Dans cette partie, on va parler sur le mouvement de la balle, et la coloration de trajectoire.

Pour manipuler et déplacer la balle dans le map, nous avons utilisé des class et des fonctions déjà définie, Cocos2d-x prend en charge les événements clavier. La class EventListenerKeyboard Permet de fais ça.

Voilà le code source :

```
  auto eventListener = EventListenerKeyboard::create();
    eventListener->onKeyPressed = [=](EventKeyboard::KeyCode KeyCode, Event*
event) {
        int offsetX = 0, offsetY = 0;
        float x, y;

        switch (KeyCode) {
        case EventKeyboard::KeyCode::KEY_LEFT_ARROW:
        case EventKeyboard::KeyCode::KEY_A:
            x = event->getCurrentTarget()->getPositionX();
            y = event->getCurrentTarget()->getPositionY();
            CCLOG("good_position");
            if  ((x == 350 && y == 170)||(x == 340 && y == 450))
            {
                CCLOG("good_position");
                offsetX = -215;
            if ((x == 340 && y == 450))
                {
                 emitter->setVisible(true);
                    auto scene = level2::createScene();
                    Director::getInstance()->replaceScene(TransitionFade::cre-
ate(4, scene));
                }
            }
            else if ((x == 340 && y == 310))
            {
                CCLOG("good_position");
                offsetX = -205;
            }
            else CCLOG("bad_position");
            break;
        case EventKeyboard::KeyCode::KEY_RIGHT_ARROW:
        case EventKeyboard::KeyCode::KEY_D:
            x = event->getCurrentTarget()->getPositionX();
            y = event->getCurrentTarget()->getPositionY();
            if ((x == 135 && y == 310)  )
            {
                CCLOG("good_position");
                offsetX = 205;
            }
            else if ((x == 135 && y == 170))
            {
                CCLOG("good_position");
                offsetX = 215;
            }
            else CCLOG("bad_position");
            break;;

        case EventKeyboard::KeyCode::KEY_UP_ARROW:
        case EventKeyboard::KeyCode::KEY_W:
            x = event->getCurrentTarget()->getPositionX();
            y = event->getCurrentTarget()->getPositionY();
            if ((x == 135 && y == 170)|| (x == 340 && y == 310))
            {
```

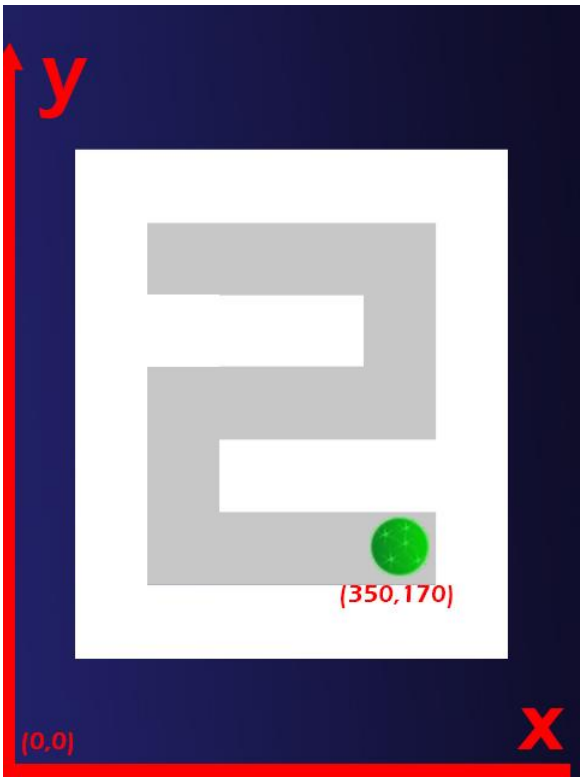
```
                CCLOG("good_position");
                offsetY = 140;
            }
            else CCLOG("bad_position");
            break;
        case EventKeyboard::KeyCode::KEY_DOWN_ARROW:
            x = event->getCurrentTarget()->getPositionX();
            y = event->getCurrentTarget()->getPositionY();
            if ((x == 135 && y == 445)||(x == 340 && y == 445))
            {
                CCLOG("good_position");
                offsetY = -205;
            }
            if ( (x == 135 && y == 310)|| (x == 340 && y == 450))
            {
                CCLOG("good_position");
                offsetY = -140;
            }
            else CCLOG("bad_position");
            break;
        }
        auto moveTo = MoveTo::create(0.2, Vec2(event->getCurrentTarget()->get-
PositionX() + offsetX, event->getCurrentTarget()->getPositionY() + offsetY));
        event->getCurrentTarget()->runAction(moveTo);
        drawNode = DrawNode::create();
        drawNode->drawSegment(Vec2(x, y), Vec2(event->getCurrentTarget()->get-
PositionX() + offsetX, event->getCurrentTarget()->getPositionY() + offsetY),
45.0f, Color4F::BLUE);
        this->addChild(drawNode);
    };
    this->_eventDispatcher->addEventListenerWithSceneGraphPriority(eventListener,
sprite2);
```

