目錄

# Global Variable 全域變數定義

## 通用定義

定義Pixel per meter作用於使每Pixel平均分配，通常不做更動。

public static final int PPM = 100;

介面定義與繪畫範圍，分別有邊界與繪畫比例(放大係數)。

public static String GAME\_TITLE = "New Game";

public static int SCALE = 3;

public static int WIDTH = 320;

public static int HEIGHT = 200;

繪畫相關其他定義需要依賴Screen的Resize來設定畫面顯示。

參考[場景介定 resize]

## 碰撞體定義

編輯各種物件的碰撞定義或是事件碰撞等。

public static final short NOTHING\_BIT = 0;

public static final short GROUND\_BIT = 1;

public static final short PLAYER\_BIT = 2;

public static final short OBJECT\_BIT = 4;

public static final short PORTAL\_BIT = 8; 切換地圖的判定點

定義解讀參考，「碰撞事件控制器」中之說明。

## 控制器設定

[預設] 經典鍵盤: 右手方向鍵控制

街機鍵盤: 左手方向鍵控制

控制器 (Xbox360)

[19/1/11] TODO:

檢查預設是否存在，後續需要編輯使用者輸入，現行使用固定定義可能需要配合使用Java的元件去取輸入。

按鈕定義

public static int buttonUp = Keys.UP;

public static int buttonLeft = Keys.LEFT;

public static int buttonRight = Keys.RIGHT;

public static int buttonDown = Keys.DOWN;

public static int button1; // A

public static int button2; // B

public static int button3; // X

public static int button4; // Y

public static int button5; // L Button

public static int button6; // R Button

public static int button7; // Select Button

public static int button8; // Start Button

# Screen 場景介定

場景的整體架構大略為(待補圖)：

<https://subscription.packtpub.com/book/game_development/9781783554775/7/ch08lvl1sec47/managing-multiple-screens>

Main ->

StageManager()

setScreen(Screen) : void

VarableManager()

每個場景設定控制器(Stage Manager)

[19/01/11]可能可以換成共用類別，待考量後決定更換與否。

PlayableStage : Screen ->

InputHandler()

StageVarableManager ()

@override Resize() : void 定義重畫介面

使用viewport.update(w, h, [return to center] : Boolean )

顯示動畫後並轉場至介面選單(MenuScreen)

TitleScreen

MenuScreen

## 場景類別

PlayableStage

定義基礎元件，有關攝影機視角、玩家與物件生成、碰撞體等定義

地圖繪製上使用TileMap協助繪製，需要嵌入LibGDX內建的類別使用。

### 變數定義

基礎物件相關

public Main game;

public World world;

public Box2DDebugRenderer b2dr;

public OrthographicCamera camera;

public Viewport cameraViewPort;

public boolean mapSwitching; 地圖切換判定

地圖相關

~~private TmxMapLoader mapLoader;~~

private TiledMap map;

private OrthogonalTiledMapRenderer mapRenderer;

布景相關

public Stage stage; 布景控制器依需求可能會有多個

private Image image; 布景控制器需要使用到可繪製圖形，依需求可能多個

物件相關

public Player player;

public HUD hud;

地圖變數相關

public boolean portalabled;

public String mapTo;

public Vector2 teleportTo;

public boolean playerFacingRight;

public String mapName;

## 攝影機

變數定義

public OrthographicCamera camera;

public Viewport cameraViewPort;

變數初始化與定義

public PlayableStage(Main game){

camera = new OrthographicCamera();

cameraViewPort = new FitViewport(WIDTH / PPM, HEIGHT / PPM, camera);

camera.setToOrtho(false, cameraViewPort.getWorldWidth(), cameraViewPort.getWorldHeight());

