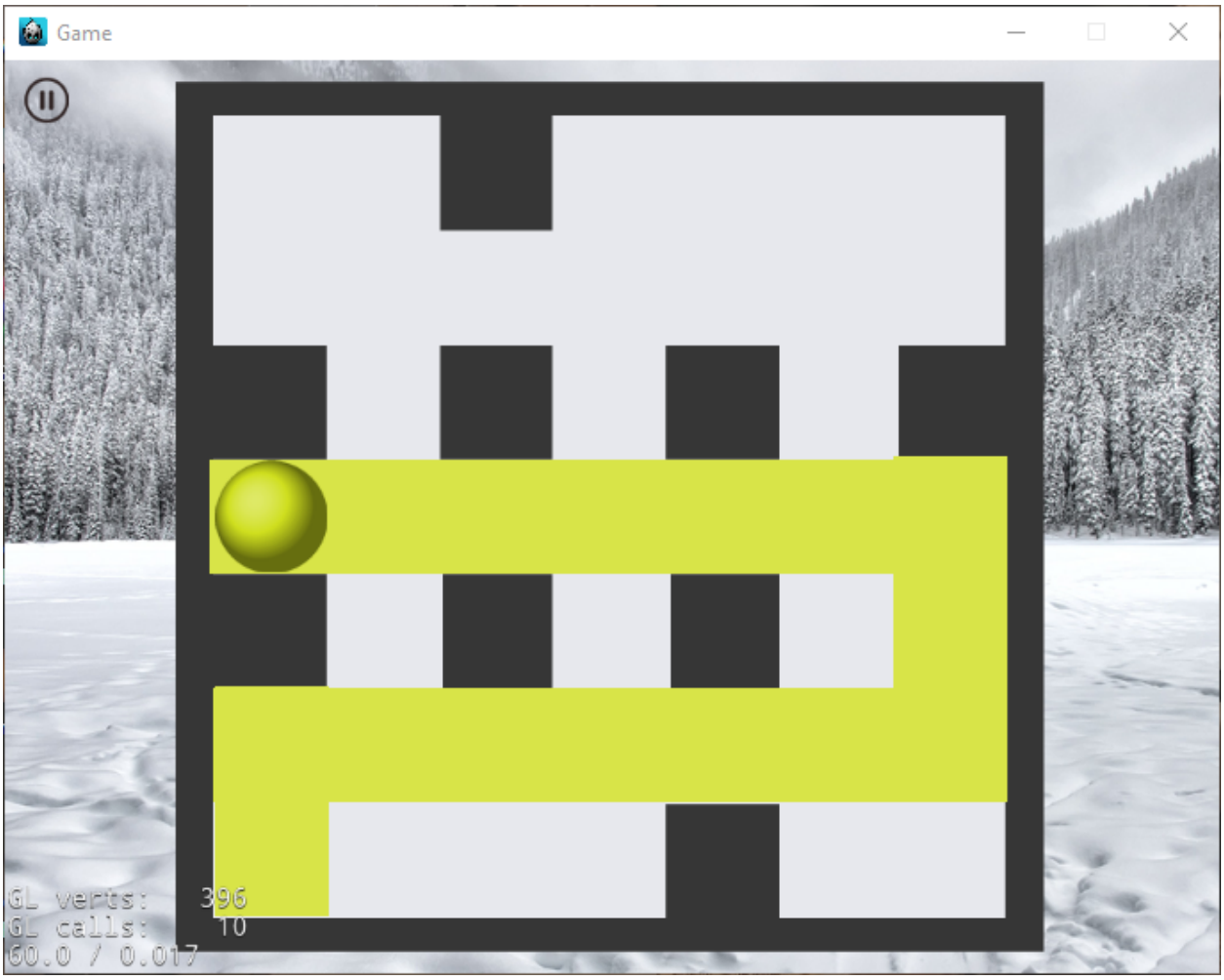
Université Abdelmalek Essaâdi

Faculté Des Sciences Et Techniques – Tanger

LST GENIE INFORMATIQUE GRP2 2021/2022

LE JEUX ROLLER SPLAT EN C++



REALISE PAR : BEN ISSA MONCEF
ENCADRE PAR : Mme BEN ABDEL OUAHAB IKRAM & Mr EL AACHAK LOTFI
LIEN DE GITHUB :

REMERCIEMENT:

Nous voudrions exprimer nos vifs remerciements et notre gratitude ainsi
notre considération distinguée à **Mme Ben Abdel Ouahab Ikram** et **Mr El Aachak Lotfi**,
de nous donner tout d'abord l'opportunité de travailler
sur ce projet, durant lequel j'ai pu acquérir et développer plusieurs connaissances
et compétences, qui vont nous servir sans doute tout au long de nôtre carrière
en tant que futurs informaticiens, et aussi pour ses directives et conseils, ainsi
pour leurs aide et son soutien tout au long de la période de la réalisation
de ce projet.