Nous avons créé  un EventListener, dans ce cas un EventListenerKeyboard, implémente le gestionnaire d'événement onKeyPressed. Le premier paramètre transmis est l'énumération EventKeyboard::KeyCode,  est une valeur qui représente la touche qui a été enfoncée.

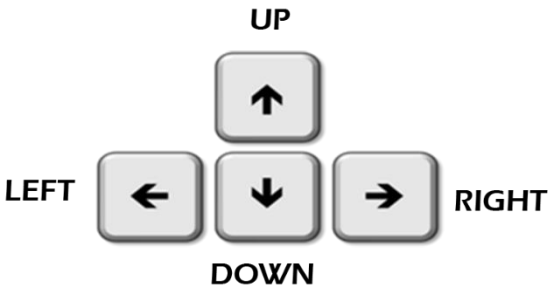
La deuxième valeur était l'event de l'événement. C'est notre sprite. Nous utilisons le pointeur d'événement pour obtenir le nœud event, et mettre à jour sa position dans une direction en fonction de la touche enfoncée .

Les positions des évènements dans le map :

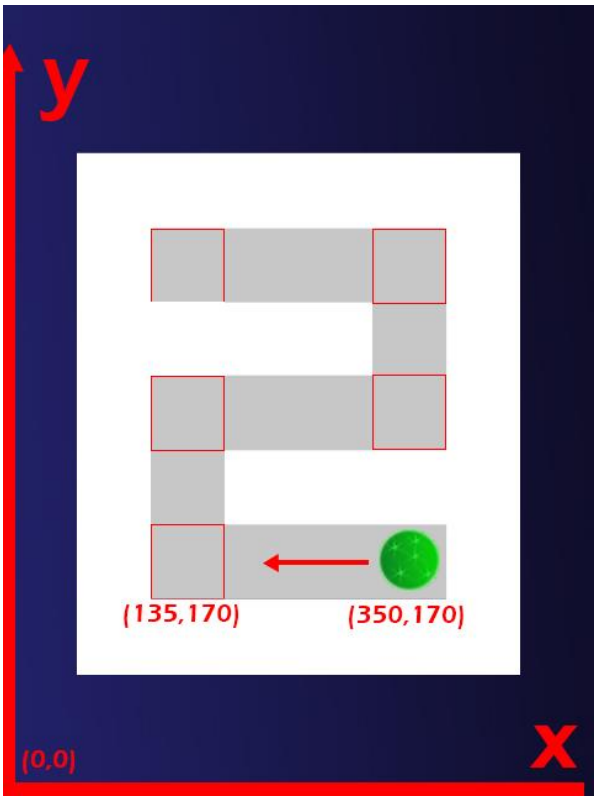
Chaque sprite ajouter à une position indiqué dans le map par rapport un repaire (x,y)




Nous avons utilisé le switch avec 4 case , chaque case représente un touche directionnelle de clavier :



La position initiale donné à la balle est (350,170) :