}

場景的Update事件，定義攝影機跟隨的目標與計算方式，範例為玩家的位置與攝影機的上一個位置在做調整，並使用乘上常數的方式來做漸進效果。

攝影機的位置設定，使用上數學公式中的四捨五入，目前用意於畫面撕裂的解決，但並未是最後的解決方案。 TODO:

public void update(float dt) {

float camPosition\_x = (player.b2Body.getPosition().x - camera.position.x) \* 0.1f;

float camPosition\_y = (player.b2Body.getPosition().y - camera.position.y + 0.2f) \* 0.1f;

camera.position.x += (float) Math.round(camPosition\_x \* 99f) / 99f;

camera.position.y += ((float) Math.round(camPosition\_y \* 99f) / 99f);

camera.update();

mapRenderer.setView(camera);

}

## 碰撞事件控制器

碰撞事件主要應用全域變數中的Short型態來定義哪種類型，附加在body上，使得物件可以有碰撞體可以偵測。

此範例是應用二元方式來偵測，如地板與玩家類別依據二進位表示的話為

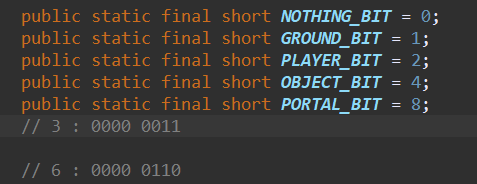
GROUND\_BIT值1 0000 0000 0000 0001

PLAYER\_BIT值2 0000 0000 0000 0010

相碰時回傳值3 0000 0000 0000 0011

物件判斷後取得依據Fixture所設定的UserData來取得該物件以及執行該物件的函式或操作等，這裡須注意在定義該fixture時即要把setUserData給指向自己或是類別，否則字串無法追溯該物件。





## 控制器

[19/1/15]

待確定是否要放置在場景內部設定，或是使用其他類別交由場景來執行？

移動速度數據交由 玩家類別中設定，在行取值

[19/1/16] 待此區分類需要移動至Sprite物件相關

方向控制依據加速度，必須限制加速上限

if (Gdx.input.isKeyPressed(Keys.RIGHT) && player.b2Body.getLinearVelocity().x <= 1.25) {

player.b2Body.applyLinearImpulse(new Vector2(0.2f, 0), player.b2Body.getWorldCenter(), true); }

方向控制依據等線性加速

b2Body.setLinearVelocity(-movementSpeed, b2Body.getLinearVelocity().y);

方向控制與不按按鍵時取消柔滑移動

else if (!Gdx.input.isKeyPressed(Keys.RIGHT) && !Gdx.input.isKeyPressed(Keys.LEFT) ){

player.b2Body.setLinearVelocity(0, player.b2Body.getLinearVelocity().y);

}

另行參考摩擦問題

卡牆pressing-against-walls-in-midair：

[2019/1/16]

設定該碰撞體中的fdef friction 為0，故摩擦力就不存在。

<https://stackoverflow.com/questions/8205088/preventing-box2d-player-from-pressing-against-walls-in-midair>

## 地圖繪製與地圖事件

地圖塊(TileSet)要再重新排列

地圖繪製順序依序：

圖層：

BackgroundColor

景色底色

CloudBack (offset Movable)

黑雲層, other?

BackgroundBack

山後景圖塊, 顏色圖塊, 草色圖塊, 天空景色塊

CloudFront (offset Movable)

白雲層, other?

BackgroundFront

山前景圖塊, 地板塊, 後雲層色圖塊, 草邊角背景

Frontground

空中地塊, 裝飾物件, 前雲層色圖塊

物件區域：

Wall

GameObject

Platform

讀取問題地圖若無法取到時，有資源檔案上的問題，然而使用setToOrtho可以解決，參考以下：

camera.setToOrtho(false, cameraViewPort.getWorldWidth() / 2, cameraViewPort.getWorldHeight() / 2);