En effet, au cours de ce projet nous avons pu développer certaines compétences
professionnelles tel que la prise d'initiative, le respect des délais et le travail
en équipe qui seront des aspects essentiels de notre futur métier.
De plus, ce projet nous a permis d'appliquer nos connaissances qu'on a acquis
à travers le module de la programmation orienté objet en C++ et Python.

INTRODUCTION:

1- L'IDEE DU JEUX:

L'idée du jeu c'est un ballon qui vous pouvez le balayer vers le haut, le bas à travers le labyrinthe pendant que vous éclaboussez la couleur partout dans le labyrinthe. Vous pouvez gagner dans les niveaux de chaque puzzle en couvrant chaque couloir et coin avec une peinture.

2- COCOS2D-x V4 :

Cocos2d-x est l'une des bibliothèques logicielles de référence. En plus de permettre de créer des jeux en 2D pour les appareils mobiles Android, iOS et Windows Phone, elle compile sur Windows, Mac et Linux. La bibliothèque peut être utilisée pour le développement en C++, Javascript et Lua.

3- ADOBE PHOTOSHOP:

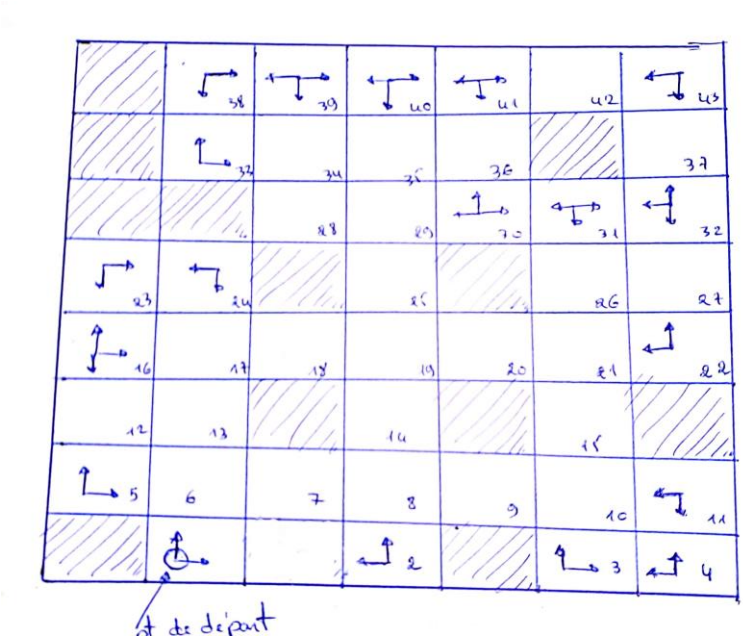
Photoshop est un logiciel de retouche, de traitement et de dessin assisté par ordinateur, édité par Adobe, il est principalement utilisé pour le traitement des photographies numériques, Il travaille essentiellement sur images matricielles car les images sont constituées d'une grille de points appelés pixels. L'intérêt de ces images est de reproduire des gradations subtiles de couleurs.

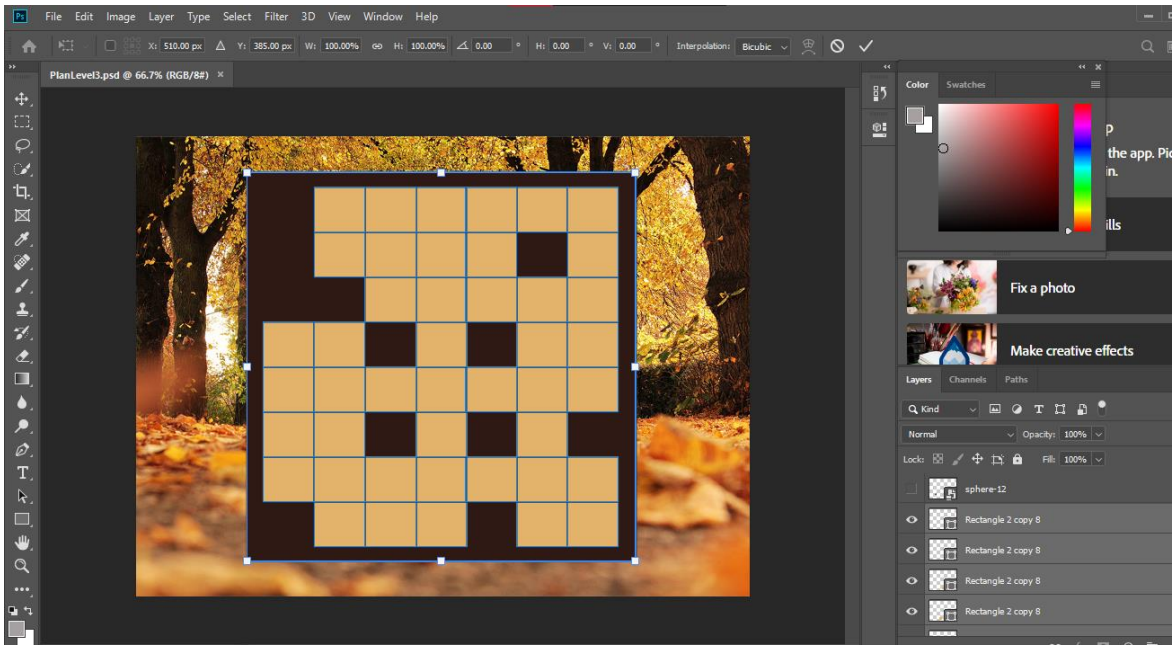
4- MICROSOFT VISUAL STUDIO:

Visual Studio est un ensemble complet d'outils de développement permettant de générer des applications web ASP.NET, des services web XML, des applications bureautiques et des applications mobiles.

CONCEPTION DES PLANS:

L’idée generale c’est que j’ai éssayé de traiter les plans comme une matrice de N lignes et M colonnes pour la facilité de conception graphique et la facilité des déplacements après.Tous les plans ont une dimension de 1024px/768px, et chaque plan a une arrire plan differente. pour la conception graphique j’ai utilisé le logiciel Adobe photoshop et les sites web Unsplash et Pexels pour les arrières plans.





CREATION DE SPRITE:

Prenant l'exemple du ballon de premier niveau:

1- Déclaration de Sprite dans la classe principale (Dans le fichier Niveau1.h) :

```
cocos2d::Sprite* Sprite1;
```

2- Après la déclaration, dans le fichier Niveau1.cpp on fait la création en utilisant la fonction **Sprite::create(),** On place notre élément dans une position initiale en utilisant la fonction **setPosition()** et finalement on ajoute l'élément avec un z-index (Dans ce cas le z-index = 1).

```
//CREATION DU BALLON:

Sprite1 = Sprite::create("Niveau1/SPRITE1.png");
Sprite1->setPosition(Point(121, 37));
this->addChild(Sprite1, 1);
```

J'ai fait la même chose pour les autres objets de classe Sprite sauf par exemple dans le cas d'arrière-plan je fait la définition de position au centre de l'écran en utilisant la taille visible, qui est la zone de l'écran qui est disponible sur un appareil particulier, car il existe différentes tailles d'écran.

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

auto PlanNiveau1 = MenuItemImage::create("Niveau1/PlanLevel1.png", "MainMenuScreen/Game_Title.png");
PlanNiveau1->setPosition(Point((visibleSize.width / 2) +
origin.x, (visibleSize.height / 2) + origin.y));
this->addChild(PlanNiveau1, -1);
```

CREATION DU MENU:

Prenant l'exemple du premier menu (MainMenuScene):

```
//LA CREATION DES ELEMENTS DE MENU

auto playItem = MenuItemImage::create("MainMenu/CloseNormal.png", "MainMenu/CloseNormal.png",
        CC_CALLBACK_1(MainMenu::GoToGameScene, this));

auto menu = Menu::create(playItem, NULL);
menu->setPosition(Point((visibleSize.width / 2) +
origin.x, (visibleSize.height / 2) + origin.y));
this->addChild(menu);
```

1- On commence par la création des éléments de menu dans ce cas (playerItem) quand on clique sur ce boutton on passe au premier niveau autrement dit on fait l'appel de fonction GoToGameScene().
2- Après on créé le menu en indiquant leur position dans la scène. (Dans ce cas le menu est centré).
3- On ajoute le menu en utilisant la fonction addChild().

La fonction GoToGameScene() :

1- La déclaration :

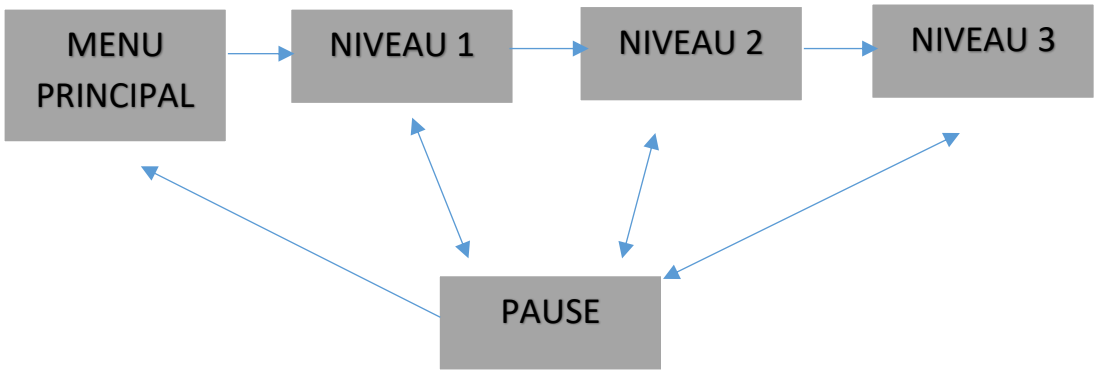
```
void GoToGameScene(Ref* pSender);
```

2- Le code :

