TUGAS PEMROGRAMAN 4

RELASI ANTAR CLASS DALAM FLAPPY COW



Nama :

Mochamad Ray Farandy Lasahido

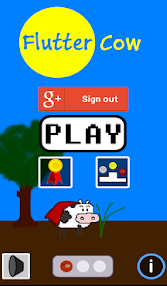
4210161019

PROGRAM STUDI TEKNOLOGI GAME

POLITEKNIK ELEKTRONIKA NEGERI SURABAYA

2018

Relasi Antar Class dalam Permainan Flappy Cow



**Definisi**

1. Agregasi

Agregasi adalah hubungan antar obyek yang searah yang salah satu obyeknya tidak hanya dapat dimiliki/dikuasai oleh obyek tertentu tapi juga obyek lain dan juga antar obyek tidak bertanggung jawab terhadap eksistensi dan jangka hidup obyek lainnya.

1. Komposisi

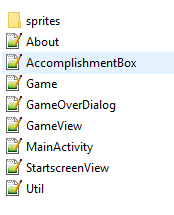
Komposisi adalah hubungan antar obyek yang salah satu obyeknya hanya dapat dimiliki/dikuasai oleh obyek yang bersangkutan dan juga antar obyek saling bertanggung jawab terhadap eksistensi dan jangka hidup obyek tersebut.

1. Inheritance

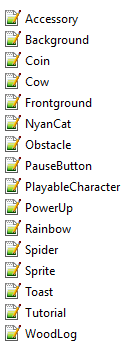
Inheritance adalah hubungan antar obyek layaknya hubungan anak dan orang tua. obyek yang satu mempunyai unsur-unsur dari obyek orang tuanya dan tidak berlaku sebaliknya. Jika obyek orang tua hancur maka akan mempengaruhi obyek lainnya.

**Class in Game**

Main Class :



Dalam folder sprites :



Main Class :

Main Class dalam game ini berguna untuk inisiasi pertama dalam menjalankan game (Start Game Awal).

* MainActivity : Memulai Aplikasinya
* StartscreenView : Membuat objek dalam menu (membuat main menu)
* Game : Memulai inisiasi gamenya
* GameView : Membuat objek dalam gamenya, serta mulai bekerjanya game mekanik
* Util : untuk render game

Class dalam folder sprites :

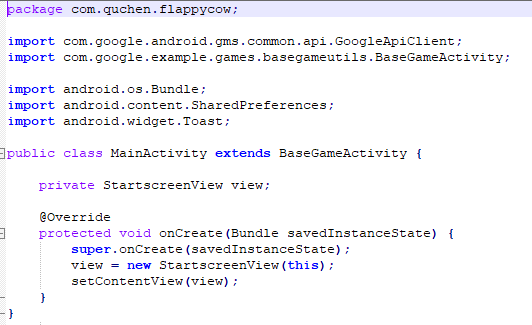
* Sprite : Template class yang digunakan untuk membuat suatu entity objek (jadi objeknya ada “posisi dan wujud” dalam gamenya).

**Relasi**

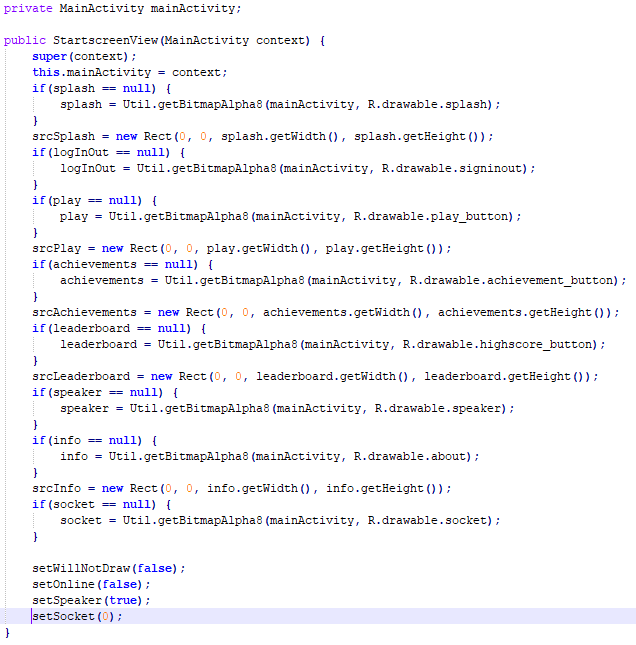
Komposisi

Hubungan ini dapat ditemukan antara MainActivity dan StartscreenView serta Game dan GameView. Jika StartscreenView hilang maka aplikasi tidak akan jalan karena hilang tampilan visualnya, sama halnya dengan MainActivity.

MainActivity.Java



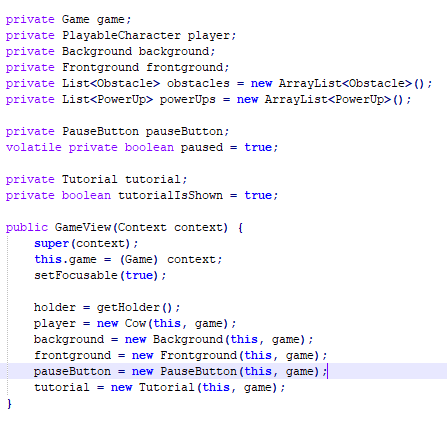
StartscreenView.Java



Agregasi

Hubungan ini terdapat di antara objek-objek yang ada dalam GameView membuat game ini ada dan dapat di jalankan, tapi jika salah satu objek hilang (misal PlayableCharacter.java) maka objek lainnya akan tetap ada.

GameView.Java



Inheritance

Penerapan inheritance ada pada Sprite.Java dengan class-class di sekitarnya (Cow, WoodLog, Spider, PlayableCharacter, dan lain-lain). Dengan mewarisi sifat Sprite, maka class-class ini mempunyai bentuk “Rect” serta posisi “X & Y”.

/\*\*

\* The template for every game object

\*

\* @author Lars Harmsen

\* Copyright (c) <2014> <Lars Harmsen - Quchen>

\*/

package com.quchen.flappycow.sprites;

import com.quchen.flappycow.Game;

import com.quchen.flappycow.GameView;

import android.graphics.Bitmap;

import android.graphics.Canvas;

import android.graphics.Rect;

public abstract class Sprite {

/\*\* The bitmaps that holds the frames that should be drawn \*/

protected Bitmap bitmap;

protected int height, width;

protected int x, y;

protected float speedX, speedY;

protected Rect src;

protected Rect dst;

protected byte col, row;

protected byte colNr = 1;

protected short frameTime;

protected short frameTimeCounter;

/\*\* The GameView that holds this Sprite \*/

protected GameView view;

/\*\* The context \*/

protected Game game;

public Sprite(GameView view, Game game){

this.view = view;

this.game = game;

frameTime = 1;

src = new Rect();

dst = new Rect();

}

/\*\*

\* Draws the frame of the bitmap specified by col and row

\* at the position given by x and y

\* @param canvas Canvas that should be drawn on

\*/

public void draw(Canvas canvas){

src.set(col\*width, row\*height, (col+1)\*width, (row+1)\*height);

dst.set(x, y, x+width, y+height);

canvas.drawBitmap(bitmap, src, dst, null);

}

/\*\*

\* Modifies the x and y coordinates according to the speedX and speedY value

\*/

public void move(){

// changeToNextFrame();

// Its more efficient if only the classes that need this implement it in their move method.

x+= speedX;

y+= speedY;

}

/\*\*

\* Changes the frame by cycling through the columns.

\*/

protected void changeToNextFrame(){

this.frameTimeCounter++;

if(this.frameTimeCounter >= this.frameTime){

this.col = (byte) ((this.col+1) % this.colNr);

this.frameTimeCounter = 0;

}

}

/\*\*

\* Checks whether this sprite is so far to the left, it's not visible anymore.

\* @return

\*/

public boolean isOutOfRange(){

return this.x + width < 0;

}

/\*\*

\* Checks whether the sprite is touching this.

\* Seeing the sprites as rectangles.

\* @param sprite

\* @return

\*/

public boolean isColliding(Sprite sprite){

if(this.x + getCollisionTolerance() < sprite.x + sprite.width

&& this.x + this.width > sprite.x + getCollisionTolerance()

&& this.y + getCollisionTolerance() < sprite.y + sprite.height

&& this.y + this.height > sprite.y + getCollisionTolerance()){

return true;

}

return false;

}

/\*\*

\* Checks whether the sprite is touching this.

\* With the distance of the 2 centers.

\* @param sprite

\* @return

\*/

public boolean isCollidingRadius(Sprite sprite, float factor){

int m1x = this.x+(this.width>>1);

int m1y = this.y+(this.height>>1);

int m2x = sprite.x+(sprite.width>>1);

int m2y = sprite.y+(sprite.height>>1);

int dx = m1x - m2x;

int dy = m1y - m2y;

int d = (int) Math.sqrt(dy\*dy + dx\*dx);

if(d < (this.width + sprite.width) \* factor

|| d < (this.height + sprite.height) \* factor){

return true;

}else{

return false;

}

}

/\*\*

\* Checks whether the point specified by the x and y coordinates is touching the sprite.

\* @param x

\* @param y

\* @return

\*/

public boolean isTouching(int x, int y){

return (x > this.x && x < this.x + width

&& y > this.y && y < this.y + height);

}

/\*\*

\* What should be done, when the player collide with this sprite?

\*/

public void onCollision(){

// Every subclass has to specify this itself

}

/\*\*

\* Checks whether the sprite is touching the ground or the sky.

\* @return

\*/

public boolean isTouchingEdge(){

return isTouchingGround() || isTouchingSky();

}

/\*\*

\* Checks whether the sprite is touching the ground.

\* @return

\*/

public boolean isTouchingGround(){

return this.y + this.height > this.view.getHeight() - this.view.getHeight() \* Frontground.GROUND\_HEIGHT;

}

/\*\*

\* Checks whether the sprite is touching the sky.

\* @return

\*/

public boolean isTouchingSky(){

return this.y < 0;

}

/\*\*

\* Checks whether the play has passed this sprite.

\* @return

\*/

public boolean isPassed(){

return this.x + this.width < view.getPlayer().getX();

}

public int getX() {

return x;

}

public void setX(int x) {

this.x = x;

}

public int getY() {

return y;

}

public void setY(int y) {

this.y = y;

}

public float getSpeedX() {

return speedX;

}

public void setSpeedX(float speedX) {

this.speedX = speedX;

}

public float getSpeedY() {

return speedY;

}

public void setSpeedY(float speedY) {

this.speedY = speedY;

}

public int getWidth() {

return width;

}

/\*\*

\* Gives a value that will be tolerated when touching a sprite.

\* Because my images have some whitespace to the edge.

\* @return

\*/

private int getCollisionTolerance(){

// 25 @ 720x1280 px

return game.getResources().getDisplayMetrics().heightPixels / 50;

}

}

Background.Java

/\*\*

\* Manages the Bitmap for the background

\*

\* @author Lars Harmsen

\* Copyright (c) <2014> <Lars Harmsen - Quchen>

\*/

package com.quchen.flappycow.sprites;

import com.quchen.flappycow.Game;

import com.quchen.flappycow.GameView;

import com.quchen.flappycow.R;

import com.quchen.flappycow.Util;

import android.graphics.Bitmap;

import android.graphics.Canvas;

public class Background extends Sprite {

/\*\* Static bitmap to reduce memory usage \*/

public static Bitmap globalBitmap;

public Background(GameView view, Game game) {

super(view, game);

if(globalBitmap == null){

globalBitmap = Util.getDownScaledBitmapAlpha8(game, R.drawable.bg);

}

this.bitmap = globalBitmap;

}

/\*\*

\* Draws the bitmap to the Canvas.

\* The height of the bitmap will be scaled to the height of the canvas.

\* When the bitmap is scrolled to far to the left, so it won't cover the whole screen,

\* the bitmap will be drawn another time behind the first one.

\*/

@Override

public void draw(Canvas canvas) {

double factor = (1.0 \* canvas.getHeight()) / bitmap.getHeight();

if(-x > bitmap.getWidth()){

// The first bitmap is completely out of the screen

x += bitmap.getWidth();

}

int endBitmap = Math.min(-x + (int) (canvas.getWidth() / factor), bitmap.getWidth());

int endCanvas = (int) ((endBitmap + x) \* factor) + 1;

src.set(-x, 0, endBitmap, bitmap.getHeight());

dst.set(0, 0, endCanvas, canvas.getHeight());

canvas.drawBitmap(this.bitmap, src, dst, null);

if(endBitmap == bitmap.getWidth()){

// draw second bitmap

src.set(0, 0, (int) (canvas.getWidth() / factor), bitmap.getHeight());

dst.set(endCanvas, 0, endCanvas + canvas.getWidth(), canvas.getHeight());

canvas.drawBitmap(this.bitmap, src, dst, null);

}

}

}

Refrensi :

* <http://www.learncpp.com/cpp-tutorial/103-aggregation/>
* <http://www.learncpp.com/cpp-tutorial/102-composition/>
* <https://www.tutorialcup.com/cplusplus/inheritance.htm>
* <https://github.com/cubei/FlappyCow>