舊有的方法：

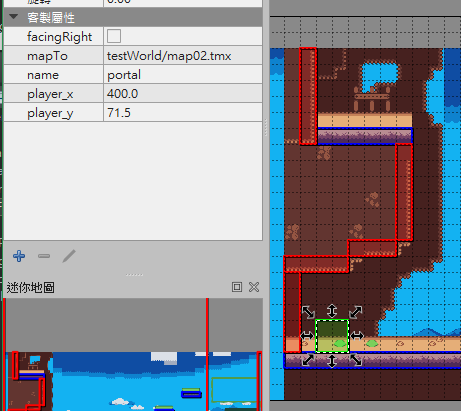
camera.position.set(cameraViewPort.getScreenWidth() / 2, cameraViewPort.getWorldHeight() / 2, 0);

另行注意，若是地圖的攝影機會出現破圖黑線參考以下計算方法：

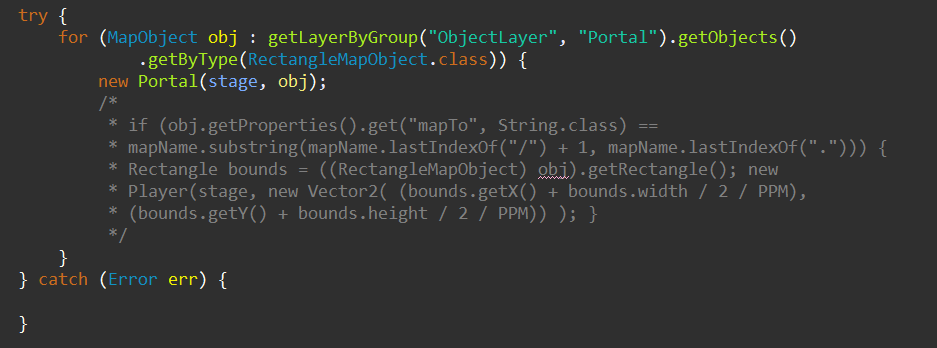
(float) Math.round(player.b2body.getPosition().x \* 100f) / 100f;

### 地圖事件

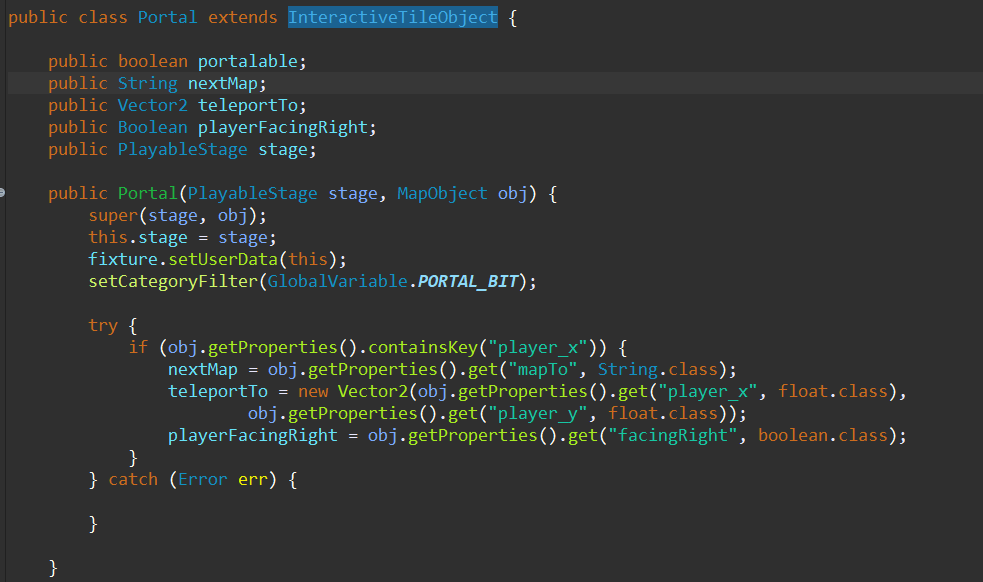
地圖的物件：”傳送門”帶有自訂義的屬性，該屬性用於定義下一張地圖的物件生成的資料，與玩家的位置點或是面向位置等，或許會再增加等。