```
void MainMenu::GoToGameScene(Ref* pSender)
{
    auto scene = Niveau1::createScene();

    Director::getInstance()->replaceScene(scene);
}
```

Dans ce cas quand on fait l'appel de la fonction elle créée une scène de Niveau1 et change la scène actuelle (MainMenuScene) par la nouvelle scène (Le premier niveau).

SCENES DE JEUX:



1- La creation du scene:

Prenant l'exemple de la deuxième scène:

a- Les déclarations:

```
class Niveau2 : public cocos2d::Scene
{
public:
    static cocos2d::Scene* createScene();
    cocos2d::PhysicsWorld* sceneWorld;
    void SetPhysicsWorld(cocos2d::PhysicsWorld* world)
    {
        sceneWorld = world;
    };
```

b- La création du scene (Dans le fichier Niveau2.cpp) :

```
Scene* Niveau2::createScene()
{
    auto scene = Scene::createWithPhysics();
    scene->getPhysicsWorld()->setDebugDrawMask(PhysicsWorld::DEBUGDRAW_ALL);

    auto layer = Niveau2::create();
    scene->getPhysicsWorld()->setGravity(Vec2(0, 0));
    layer->SetPhysicsWorld(scene->getPhysicsWorld());
    scene->addChild(layer);
    return scene;
}
```

Commentaires :

On fait déclarer et initialiser la méthode du monde physique qui attribue le monde de la physique et définit la gravité à 0 car notre jeu Roller Splat ne nécessite pas de gravité. Ainsi, le moteur physique dans ce cas sera simplement utilisé pour la détection de collision, et après on déclare le monde physique local.

JOUER AVEC LES TOUCHES DE CLAVIER:

```
cocos2d::EventListenerKeyboard* _2ndListner;
```

```
_2ndListner = EventListenerKeyboard::create();
_2ndListner->onKeyPressed = CC_CALLBACK_2(Niveau2::MoveBall, this);
_eventDispatcher->addEventListenerWithFixedPriority(_2ndListner, 1);
```

```
switch (Keycode) {

case EventKeyboard::KeyCode::KEY_LEFT_ARROW:
```

On declare premièrement un événement qui détecte le keycode chaque fois que vous cliquez sur les touches de clavier, pour cela on utilise la fonction prédéfinie **EventListenerKeyboard : : create()**.Après, comme une solution proposée j'ai choisi de travailler avec un switch qui a comme argument le keycode de la touche de clavier avec 4 possibilités (Up, Down, Left and Right) dans chaque cas elle exécute une partie de code.

METHODE DE DEPLACEMENT ET DE COLORATION:

Dans chaque plan il y'a des points dans lesquels la balles peut se déplacer dans N directions (entre 1 et 3 directions).
Ces points sont situés en X ligne et Y colonne, prenant l'exemple du 3éme niveau :

Par exemple dans cette figure si la balle est située dans le point entouré, la balle à 3 possibilités soit se déplacer vers le haut soit vers le bas soit vers la gauche, si l'utilisateur a cliqué sur la flèche « en haut », la balle va se déplacer 2 cases vers le haut et va colorer 3 cases (incluse la position de la balle) et de même pour les autres directions, la coloration et faite par une boucle qui insère N carrées (Cette carrée a la même dimension que la case de la matrice). Si on fait une traduction de cette idée par code on obtient :

Si on clique sur UP :

```
    if (_Location.x == 360 && _Location.y == 212) {
        Sprite3->runAction(UP70);

        for (int x = 0; x < 3; x++) {
            auto REC = Sprite::create("Niveau3/COLOR3.png");
            REC->setPosition(358, 212 + (x * (REC->getContentSize().height)));
            addChild(REC, 0);
        }
    }
```

Avec :

```
auto UP70 = MoveBy::create(0.1, Point(0, 70));
```

Si on clique sur Down :

```
    if (_Location.x == 360 && _Location.y == 212) {
        Sprite3->runAction(DOWN70);

        for (int x = 0; x < 3; x++) {
            auto REC = Sprite::create("Niveau3/COLOR3.png");
            REC->setPosition(358, 212 - (x * (REC->getContentSize().height)));
            addChild(REC, 0);
        }
    }
```

Avec :

```
auto DOWN35 = MoveBy::create(0.1, Point(0,-35));
```

Si on clique sur Left :

