameOver.mp3");
            auto scene = GameOver1::createScene();
            Director::getInstance()->replaceScene(scene);
        }
    }
    return true;
};
this->getEventDispatcher()->addEventListenerWithSceneGraphPriority(listener,
this);
```

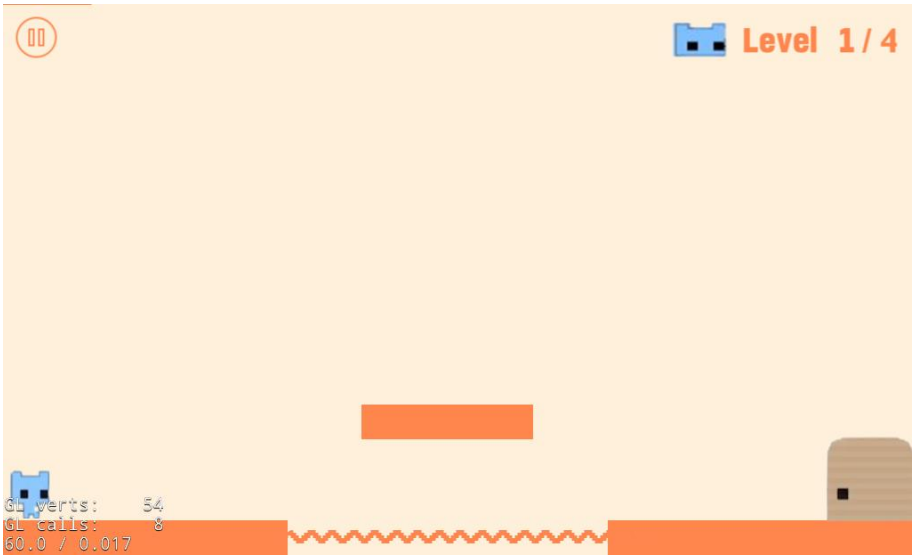

Figure 3: Level 1 Scene

On définit dans ce niveau la notion de gravité en ajoutant le code suivant dans la fonction createScene() :

```
scene->getPhysicsWorld()->setGravity(Vec2(0, -100.f));
```

On définit ensuite les physiques pour le joueur et pour chacune des sprites de la catégorie Platformes :

```
auto playerSprite = Sprite::create("Right.png");
playerSprite->setPosition(Vec2(30, 70));
PhysicsBody* playerBody = PhysicsBody::createBox(playerSprite-> getContentSize(),
PhysicsMaterial(1000000000.0f, 1.0f, 1000000000.0f));
playerBody->setGravityEnable(false);
playerBody->setCategoryBitmask(BITMASK_PLAYER);
playerBody->setCollisionBitmask(BITMASK_PLAYER);
playerBody->setContactTestBitmask(BITMASK_OBSTACLE | BITMASK_DOOR);
playerBody->setDynamic(true);
playerBody->setRotationEnable(false);
playerSprite->setPhysicsBody(playerBody);
this->addChild(playerSprite, 0, 1);
```

```
auto scaleSprite = Sprite::create("Scale.png");
```

```
scaleSprite->setPosition(Vec2(500, 130));
PhysicsBody* scaleBody = PhysicsBody::createBox(scaleSprite->getContentSize(),
PhysicsMaterial(1000000000.0f, 1.0f, 1000000000.0f));
scaleBody->setGravityEnable(false);
scaleBody->setDynamic(false);
scaleSprite->setPhysicsBody(scaleBody);
this->addChild(scaleSprite);
```

Et on ajoute enfin le code permettant de placer le joueur sur les plateformes et ne pas tomber :