定義完成後交由產生器，將物件自動產生出，使用Try & Catch防止空物件之錯誤，但仍須確保該物件的存有。



地圖物件的產生目前使用InteractiveTileObject生成該物件的碰撞體，後續若有可能會更動，在此留下紀錄，繼承類別後，主要設定該做的事件，此例作為碰撞體後確認物件便可以轉換地圖，同時也把所需的資料都交付與場景。



[2019/1/18]TODO:

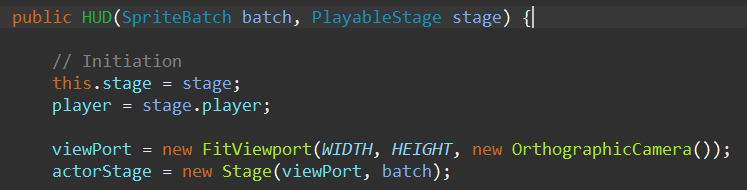
須注意的地方，後續整理時，需要地圖管理器(MapHandler)來處理個別的地圖數據，再更換場景時無法留存該圖數據，會造成藕合狀況。

## HUD 介面顯示器

介面顯示器主要應用到主畫面Main的Batch來進行繪製，產生該類別時通常應用於介面中，並把主畫面的繪畫元件交付給該物件來進行繪製。

基本定義範例：





[2019/1/18]TODO:貼圖與說明部分待補

該類別的控制主要應用Tabel、Label以及Actors等相關元件，可以做到動畫的工作，如翻轉、旋轉等，同時也具有Update()與Dispose()等函式可以做數據追溯，但也需要記得在主要的場景中加入該物件的事件，才能夠執行。

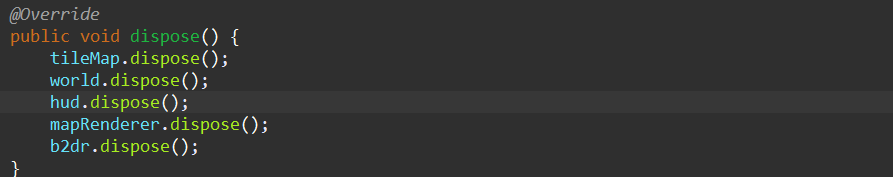
TODO: 詳細操作待補：



使用該物件需要加入的東西如範例紅色範圍內：



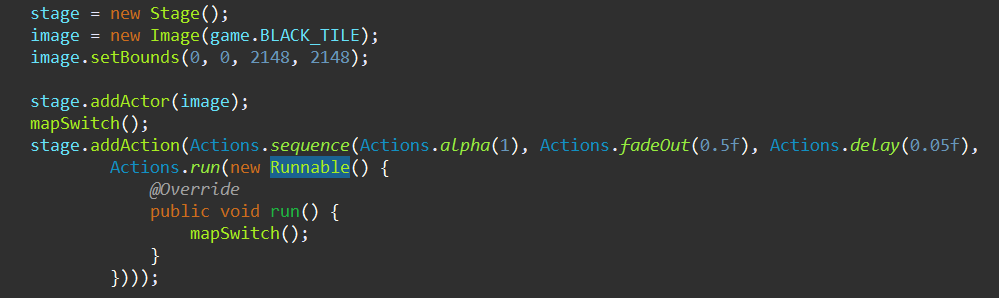




## Stage 可繪動畫舞台

動畫使用前需準備Stage與Image做為動畫基礎，該動畫可以執行重畫，與圖片影格不相同，像是浮動、旋轉、縮小放大等動作都可以由這個來製作，Image需要配合Texture來使用。

將Image加入至Stage的Actor內即可開始編輯演示動畫，如範例執行黑幕的淡進與淡出，並加入了一個匿名函式事件，來控制該動畫的起始與結束。



## 雜項問題

**回收問題**

地圖物件回收時dispose()，會同時執行hide()，若hide()再執行dispose()的狀況下會造成迴圈，而導致應用程式的關閉，須注意。

# Sprite 元件

### 圖片

匯入需要使用到工具gdx-texturepacker，來進行包裝，以節省記憶體的存取，只需存取一次，並以切割圖片的方式來交由GPU做顯示。

匯入圖片等順序大略依據以下等順序來分，圖片讀取時都是以Texture做使用，動畫部分則需要依賴到 Animation<TextureRegion>作為容器使用，作為每張Texture播放。