```
if (_Location.x == 360 && _Location.y == 212) {
    Sprite3->runAction(L160);

    for (int x = 0; x < 5; x++) {
        auto REC = Sprite::create("Niveau3/COLOR3.png");
        REC->setPosition(358 - (x * (REC->getContentSize().width)), 212);
        addChild(REC, 0);
    }
}
```

Avec :

```
auto L160 = MoveBy::create(0.1, Point(-160, 0));
```

DETECTION DES COLLISIONS:

```
auto contactListener =
    EventListenerPhysicsContact::create();
contactListener->onContactBegin =
    CC_CALLBACK_1(Niveau2::onContactBegin, this);
this->getEventDispatcher()->addEventListenerWithSceneGraphPriority(contactListener, this);
```

Le code précédent ajoute un écouteur pour écouter activement les collisions. en cas de détection de collision, la méthode **onContactBegin** sera appelée.

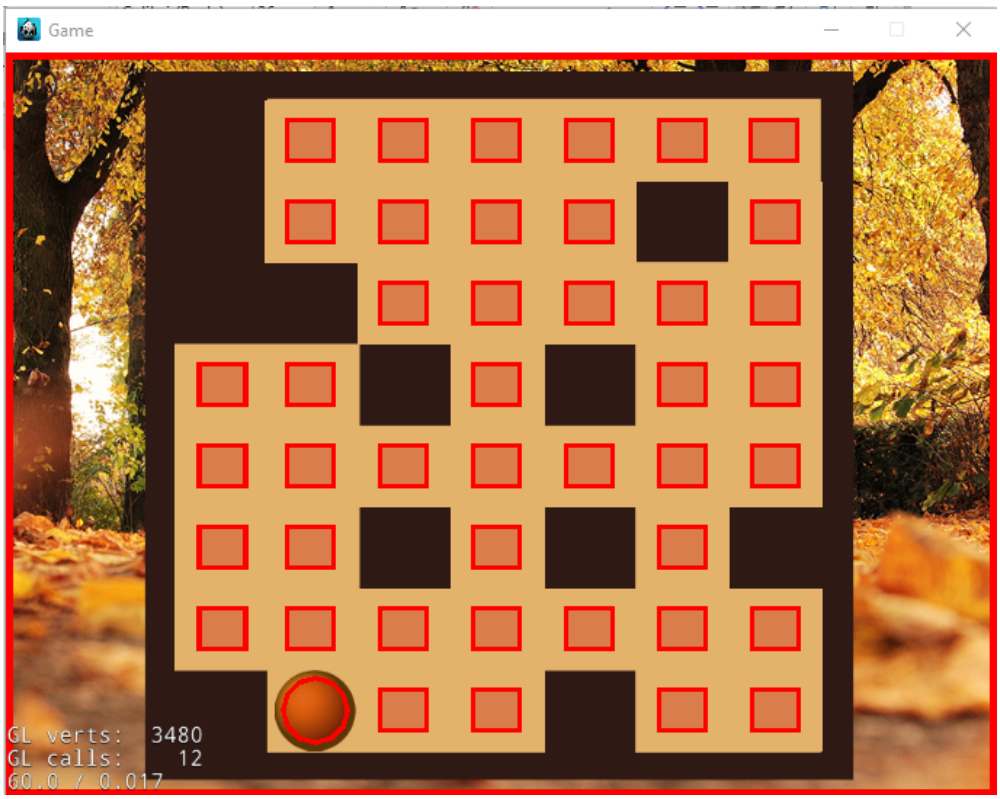
```
bool Niveau3::onContactBegin(PhysicsContact& contact) {
    PhysicsBody* a = contact.getShapeA()->getBody();
    PhysicsBody* b = contact.getShapeB()->getBody();
    static int x = 0;

    if (x != 43) {
        if ((1 == a->getCollisionBitmask()) && 0 == b->getCollisionBitmask() || (0 == a->getCollisionBitmask()) && 1 == b->getCollisionBitmask()) {
            OB1->removeFromParent();
            x++;
            cocos2d::log("%d", x);
            if (x == 43) {
                Bravo();
                x = 0;
            }
            return true;
        }
```

Cette fonction dans notre cas va pour chaque collision savoir quel obstacle a fait la collision et par suite va exécuter un algorithme dans le but de savoir si la balle a parcouru tout le plan.

CREATION DES OBSTACLES:

```
//Creation des obstacles:

//OBS 1
OB1 = Sprite::create("Niveau2/OBS2.png");
OB1->setPosition(120, 80);
auto OB1B = PhysicsBody::createBox(OB1->getContentSize()/2, PhysicsMaterial(0.0f, 0.0f, 0.0f));
OB1B->setDynamic(false);
OB1B->setCollisionBitmask(1);
OB1B->setContactTestBitmask(1);
OB1->setPhysicsBody(OB1B);
this->addChild(OB1);
```

On crée un sprite non dynamique avec une position fixe. apès, on fait la création d'un corps physique qui a la même taille que l'élement avec un identificateur diffirent que le sprite principale (Dans notre cas la balle a un CollisionBitMask = 0 et d'obstacle égale a 1). Les obstacles sont placés avec une manière qu'ils couvrent tout le plan ou bien les couloires principales de plan.



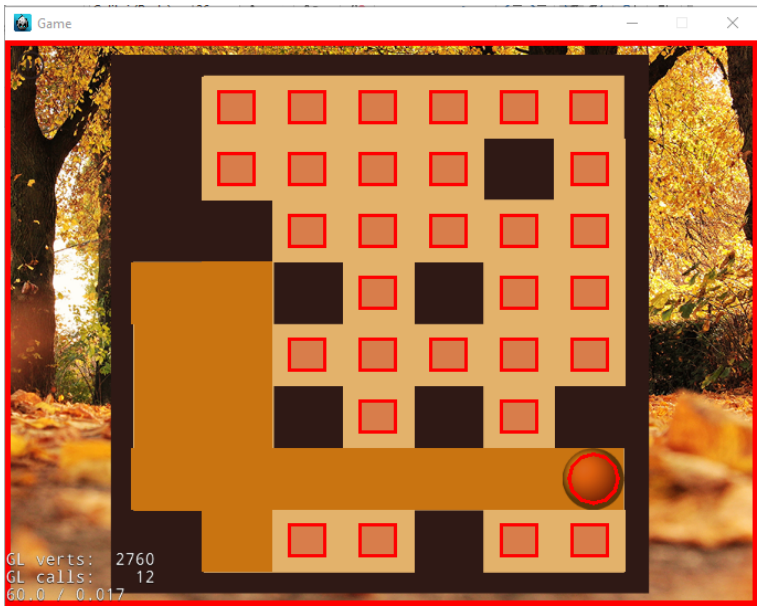
COMMENT SAVOIR QUE LA BALLE A COLOREE TOUT LE PLAN?

Par exemple dans un plan qui contient N obstacles, comme une première étape j'ai déclaré une variable par exemple x égale a 0 et chaque fois qu'on détecte une collision entre la balle et cet obstacle on incrémente la variable et on efface cet obstacle. Si la variable x = N ca veut dire que la balle est passé par tous les obstacles et par suite le niveau est terminé, la dernière étape j'appelle la fonction bravo() ;

```
bool Niveau3::onContactBegin(PhysicsContact& contact) {
    PhysicsBody* a = contact.getShapeA()->getBody();
    PhysicsBody* b = contact.getShapeB()->getBody();
    static int x = 0;