```
auto scaleRect = scaleSprite->getBoundingBox();
auto ground1Rect = ground1Sprite->getBoundingBox();
auto playerRect = playerSprite->getBoundingBox();

if (!playerRect.intersectsRect(ground1Rect))
{
    playerBody->setGravityEnable(true);
    playerSprite->setPosition(playerSprite->getPosition().x,
ground1Rect.getMaxY());
}
if (!playerRect.intersectsRect(scaleRect))
{
    playerSprite->setPosition(playerSprite->getPosition().x,
scaleRect.getMaxY());
}
```

Figure 4: Level 2 Scene

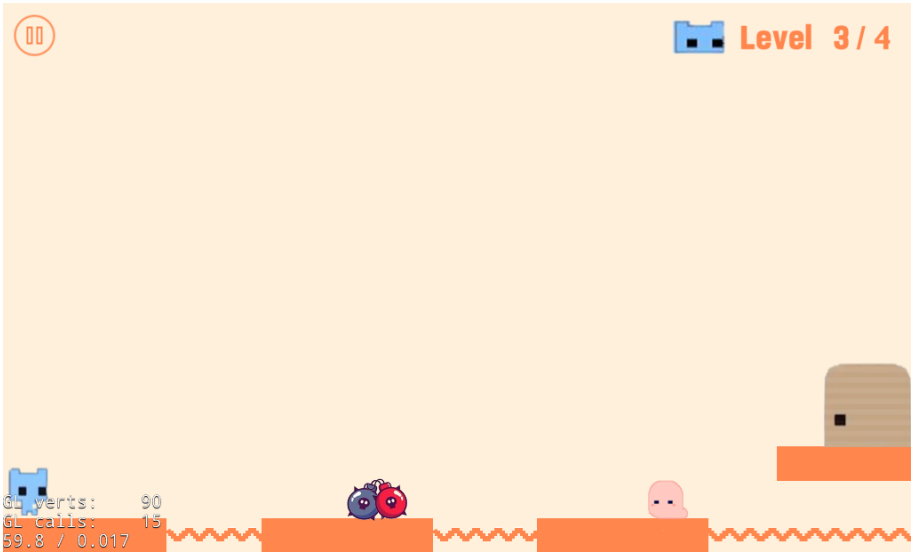

Figure 5: Level 3 Scene

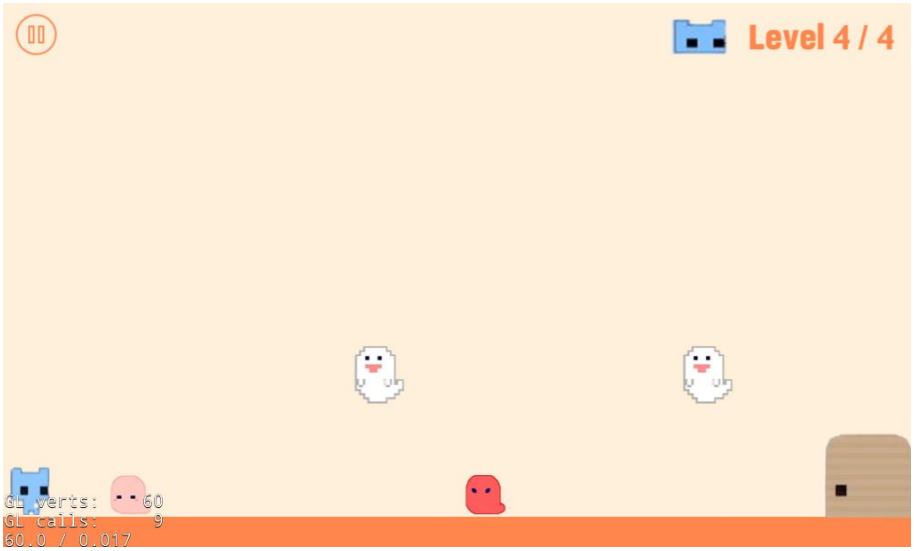

Figure 6: Level 4 Scene

Pour ce dernier niveau, on a développé une nouvelle fonctionnalité permettant la mise à jour de la position de sprite du sol en la déplaçant 250 pixels, et des différents ennemis en la déplaçant 200 pixels, et revenir ensuite à la position d'origine lorsqu'ils atteignent le bord de l'écran. Et ce en utilisant la fonction update() suivante :

```
void Level4::update(float dt) {
    Size visibleSize = Director::getInstance()->getVisibleSize();
    Point origin = Director::getInstance()->getVisibleOrigin();

    auto position = roadSprite->getPosition();
    position.x -= 250 * dt;
    if (position.x < 0 - (roadSprite->getBoundingBox().size.width / 2))
        position.x = this->getBoundingBox().getMaxX() + roadSprite->
getBoundingBox().size.width / 2;
    roadSprite->setPosition(position);

    auto position1 = enemy1->getPosition();
    position1.x -= 200 * dt;
    if (position1.x < 0 - (enemy1->getBoundingBox().size.width / 2) -
(doorSprite->getBoundingBox().size.width / 2))
```

```
        position1.x = this->getBoundingBox().getMaxX() + enemy1-
>getBoundingBox().size.width / 2;
    enemy1->setPosition(position1);

    auto position2 = enemy2->getPosition();
    position2.x -= 200 * dt;
    if (position2.x < 0 - (enemy2->getBoundingBox().size.width / 2))
        position2.x = this->getBoundingBox().getMaxX() + enemy2-
>getBoundingBox().size.width / 2;
    enemy2->setPosition(position2);

    auto position3 = enemy3->getPosition();
    position3.x -= 200 * dt;
    if (position3.x < 0 - (enemy3->getBoundingBox().size.width / 2))
        position3.x = this->getBoundingBox().getMaxX() + enemy3-
>getBoundingBox().size.width / 2;
    enemy3->setPosition(position3);

    auto position4 = enemy4->getPosition();
    position4.x -= 200 * dt;
    if (position4.x < 0 - (enemy4->getBoundingBox().size.width / 2))
        position4.x = this->getBoundingBox().getMaxX() + enemy4-
>getBoundingBox().size.width / 2;
    enemy4->setPosition(position4);
}
```

- **Level Up Scenes**

Ce sont les scènes affichées lorsque le joueur arrive au port et passe alors le niveau avec succès. Chacune des scènes contient une différente sprite de l'arrière-plan, et un menu composé de deux élément : "backItem" permettant de retourner vers la scène de Select Lecel afin de choisir de nouveau un niveau, et "nextItem" qui amène directement vers le niveau suivant. Pour level 4, lorsqu'on est terminé tous les niveaux, ce bouton amène vers la scène de menu principale ou le joueur aura le choix de continuer le jeu en "Endless Mode" ou choisir le "World Mode" et commencer à nouveau. Le suivant est un exemple de code utilisé pour la scène Level Up du premier niveau :

```
auto backItem = MenuItemImage::create("Back.png", "Back(Click).png",
CC_CALLBACK_1(LevelUp1::GoToSelectLevelScene, this));
auto nextItem = MenuItemImage::create("Next.png", "Next(Click).png",
CC_CALLBACK_1(LevelUp1::GoToLevel2Scene, this));
auto menu = Menu::create(backItem, nextItem, NULL);
menu->alignItemsHorizontallyWithPadding(visibleSize.height / 2);
backItem->setPosition(Vec2(-450, 12));
nextItem->setPosition(Vec2(420, 12));
this->addChild(menu);
```

```
void LevelUp1::GoToSelectLevelScene(cocos2d::Ref* pSender)
{
    auto scene = SelectLevel::createScene();
    Director::getInstance()->replaceScene(scene);
}
```