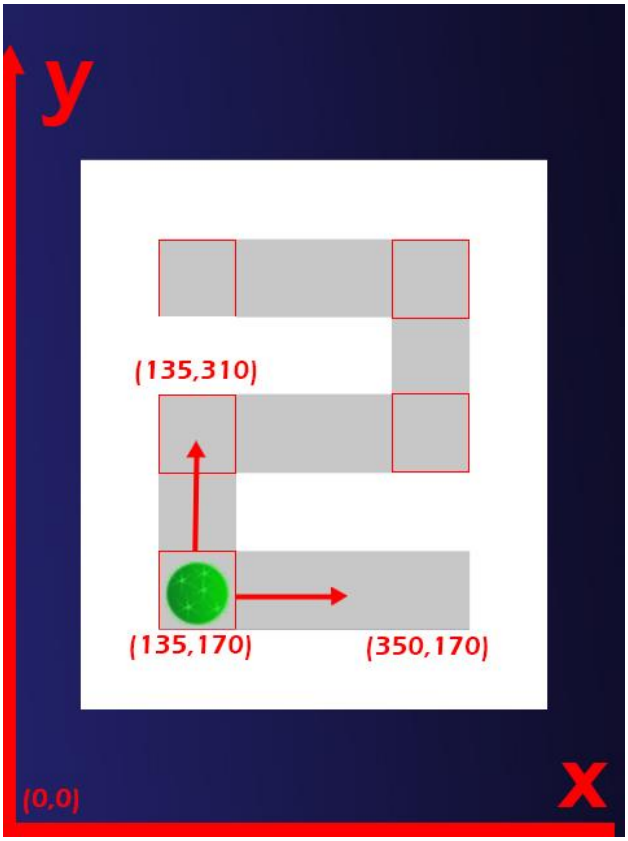

Pour le mouvement du balle de sa position initiale , il y'en a une seul possibilité c'est de la déplacer vers la position (135,170) en cliquant sur la touche KEY_LEFT.

```
if  ((x == 350 && y == 170) {  offsetX = -215; }
```

la position initiale ( x==350 ; y==170) , pou la changer pa rapport à x , il faut qu'on donne -215 à offsetX.

Moveto est une classe déjà définit à cocos2d qui sert à exécuter le déplacement du balle au position trouver ,en utilisant une sous méthode create() qui prendre en charge la vitesse d'évènement et sa position

```
auto moveTo = MoveTo::create(0.2, Vec2(event->getCurrentTarget()>getPositionX()
+ offsetX, event->getCurrentTarget()->getPositionY() + offsetY))
```




Dans la nouvelle position de la balle, on peut la déplacer vers une autre position (135,310) en cliquant sur le Botton Up, ou bien revenir a la position initiale (350,170) en cliquant sut right.

```
            if ((x == 135 && y == 310)  )
            {
                CCLOG("good_position");
                offsetX = 205;
            }
```

Pour la modification des coordonnés de la balle, on va  évaluer la valeur de offsetX (comme ci-dessus) , et la valeur du offsetY (ci-dessous), jusqu'à qu'il traites toute les cases du jeu .

```
        if ((x == 135 && y == 170)
                {
                CCLOG("good_position");
                offsetY = 140;
                  }
```

Remarque : les valeurs modifié n'ont pas en hasard , ils sont choisit par le programme Photoshop qui donne la valeur exacte du position .

Partie de coloration :

```
drawNode = DrawNode::create();
drawNode->drawSegment(Vec2(x, y), Vec2(event->getCurrentTarget()->getPositionX()
+ offsetX, event->getCurrentTarget()->getPositionY() + offsetY), 45.0f,
Color4F::BLUE);
 this->addChild(drawNode);
```

Pour la coloration du trajet du balle , on a utiliser la class DrawNode . Aussi la class DrawSegement qui permet de colorer le trajet par segment . cette dernier prendre en charge 4 valeurs :

le premier valeur : la position à l'instant  Vec2(x, y).

le deuxième valeur : la position de balle  qui va aller  à elle .

la troisième valeur : c'est le Rayon .

et la dernier valeur c'est le colore de trajet .

Pour la coloration du trajet du balle , on a utiliser la class DrawNode . Aussi la class DrawSegement qui permet de colorer le trajet par segment . cette dernier prendre en charge 4 valeurs :