Atlas > (Array)TextureRegion > Texture

範例

playerAtlas = new TextureAtlas("img/player/Player.atlas");

idel = new TextureRegion(playerAtlas.findRegion("idel").getTexture(), 2, 2, 40, 50);

設定圖片大小

setBounds(x, y, width, height) 需計算PPM

設定圖片(張)

setRegion(idel);

需要將圖片與Sprite跟著物件走，需另行定義每次更新的事件

public void update(float dt) {

setPosition((b2Body.getPosition().x - getWidth() / 2)+0.05f, (b2Body.getPosition().y - getHeight() / 2)+0.15f);

}

且在主畫面時需要將物件加入Render事件中

game.batch.setProjectionMatrix(camera.combined);

game.batch.begin();

player.draw(game.batch);

game.batch.end();

Body Position and WorldCenter

https://gamedev.stackexchange.com/questions/26193/box2d-difference-between-worldcenter-and-position

### 新增動畫

需要有Animation<TextureRegion>容器，

如：

private Animation<TextureRegion> runing;

編輯動畫前，需先宣告暫存容器Array<TextureRegion> frames

並使用此容器存取與切割Image(來自Atlas的參數與圖

範例：

Array<TextureRegion> frames = new Array<TextureRegion>();

切割時需要先注意Atlas內的x, y軸與圖片大小，參數用於切割，依據動畫張數來計算該切割的範圍，以此例為動畫有八張，大小為x=50, y=50, 而xy的起始點為x=2, y=68，加入參數後使用迴圈將每張圖片切割。

for (int i = 0; i < 8; i++)

{

frames.add(new TextureRegion(playerAtlas.findRegion("run").getTexture(),

(i \* 50) + 2, 68, 50, 50));

}

完成動畫切割後將其加入容器，並提供使用，完成動畫後須將容器清空以便繼續進行存取下一個動畫繼續使用。

runing = new Animation<TextureRegion>(0.1f, frames);

frames.clear();

### 動畫播放

在完成動畫後，須將其撥放前，需先準備數個函式如以下：

變數

State currState : State

State prevState : State

函式

getState() : State (from enum)

getFrame(float Delta) : TextureRegion

update(float Delta) : void

定義狀態的取得，這些定義來自每個Sprite個別的Enum設定即可，定義個別情況時所回饋的狀態。

範例：

public State getState() {

if (b2Body.getLinearVelocity().x != 0)

return State.RUNNING;

else

return State.IDEL;

}

取得動畫所使用的函式，須配合變數，currState與prevState表示目前與前一張是哪個動畫，其中使用currState作為接收的容器取得當前狀態。region定義為容器，回傳提供給Sprite的函式setRegion使用用於更改圖片。

動畫的狀態使用switch來定義什麼狀態時該取得那些動畫，在該狀態時值行那些動畫，並且重設圖片大小。

public TextureRegion getFrame(float dt) {

currState = getState();

TextureRegion region;

switch (currState) {

case RUNNING:

region = runing.getKeyFrame(stateTimer, true);

setBounds(b2Body.getWorldCenter().x - (50 / PPM / 2), b2Body.getWorldCenter().y - (50 / PPM / 2), 50 / PPM, 50 / PPM);

break;

動畫需要判斷左右邊方向，依據下列的方式來做判斷，也可以使用其他的方式來做決定，依據範例使用若該物件的加速度大於或小於時決定翻轉的方向。

if ((b2Body.getLinearVelocity().x < 0 || !runningRight) && !region.isFlipX()) {

region.flip(true, false);

runningRight = false;

} else if ((b2Body.getLinearVelocity().x > 0 || runningRight) && region.isFlipX()) {

region.flip(true, false);

runningRight = true;