```
void LevelUp1::GoToLevel2Scene(cocos2d::Ref* pSender)
{
    auto scene = Level2::createScene();
    Director::getInstance()->replaceScene(scene);
}
```

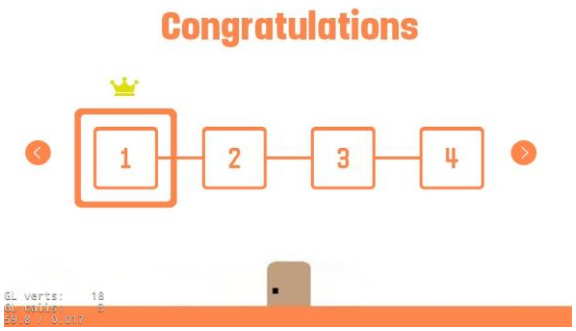

Figure 7: Level Up 1 Scene

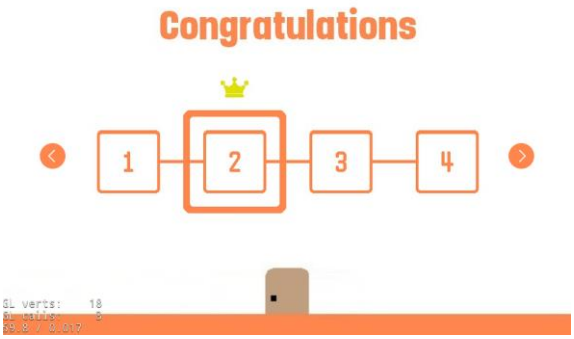

Figure 8: Level Up 2 Scene

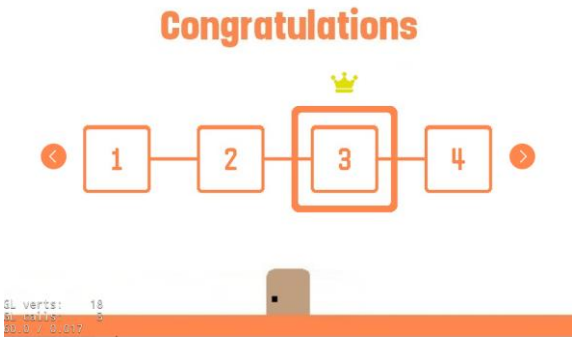

Figure 9: Level Up 3 Scene

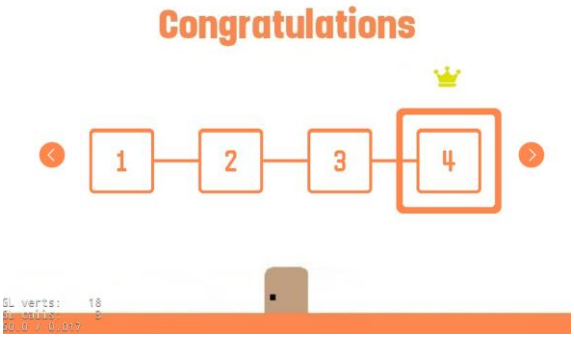

Figure 10: Level Up 4 Scene

3. Endless Mode

Le concept de cette mode est de faire bouger le joueur à droite et à gauche, afin d'éviter le contact avec des bombes qui apparaissent dans des positions aléatoires.

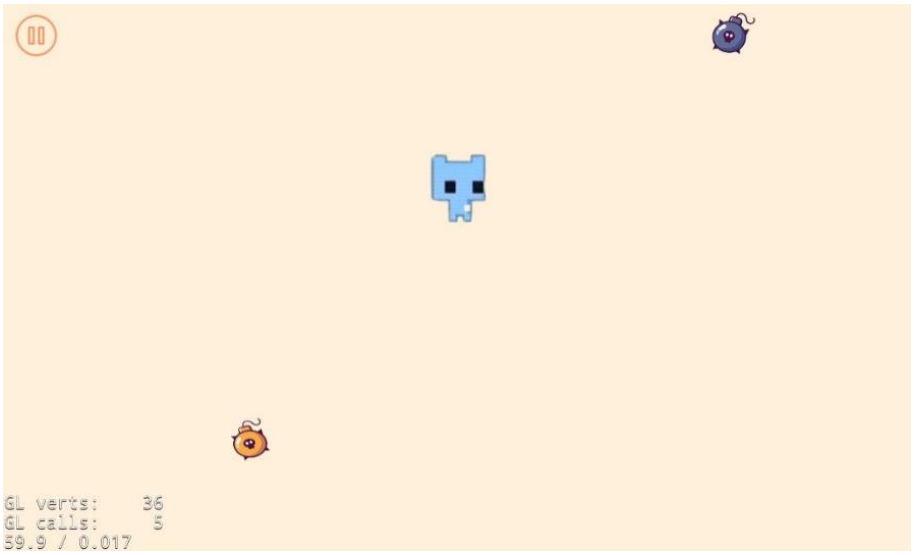

Figure 11: Game Scene

Sprites :

Pour le sprite de l'arrière-plan, on déclare un tableau composé de deux sprites qui seront utilisés pour donner l'illusion d'un arrière-plan qui défile. Une sera initialement affichée à l'écran, et l'autre sous l'écran sera affichée lorsque l'arrière-plan défile vers le haut.

```
cocos2d::Sprite* backgroundSpriteArray[2];
```

```
for (int i = 0; i < 2; i++) {
        backgroundSpriteArray[i] = Sprite::create("Game_Screen_Background.png");
        backgroundSpriteArray[i]->setPosition(
            Point((visibleSize.width / 2) + origin.x,
                (visibleSize.height / 2) + origin.y));
        this->addChild(backgroundSpriteArray[i], -2);
    }
```

On crée également le sprite du joueur en définissant ses physics à partir du code suivant :

```
auto body = PhysicsBody::createCircle(playerSprite->getContentSize().width / 2);
body->setContactTestBitmask(true);
body->setDynamic(true);
playerSprite->setPosition(Point((visibleSize.width / 2),(visibleSize.height /
1.5)));
playerSprite->setPhysicsBody(body);
this->addChild(playerSprite, -1);
```

Pour les sprites des bombs, on utilise le code suivant pour les faire apparaitre au hasard sur l'axe des x juste en dessous de l'écran et les détruire une fois qu'elles sortent de l'écran visible :

```
void GameScreen::spawnBomb(float dt)
{
    Size visibleSize = Director::getInstance()->getVisibleSize();
    Point origin = Director::getInstance()->getVisibleOrigin();

    int bombIndex = (random() % 3) + 1;
    std::string bombString = StringUtils::format("Bomb_%i.png", bombIndex);

    Sprite* tempBomb = Sprite::create(bombString.c_str());

    int xRandomPosition = (random() % (int)(visibleSize.width - (tempBomb->
getContentSize().width / 2))) + (tempBomb->getContentSize().width / 2);
    tempBomb->setPosition(Point(xRandomPosition + origin.x, -tempBomb->
getContentSize().height + origin.y));

    bombs.push_back(tempBomb);

    auto body = PhysicsBody::createCircle(bombs[bombs.size() - 1]->
getContentSize().width / 2);
    body->setContactTestBitmask(true);
    body->setDynamic(true);
    bombs[bombs.size() - 1]->setPhysicsBody(body);
    this->addChild(bombs[bombs.size() - 1], -1);

}
```

```
void GameScreen::update(float dt)
{
    Size visibleSize = Director::getInstance()->getVisibleSize();
    Point origin = Director::getInstance()->getVisibleOrigin();

    for (int i = 0; i < bombs.size(); i++) {
        bombs[i]->setPosition(
            Point(bombs[i]->getPositionX(),
                bombs[i]->getPositionY() + (0.75 * visibleSize.height * dt)));
```

```
        if (bombs[i]->getPosition().y > visibleSize.height + (bombs[i]->
getContentSize().height / 2)) {
            this->removeChild(bombs[i]);
            bombs.erase(bombs.begin() + i);
        }
    }
}
```

Touch controls :

Pour cette mode de jeu, en utilise une autre méthode pour bouger le joueur est ce en ajoutant tout d'abord les événements de touch suivants:

```
auto listener = EventListenerTouchOneByOne::create();
listener->setSwallowTouches(true);
listener->onTouchBegan = CC_CALLBACK_2(GameScreen::onTouchBegan, this);
listener->onTouchMoved = CC_CALLBACK_2(GameScreen::onTouchMoved, this);
listener->onTouchEnded = CC_CALLBACK_2(GameScreen::onTouchEnded, this);
listener->onTouchCancelled = CC_CALLBACK_2(GameScreen::onTouchCancelled, this);
this->getEventDispatcher()->addEventListenerWithSceneGraphPriority(listener,
this);
```

Pour gérer ces événements, on déclare des variables `isTouching` et `touchPosition` qui ont comeme valeur initiale respectivement `false` et `0` et selon chaque événement on change ces valeurs comme suit :

```
bool GameScreen::onTouchBegan(cocos2d::Touch* touch, cocos2d::Event* event)
{
    isTouching = true;
    touchPosition = touch->getLocation().x;
    return true;
}
```

```
void GameScreen::onTouchMoved(cocos2d::Touch* touch, cocos2d::Event* event)
{
}
```

```
void GameScreen::onTouchEnded(cocos2d::Touch* touch, cocos2d::Event* event)
{
    isTouching = false;
}
```

```
void GameScreen::onTouchCancelled(cocos2d::Touch* touch, cocos2d::Event* event)
{
    onTouchEnded(touch, event);

}
```

On ajoute finalement le code suivant dans la fonction update() permettant de déplacer le joueur vers la gauche et la droite :

```
if (isTouching) {
    if (touchPosition < visibleSize.width / 2) {
        //Déplacer le joueur vers la gauche
        playerSprite->setPositionX(playerSprite->getPositionX() - (0.50 *
visibleSize.width * dt));
        playerSprite->setTexture("playerLeft.png");
        //Vérifier pour empêcher le joueur de sortir du côté gauche de l'écran
```

```
        if (playerSprite->getPositionX() <= 0 + (playerSprite->
getContentSize().width / 2)) {
            playerSprite->setPositionX(playerSprite->getContentSize().width / 2);
        }
    }
    else {
        // Déplacer le joueur vers la droite
        playerSprite->setPositionX(playerSprite->getPositionX() + (0.50 *
visibleSize.width * dt));
        playerSprite->setTexture("playerRight.png");
        //Vérifier pour empêcher le joueur de sortir du côté droite de l'écran
        if (playerSprite->getPositionX() >= visibleSize.width - (playerSprite->
getContentSize().width / 2)) {
            playerSprite->setPositionX(visibleSize.width - (playerSprite->
getContentSize().width / 2));
        }
    }
 }
```

Collision detection :
Une fois une collision a eu lieu entre le joueur et les bombs, le joueur meurt et la fonction suivante sera donc appelée :

```
bool GameScreen::onContactBegin(cocos2d::PhysicsContact& contact) {
    CocosDenshion::SimpleAudioEngine::getInstance()->
playEffect("audios/GameOver.mp3");

    playerSprite->setTexture("playerDead.png");
    GoToGameOverScene(this);
    return true;
}
```

4. Pause Scenes

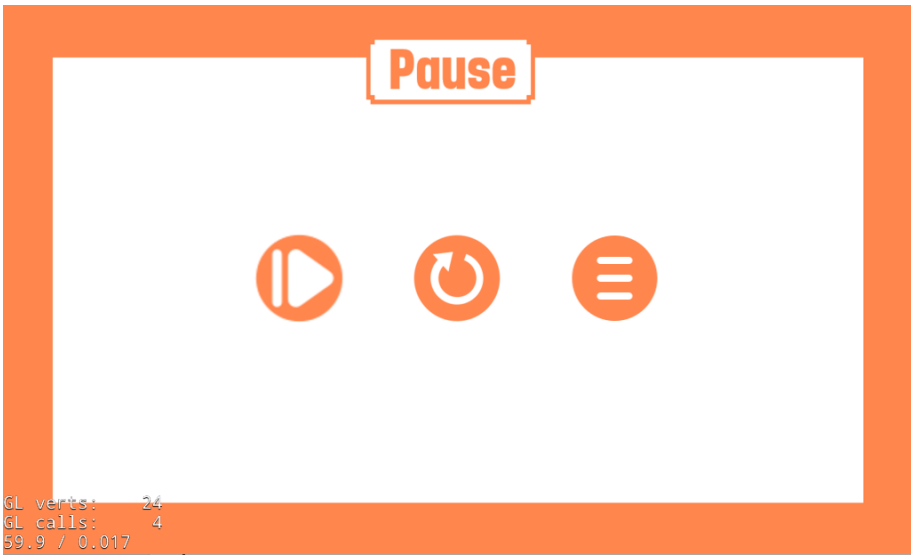

Figure 12: Pause Scene

On a développé 5 scènes de pause, une pour chaque niveau et une autre pour "Endless Mode". Chacune de ces scènes contient :

Menu :
Un menu constitué de trois boutons permettant la navigation entre les scènes. Le premier permet de continuer le jeu, le deuxième permet de le recommencer, et le troisième amène vers le menu principal.

```
auto resumeItem = MenuItemImage::create("Resume.png", "Resume.png",
CC_CALLBACK_1(Pause1Menu::Resume, this));
    auto retryItem = MenuItemImage::create("Retry.png", "Retry.png",
CC_CALLBACK_1(Pause1Menu::Retry, this));
    auto mainMenuItem = MenuItemImage::create("Menu.png", "Menu.png",
CC_CALLBACK_1(Pause1Menu::GoToMainMenuScene, this));

    auto menu = Menu::create(resumeItem, retryItem, mainMenuItem, NULL);
    menu->alignItemsHorizontallyWithPadding(visibleSize.height / 8);
    this->addChild(menu);
```

Sprite :
Une sprite de l'arrière-plan créée à partir d'une image PNG.

```
auto backgroundSprite = Sprite::create("Pause_Background.png");
backgroundSprite->setPosition(Point((visibleSize.width / 2) + origin.x,
(visibleSize.height / 2) + origin.y));
this->addChild(backgroundSprite, -1);
```

Méthodes de manipulation des scènes :
On utilise les deux méthodes :
popScene() qui supprime la scène actuelle et revient à la scène du jeu.
replaceScene() qui remplace la scène actuelle par une nouvelle scène.

```
void Pause1Menu::Resume(cocos2d::Ref* pSender)
{
    Director::getInstance()->popScene();
}
```

```
void Pause1Menu::GoToMainMenuScene(cocos2d::Ref* pSender)
{
    auto scene = MainMenu::createScene();
    Director::getInstance()->popScene();
    Director::getInstance()->replaceScene(scene);
}
```

```
void Pause1Menu::Retry(cocos2d::Ref* pSender)
{
    auto scene = Level1::createScene();
    Director::getInstance()->popScene();
    Director::getInstance()->replaceScene(scene);
}
```

5. Game Over Scenes

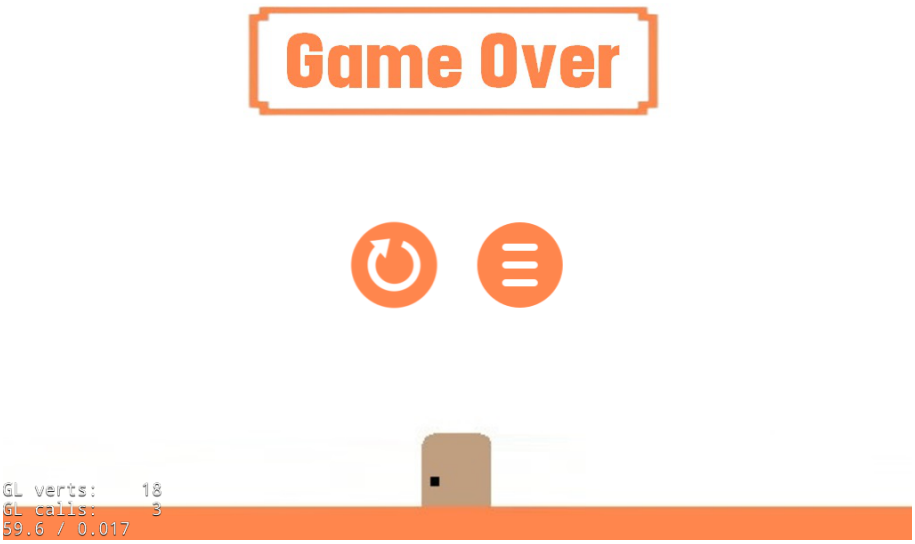

Figure 13: Game Over Scene

De même que Pause Scene, on développe 5 scènes qui s'afficheront lorsque le joueur meurt et perd le jeu dans l'une des 4 niveaux ou dans "Endless Mode". Chacune des 5 scènes est constitué d'un sprite de l'arrière-plan et un menu composé de deux élément, l'un pour recommencer le jeu que le joueur vient de perdre et l'autre pour abandonner et aller vers le menu principal, et ceci en utilisant la méthode replaceScene().

```
void GameOver::GoToGameScene(cocos2d::Ref* pSender)
{
    auto scene = GameScreen::createScene();
    Director::getInstance()->replaceScene(scene);
}
```

```
void GameOver::GoToMainMenuScene(cocos2d::Ref* pSender)
{
    auto scene = MainMenu::createScene();
    Director::getInstance()->replaceScene(scene);
}
```

CONCLUSION

Le développement d'un tel jeu vidéo en utilisant un tel moteur de jeu n'était pas vraiment assez simple. On a rencontré plusieurs difficultés, mais à force du travail et de la persévérance, on a pu réussir notre projet. On a voulu développé plusieurs autres options et fonctionnalités, et mettre en place un meilleur jeu, cependant, on est tellement fiers de l'effort qu'on a fait ainsi que le résultat qu'on a obtenu.

Ci-dessous, quelques difficultés rencontrées parmi plusieurs autres :

Difficultés rencontrées

Contrainte du temps :

Il est vrai que la durée accordée au projet était largement suffisante, mais en attendant l'avancement du cours et des leçons prévues pour le développement des jeux vidéo, on ne l'a commencé qu'après tard, et avec un emploi du temps trop chargé et des devoirs, travaux pratiques, projets et contrôles continus des 8 autres modules c'était vraiment difficile de le terminer avant le délai précis.

Manque de ressources :

Le manque immense de ressources disponibles pour ce moteur de jeu a sérieusement limité notre travail et rendu le développement de plusieurs options quasi impossible. On a utilisé toutes les ressources possibles auxquelles vous pouvez penser, pourtant, ce n'était pas ce qu'on cherche exactement et ce qu'on est demandé de faire, et même lorsqu'on trouve une solution répondant à notre besoin, cette solution dans la plupart de temps utilise des méthodes dépassées ou qui ne sont pas définies dans notre version de Cocos2d.

Répartition du travail entre le groupe

Notre travail n'était pas vraiment réparti en tâches ; chacun des membres s'est chargé de la recherche et l'essai du développement de chaque option, et celui qui a réussi à développer une, on la mettre en place et on passe à la suivante, tout en gardant une très bonne communication sur l'état d'avancement du projet et un bon esprit d'équipe afin de réussir le travail.

ANNEXES

Code source du jeu

https://github.com/firdaous-boulben/Boulben-Chibani-JEUX.git

Vidéo démonstrative du jeu

https://photos.app.goo.gl/gjL8xM8TmTakEk718