    if (x != 43) {
        if ((1 == a->getCollisionBitmask()) && 0 == b->getCollisionBitmask() || (0 == a->getCollisionBitmask()) && 1 == b->getCollisionBitmask()) {
            OB1->removeFromParent();
            x++;
            cocos2d::log("%d", x);
            if (x == 43) {
                Bravo();
                x = 0;
            }
            return true;
        }
```

Concernant la fonction **onContactBegin()** s'exécute chaque fois qu'on détecte une collision.

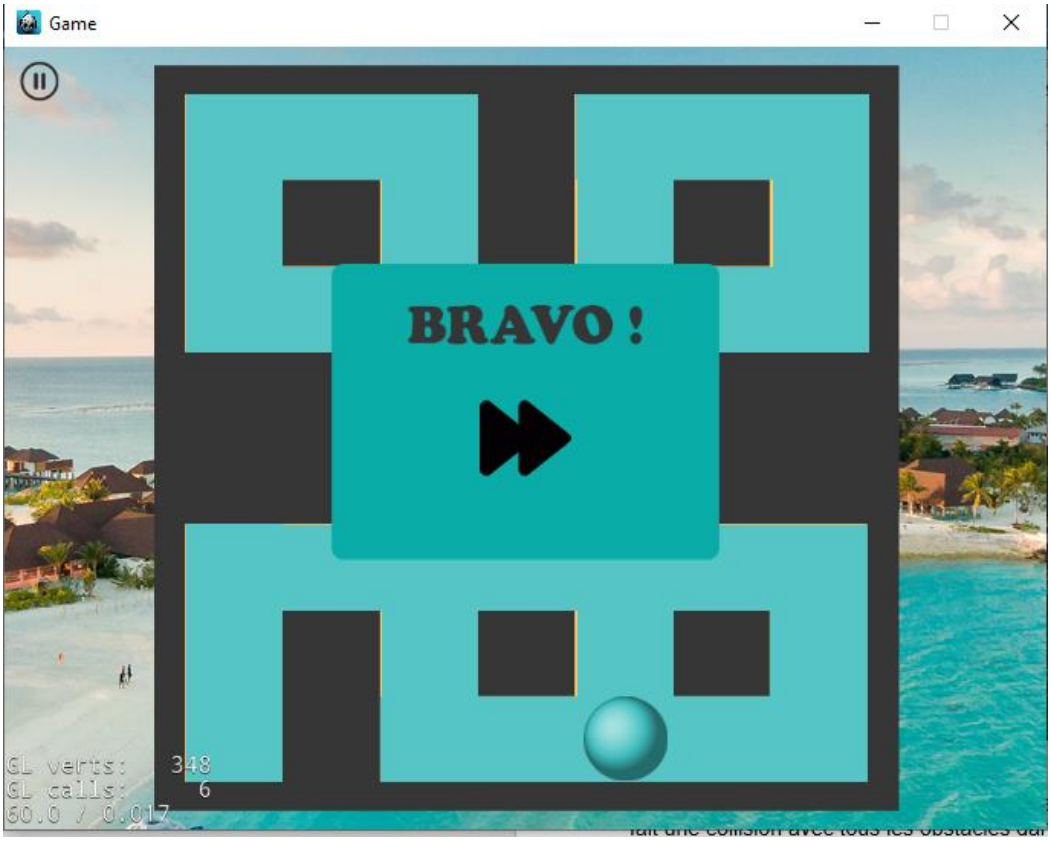
```
bool onContactBegin(cocos2d::PhysicsContact& contact);
```



LA FONCTION BRAVO:

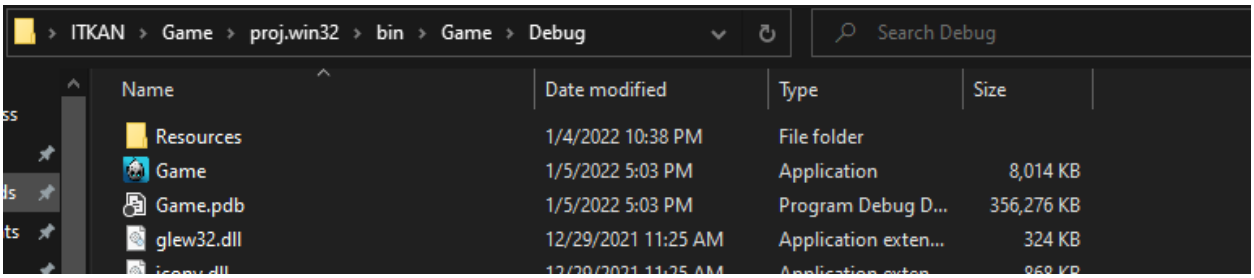
```
void Niveau2::Bravo() {
    auto Saute = JumpBy::create(2.0, Point(0, 0), 15, 5);
    Sprite2->runAction(Saute);

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

    auto Next = MenuItemImage::create("NextLevel.png", "PauseScreen/Menu_Button(Click).png",
        CC_CALLBACK_1(Niveau2::GoToTheNextLevel, this));
    auto MenuNext = Menu::create(Next, NULL);
    this->addChild(MenuNext,6);