}

[TODO: 待改, 需要說明更清楚]

動畫狀態的時間定義，若動畫已經切換了，將動畫的計時器歸零，並記錄下前一個狀態的資料。

stateTimer = (currState == prevState) ? stateTimer + dt : 0;

prevState = currState;

return region;

最後在更新事件中加入設定圖片，以多張的方式來撥放形成動畫。

public void update(float dt) {

setRegion(getFrame(dt));

}

動畫播放與設定

# 常見未分類問題

事件碰撞錯誤java.lang.String cannot be cast to：

[2019/1/16]

若設定的fixture在被創建至world時，所定義的userData需要為類別

若是以string則會有cast錯誤上的狀況須注意。

需要再另想辦法確認碰撞到的物件是依據字串的方式。

震動動畫

https://www.netprogs.com/libgdx-screen-shaking/

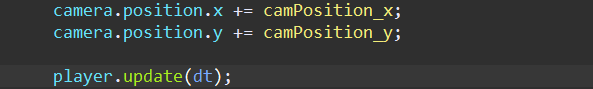
跳躍中可以穿越的平台與 腳的碰撞體

需要設定”地板” 與”平台”兩種類型 以及配合狀態去決定什麼時候落下或穿越

**動畫更新順序**

[2019/01/22]

動畫若在攝影機前，若攝影機也有使用到追溯物件位置的偏移修正時，該物件建議在攝影機位置調整後再加入該物件的繪製，否則會有距離上的差異。



**區塊地圖分裂**(Edge padding、Texture bleeding、Black line)

[2019/01/22]

區塊地圖若有分裂改使用另寫的地圖繪製工具

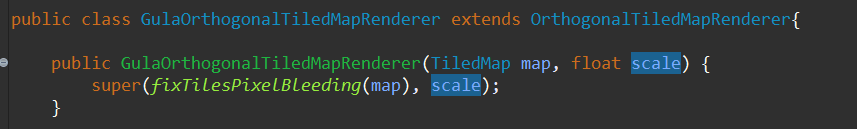
將OrthogonalTiledMapRenderer更換為GulaOrthogonalTiledMapRenderer

自訂義的繪製工具，該檔案來自：

<https://www.badlogicgames.com/forum/viewtopic.php?f=17&t=17497>

程式碼紀錄於附錄並參考之。

改寫時，若有地圖需要以PPM計算時，在繼承的地方加入float的格式即可。



# 附錄

## 地圖繪製器 GulaOrthogonalTiledMapRenderer

public class GulaOrthogonalTiledMapRenderer extends OrthogonalTiledMapRenderer {

public GulaOrthogonalTiledMapRenderer(TiledMap map, float scale) {

super(fixTilesPixelBleeding(map), scale);

}

private static TiledMap fixTilesPixelBleeding(TiledMap tiledMap) {

for (MapLayer layer : tiledMap.getLayers()) {

if (!layer.isVisible())

continue;

if (layer instanceof TiledMapTileLayer)

fixTilePixelBleeding((TiledMapTileLayer) layer);

}

return tiledMap;

}

private static void fixTilePixelBleeding(TiledMapTileLayer layer) {

for (int x = 0; x < layer.getWidth(); x++) {

for (int y = 0; y < layer.getHeight(); y++) {

Cell cell = layer.getCell(x, y);

if (cell != null)

fixPixelBleeding(cell.getTile().getTextureRegion());

}

}

}

private static void fixPixelBleeding(TextureRegion region) {

float fix = 0.01f;

float x = region.getRegionX();

float y = region.getRegionY();

float width = region.getRegionWidth();

float height = region.getRegionHeight();

float invTexWidth = 1f / region.getTexture().getWidth();

float invTexHeight = 1f / region.getTexture().getHeight();

region.setRegion((x + fix) \* invTexWidth, (y + fix) \* invTexHeight, (x + width - fix) \* invTexWidth,

(y + height - fix) \* invTexHeight);

}

}