    auto Bravo = Sprite::create("Bravo2.png");
    Bravo->setPosition(Point((visibleSize.width / 2) +
        origin.x, (visibleSize.height / 2) + origin.y));
    this->addChild(Bravo,5);
}
```

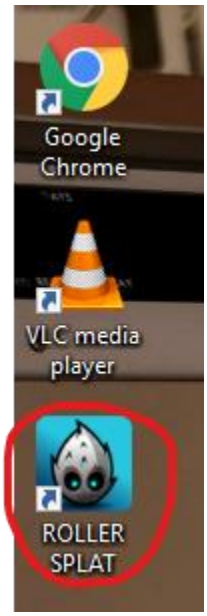
Cette fonction est appelée quand le niveau est terminé ou bien la balle a fait une collision avec tous les obstacles de plan.
La fonction fait une action (une saute), après elle créée un bouton pour passer au niveau suivant.



CREATION DE RACCOURCI:

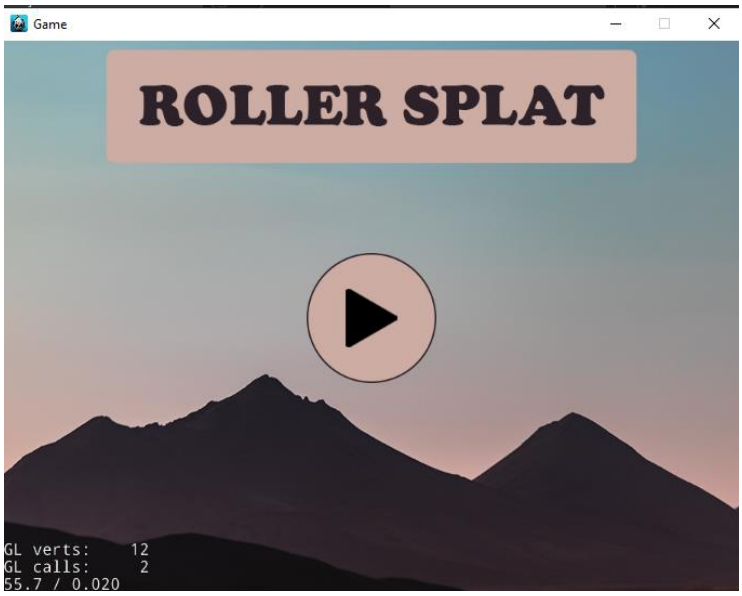
On fait un raccourci d'un fichier éxecutable de chemin Fichier_de_jeux / proj.Win32 / bin / Nom_de_jeux (Game) / Debug.



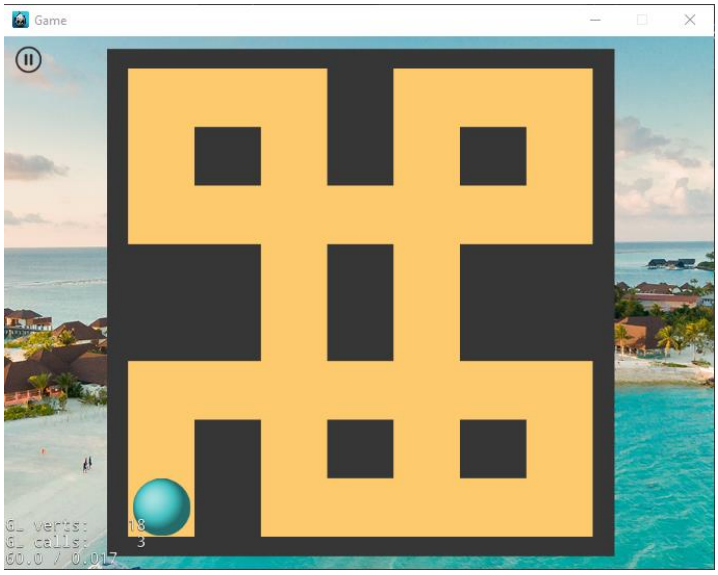


RESULTAS:

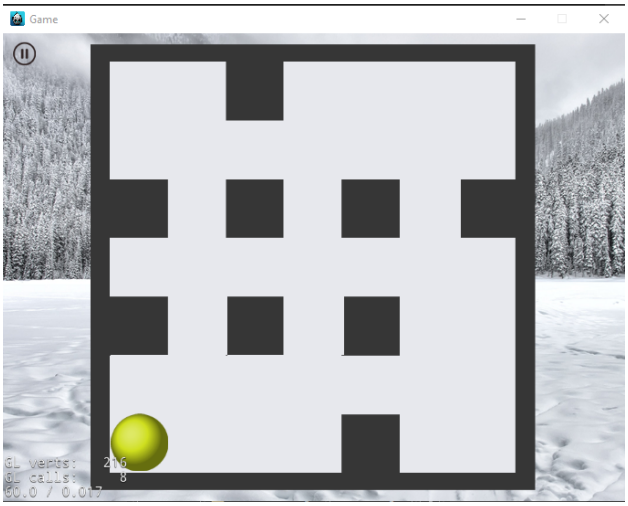
1- Le menu principal:



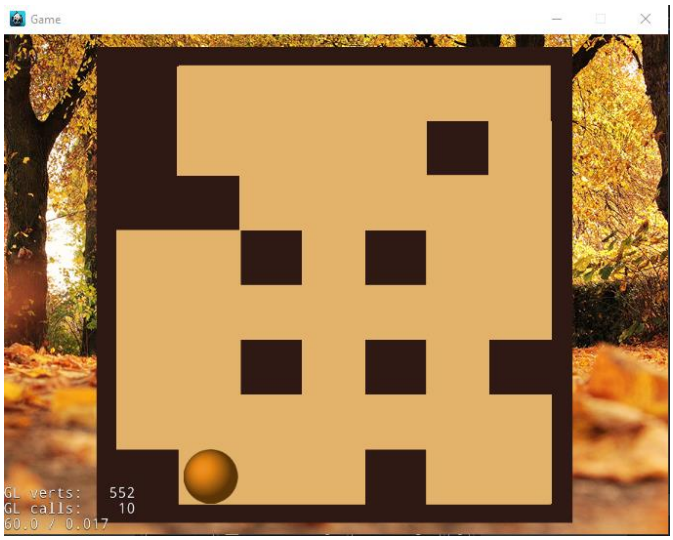
2- Le 1er niveau:



3- Le 2em niveau:



4- Le 3em niveau:



5- La scene de pause: