Phát triển ứng dụng Smartphone

Tài liệu lưu hành nội bộ

Đây là tài liệu tham khảo sử dụng trong môn học Lập trình ứng dụng Smartphone – Android được tổng hợp, biên soạn từ nhiều nguồn bởi các thành viên của Nhóm nghiên cứu và ứng dụng công nghệ A106-Đại học Hoa Sen.

Phát triển ứng dụng Smartphone

Phần 04: Đồ họa 2D

Lê Đức Huy

Email: leduchuy89vn@gmail.com

Mục lục

1 Khái niệm cơ bản

Trên Android, để vẽ bất thứ gì ta cần phải có bốn thành phần cơ bản:

- Một đối tượng kiểu Bitmap để giữ các pixel cần vẽ.
- Một đối tượng chứa nét vẽ cần vẽ ra(Có thể là Rect, Path, Bitmap…).
- Một đối tượng kiểu Paint dùng để định nghĩa màu sắc, style… dùng để vẽ ra màn hình.
- Một đối tượng Canvas dùng để thực thi lệnh vẽ.

Để làm rỏ những khái niệm trên ta tiến hành tạo một lớp CustomView cài đặt lại lớp View của Android để định nghĩa một đối tượng đồ họa mới. Trên đối tượng mới định nghĩa này, ta sẽ tự vẽ ra giao diện mà mình mong muốn.

2 Xây dựng CustomView

Để tạo một CustomView ta tiến hành tạo một class với nội dung như sau:

```
package niit.android;

import android.content.Context;
import android.util.AttributeSet;
import android.view.View;

public class CustomView extends View{

	public CustomView(Context context){
		super(context);
	}

	public CustomView(Context context, AttributeSet attrs) {
		super(context, attrs);
	}

	@Override
	protected void onDraw(Canvas canvas) {

	}

}
```

Lớp CustomView sẽ extend class View của Android để kế thừa lại các phương thức cũng như các thuộc tính của lớp View. Trong đó ta sẽ cài đặt lại 02 phương thức khởi tạo cho lớp CustomView và ghi đè phương thức onDraw() của lớp View, đây chính là phương thức sẽ vẽ ra giao diện của CustomView.

Với phương thức khởi tạo **public CustomView(Context context)** chỉ nhận vào một tham số là một đối tượng Context. Đây là đối tượng này cho phép truy xuất đến các đối tượng cũng như các dịch vụ của hệ thống. Đối tượng này cần thiết để vẽ một giao diện ra màn hình thiết bị.

Với phương thức khởi tạo thứ hai: **public CustomView(Context context, AttributeSet attrs)** có nhận vào thêm một tham số là AttributeSet attrs. Đây là đối tượng chứa các thuộc tính của đối tượng đồ họa sẽ được khởi tạo. Việc tạo phương thức khởi tạo thứ hai là cần thiết nếu bạn mong muốn sử dụng đối tượng đồ họa này trong một file giao diện .xml có chỉ định các thuộc tính như ví dụ sau:

Cho một file xml có nội dung như sau:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
     android:orientation="vertical"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     >
<niit.android.CustomView
     android:id="@+id/graphicView"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     />
</LinearLayout>
```

Trong file xml kể trên ta có sử dụng một thẻ có tên **niit.android.CustomView**, đây chính là đối tượng đồ họa ta vừa tạo ra ở trên. Trong đó niit.android là package của project hiện tại. Thẻ CustomView ở trên có ba thuộc tính là *android:id*, *android:layout_width* và *android:layout_height*. Ta có thể tạo một Activity có nội dung là file xml kể trên:

```
public class main extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
          setContentView(R.layout.main);
    }
}
```

Lúc này Android sẽ tự tạo ra đối tượng CustomView và sử dụng phương thức khởi tạo **public CustomView(Context context, AttributeSet attrs)** để khởi tạo đối tượng với các thuộc tính là *android:id*, *android:layout_width* và *android:layout_height* được truyền vào thông qua đối tượng attrs.

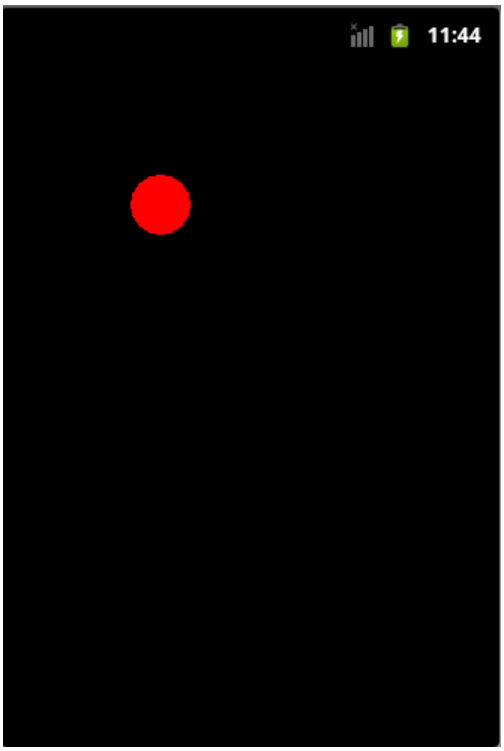
Đến đây ta đã tạo xong lớp CustomView, tuy nhiên nếu chạy chương trình ở bước này ta chỉ được một giao diện trống rỗng. Để vẽ giao diện cho CustomView ta sẽ tiến hành cài đặt phương thức onDraw() với nội dung như sau:

```
@Override
protected void onDraw(Canvas canvas) {
	Paint cPaint = new Paint();
	cPaint.setColor(Color.RED);

	Path cPath = new Path();
	cPath.addCircle(100, 100, 20, Direction.CW);

	canvas.drawPath(cPath, cPaint);
}
```

Như đã đề cập ở mục trên để vẽ ra màn hình ta tạo một đối tượng Paint được dùng để định nghĩa màu sắc, style… ta cần vẽ ra. Gọi phương thức setColor(…) để gán màu sắc của nét vẽ cần vẽ ra. Sau đó ta tiến hành khai báo một đối tượng Path để chứa các nét vẽ và gọi phương thức addCircle(…) để thêm một nét vẽ hình tròn vào đối tượng cPath. Sau đó gọi canvas.drawPath(cPath,cPaint) để vẽ ra các nét vẽ được chứa trong đối tược cPath với màu sắc được định nghĩa trong đối tượng cPaint. Khi thực thi chương trình ta sẽ được kết quả như sau:



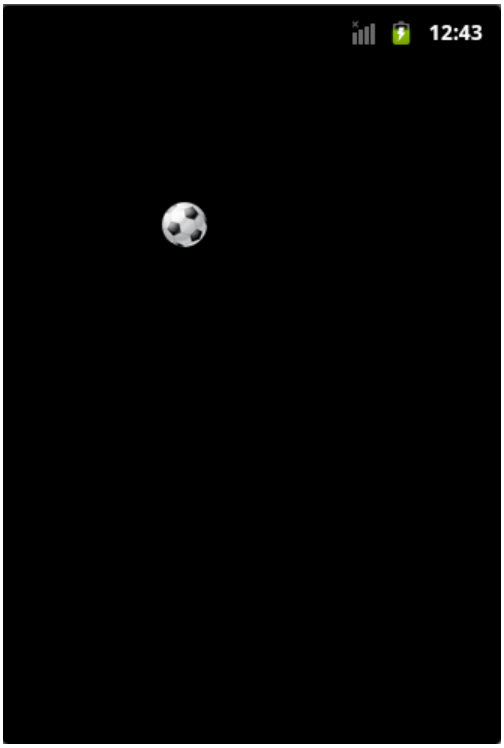
Đến đây ta đã có được một CustomView theo ý muốn của mình. Ngoài việc vẽ ra các nét vẽ đơn giản như: đường thẳng, đường tròn, hình vuông ta còn có thể vẽ file hình ảnh bằng cách sau:

```
@Override
protected void onDraw(Canvas canvas) {
      Paint bPaint = new Paint();

      Bitmap bitmap=BitmapFactory.decodeResource(getResources(),
R.drawable.ball);
      canvas.drawBitmap(bitmap, 100, 100,bPaint );
}
```

Với trường hợp muốn vẽ một file hình ảnh lên CustomView ta tạo một đối tượng Bitmap để chứa các pixel cần vẽ lên màn hình. Đối tượng Bitmap là một kiểu tổ chức bộ nhớ hoặc định dạng tệp tin dùng để lưu trữ ảnh số. Đối tượng Bitmap đơn giản là một mảng hai chiều, trong đó mỗi phần tử chứa một giá trị màu RGB tương ứng với một điểm màu trên bức ảnh số hoặc điểm màu được vẽ lên màn hình.

Ta dùng BitmapFactory.decodeResource(…) để tạo đối Bitmap từ một tệp tin hình ảnh lưu trong thư mục drawable. Sau khi thực thi chương trình ta sẽ được kết quả như sau:



Sau khi vẽ được trái banh ra màn hình ta sẽ làm cho trái banh di chuyển trong phạm vi đối tượng CustomView. Khi trái banh chạm biên ngang hay chạm biên dọc thì trái banh sẽ bị dội ngược lại.

Ta bổ sung thêm các thuộc tính sau:

```
Public    int  x=0;
public    int  y=0;
private   int  Vx=1,Vy=1;
private   int  diameter = 30;
```

Trong đó x, y là tọa độ của trái banh trên màn hình. Vx, Vy lần lượt là vận tốc theo phương dọc và phương ngang của trái banh. Sau đó ta bổ sung thêm phương thức diChuyenBanh với nội dung như sau:

```
protected void diChuyenBanh() {

	x=x+Vx;
	y=y+Vy;

	if(x<=0||(x>=getWidth()-diameter))
	{
		Vx=-Vx;
	}
	if(y<=0||y>=getHeight()-diameter)
	{
		Vy =-Vy;
	}
	invalidate();

}
```

Phương thức này sẽ lần lượt cộng tọa độ x, y của trái banh với Vx, Vy để thay đổi vị trí của trái banh. Sau khi thanh đổi tọa độ x, y của trái banh sẽ tiến hành kiểm tra xem tọa độ mới của trái banh đã vượt qua biên ngang và biên dọc của CustomView chưa. Nếu tọa độ mới đã vượt qua biên dọc và biên ngang của trái banh thì đổi hướng của vectơ vận tốc để cho trái banh chuyển động theo hướng ngược lại (Cho trái banh dội ngược lại). Cuối cùng gọi phương thức invalidate(), đây là phương thức sẽ gọi lại phương thức onDraw để vẽ lại giao diện của CustomView. Để CustomView có thể chuyển động ta tiến hành thay đổi phương thức onDraw với nội dung như sau:

```
@Override
protected void onDraw(Canvas canvas) {

	Paint bPaint = new Paint();

	Bitmap bitmap=BitmapFactory.decodeResource(getResources(),
R.drawable.ball);
	canvas.drawBitmap(bitmap, x, y,bPaint );
	diChuyenBanh();

}
```

Với phương thức onDraw được định nghĩa như ở trên đây thì CustomView sẽ vẽ ra màn hình trái banh tại vị trí (x,y) với giá trị mặc định (0,0). Sau đó gọi phương thức di chuyển banh để tính toán lại tọa độ (x,y) mới. Trong phương thức diChuyenBanh ta có gọi phương thức invalidate(), đây là một cách gián tiếp để gọi đệ quy lại onDraw. Bằng cách này CustomView sẽ liên tục được vẽ đi vẽ lại với giá trị (x,y) thay đổi qua mỗi lần vẽ lại.

3 Lập trình luồng trong đồ họa 2D

Để làm game trên bất kì nền tảng nào ta cũng cần phải quen thuộc với khái niệm lập trình luồng. Khái niệm luồng đơn giản là việc giao một công việc cần thực hiện cho một đối tượng. Sau khi giao công việc cho đối tượng đó thì nó sẽ được thực thi "đồng thời" với các công việc chính đang được thực hiện. Bằng cách này, một công việc có thể được thực thi đồng thời với các công việc khác mà không phải chờ đợi để được thực thi tuần tự như lập trình truyền thống.

Để làm quen với luồng ta tiến hành thực hiện ví dụ sau:

Tiến hành bổ sung thêm một đồng hồ đếm giờ cho CustomView. Tiến hành đếm ngược từ 30s trở về 0 và in ra "Thời gian: 29". Khi đếm đến 0 thì ngừng không in trái banh ra màn hình nữa mà thay vào đó là in câu "Bạn đã thua rồi!".

Ở đây ta có thể thấy được có hai công việc đang được thực thi đồng thời:

- Công việc 01: Vẽ ra màn hình đoạn text: "Thời gian XX" và vẽ ra trái banh tại tọa độ (x,y).
- Công việc 02: Tuần tự giảm giá trị của biến đếm mỗi lần một đơn vị. Sau khi giảm giá trị thì ngừng lại 1 giây và thực hiện lại công việc.

Để làm được việc này ta bổ xung thêm một thuộc tính **int count = 30;** và một lớp nội trong CustomView với nội dung như sau:

```
private class TimeCountThread extends Thread{
      @Override
      public void run() {
            while(count>0){
                  try {
                        count--;
                        sleep(1000);
                  } catch (InterruptedException e) {
                        e.printStackTrace();
                  }
            }
      }

}
```

Lớp TimeCountThread extend lớp Thread dùng để khai báo một luồng cho java. Lớp này chỉ có một phương thức chính là run(). Phương thức này sẽ được thực thi khi luồng được khởi động. Luồng được tạo ra từ lớp TimeCountThread làm nhiệm vụ đếm ngược biến count, mỗi lần đếm xong sẽ ngừng lại 1000 ms và thực hiện lại công việc cho đến khi count<=0.

Bổ xung thêm thuộc tính cho lớp CustomView:

```
TimeCountThread timeCountThread = new TimeCountThread();
```

Sau khi thực hiện xong các bước trên ta đã được một luồng tên là timeCountThread. Tuy nhiên đến đây thì luông timeCountThread này vẫn chưa được khởi động. Để khởi động luồng trên ta bổ xung vào hai phương khởi tạo của CustomView để sau khi khởi tạo đối tượng xong sẽ khởi động luồng timeCountThread. Nội dung thay đổi như sau:

```
package niit.android;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.util.AttributeSet;
import android.view.View;

public class CustomView extends View{

	private int count = 30;

	public int x=0;
	public int y=0;
	private   int  Vx=1,Vy=1;
	private int diameter = 30;

	TimeCountThread timeCountThread = new TimeCountThread();

	public CustomView(Context context, AttributeSet attrs) {
		super(context, attrs);
		timeCountThread.start();
	}

	public CustomView(Context context){
		super(context);
		timeCountThread.start();
	}

	@Override
	protected void onDraw(Canvas canvas) {

		if(count>0)
		{

			Paint tPaint = new Paint();
			tPaint.setColor(Color.RED);

			canvas.drawText("Time: " + count, 5, 20, tPaint);
```

```
                Paint bPaint = new Paint();

                Bitmap
bitmap=BitmapFactory.decodeResource(getResources(),
R.drawable.ball);
                canvas.drawBitmap(bitmap, x, y,bPaint );

                diChuyenBanh();

           } else {
                Paint tPaint = new Paint();
                tPaint.setColor(Color.RED);

                canvas.drawText("Bạn đã thua rồi",30,30,tPaint);
           }

     }

     protected void diChuyenBanh() {

          x=x+Vx;
          y=y+Vy;

          if(x<=0||(x>=getWidth()-diameter))
          {
               Vx=-Vx;
          }
          if(y<=0||y>=getHeight()-diameter)
          {
               Vy =-Vy;
          }
          invalidate();
     }

     private class TimeCountThread extends Thread{

          @Override
          public void run() {
               while(count>0){
                    try {
                         sleep(1000);
                         count--;
                    } catch (InterruptedException e) {
                         e.printStackTrace();
                    }
               }
```

```
            }

        }

    }
```

4 Game với SurfaceView

SurfaceView là một bề mặt chuyên sử dụng để vẽ. Nó là một lớp kế thừa từ View và bổ sung thêm một số phương thức cũng như tính năng hỗ trợ cho việc vẽ liên tục lên bề mặt. Ở phần này của tài liệu ta sẽ tiến hành tạo một lớp kế thừa từ SurfaceView để vẽ ra màn hình một trái banh và cho nó di chuyển như ở bài trước. Để bắt đầu ta tạo một lớp mới với nội dung như sau:

```
    package niit.android;

    import java.util.ArrayList;

    import android.content.Context;
    import android.graphics.Canvas;
    import android.view.SurfaceHolder;
    import android.view.SurfaceView;

    public class GamePanel extends SurfaceView implements
SurfaceHolder.Callback {
        public GamePanel(Context context) {
            super(context);
        }
        @Override
        public void surfaceCreated(SurfaceHolder holder) {
        }
        @Override
        public void surfaceChanged(SurfaceHolder holder, int
    format, int width, int height) {
        }
        @Override
        public void surfaceDestroyed(SurfaceHolder holder) {
        }
    }
```

Trên đây ta định nghĩa một lớp tên là GamePanel kế thừa SurfaceView và cài đặt lại interface SurfaceHolder.Callback. Trong đó có một phương thức khởi tạo và ba phương thức sử lý các sự kiện: surface được tạo ra, surface thay đổi các thông số và khi surface bị hủy.

Sau khi tạo xong một lớp như trên ta cần phải báo cho GamePanel biết đối tượng sẽ sử lý các sự kiện như surface được tạo ra, surface thay đổi các thông số và khi surface bị hủy bằng cách thay đổi phương thức khởi tạo với nội dung như sau:

```
public GamePanel(Context context) {
     super(context);
     getHolder().addCallback(this);
}
```

Bằng cách này đối tượng GamePanel hiện tại được chỉ định để sử lý các sự kiện như: surface được tạo ra, surface thay đổi các thông số và khi surface bị hủy. Khi một trong trong các sự kiện trên diễn ra nó sẽ gọi một trong ba phương thức:

```
public void surfaceCreated(SurfaceHolder holder);
public void surfaceChanged(SurfaceHolder holder, int format, int
width, int height);
public void surfaceDestroyed(SurfaceHolder holder);
```

Ba phương thức này ta sẽ được cài đặt ở phần sau.

Bổ sung thêm lớp Ball với nội dung như sau:

```
package niit.android;

import android.content.res.Resources;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;

public class Ball {
     // Hai thuoc tinh chieu rong va chieu cao cua GamePanel.
     private float gamePanelWidth = 0;
     private float gamePanelHeight = 0;
     // Hai thuoc tinh xac dinh toa do (x,y) hien tai cua trai
banh.
    private float x;
    private float y;
    // Duong kinh trai banh.
     private float diameter;
     // Van toc theo phuong ngang va theo phuong doc cua trai
banh.
    private float Vx;
    private float Vy;
    // Doi tuong Bitmap chua hinh anh trai banh.
    private Bitmap bitmap;
    // Hai phuong thuc set, get cua thuoc tinh Bitmap.
     public Bitmap getBitmap() {
          return bitmap;
     }
     public void setBitmap(Bitmap mBitmap) {
          this.bitmap = mBitmap;
     }
     // Phuong thuc gan toa do cho trai banh.
```

```
	public void setPosition(float x, float y){
		x = x;
		y = y;
	}
	// Phuong thuc gan van toc cho trai banh.
	public void setVelosity(float Vx, float Vy){
		this.Vx = Vx;
		this.Vy = Vy;
	}
	// Phuong thuc khoi tao trai banh.
	public Ball(Resources res, float x, float y, float
gamePanelWidth,float gamePanelHeight) {
		bitmap = BitmapFactory.decodeResource(res,
R.drawable.ball);
		this.x = x;
		this.y = y;
		diameter = bitmap.getWidth();

		Vx = 0;
		Vy = 0;

		this.gamePanelHeight = gamePanelHeight;
		this.gamePanelWidth = gamePanelWidth;

	}
	// Phuong thuc ve trai banh bang doi tuong Canvas duoc
truyen vao duoi dang tham so.
	public void drawBall(Canvas canvas) {
		canvas.drawBitmap(bitmap, x, y, null);
	}
	// Phuong thuc di chuyen trai banh.
	public void moveBall() {
		x += Vx;
		y += Vy;
		checkBallPosition();
	}
	// Phuong thuc kiem tra vi tri trai banh, neu vuot qua bien
ngang hoac bien doc thi doi huong trai banh lai.
	private void checkBallPosition() {
		if(x<=0||(x>=gamePanelWidth-diameter))
		{
			Vx=-Vx;
		}
		if(y<=0||y>=gamePanelHeight-diameter)
		{
			Vy =-Vy;
		}
```

```
    }
}
```

Lớp Ball này có phương thức drawBall(…) sẽ dùng để vẽ ra trái banh trên đối tượng GamePanel. Việc vẽ trái banh lên đối tượng GamePanel sẽ thực hiện thông qua đối tượng Canvas sẽ được cung cấp bởi GamePanel thông qua một tham số.

Đến đây ta cần dừng lại một chút để xem lại những gì đã có. Trước tiên ta có một đối tượng GamePanel sẽ là một màn hình game thể hiện một trái banh di chuyển. Đối tượng GamePanel sẽ là đối tượng đón nhận các sự kiện: người dùng tương tác lên màn hình, người dùng nhấn phím cứng trên màn hình… Nếu giao việc vẽ trái banh di chuyển cho đối tượng GamePanel luôn thì trong thời gian nó thực hiện lệnh vẽ trái banh thì nó không thể cùng lúc sử lý luôn cả các sự kiện người dùng tương tác với màn hình, bàn phím… Do đó ta sẽ định nghĩa một Thread để liên tục thực hiện việc vẽ trái banh ra màn hình. Tiến hành định nghĩa một lớp GameThread nằm trong GamePanel với nội dung như sau:

```
class GameThread extends Thread {
    private SurfaceHolder mHolder;
    private boolean mRun = false;

    public GameThread() {
        mHolder = GamePanel.this.getHolder();
    }

    public void setRunning(boolean run) {
        mRun = run;
    }

    @Override
    public void run() {
        Canvas canvas = null;
        while (mRun) {
            canvas = mHolder.lockCanvas();
            if (canvas != null) {
                GamePanel.this.thayDoiManHinhGame();
                GamePanel.this.veManHinhGame(canvas);
                mHolder.unlockCanvasAndPost(canvas);
            }
        }
    }
}
```

Luồng GameThread ở trên chỉ có hai thuộc tính:

- **SurfaceHolder mHolder**: là đối tượng nắm giữ một bề mặt hiển thị. Ở đây là bề mặt hiển thị của GamePanel.
- **boolean mRun**: là biến cờ dùng để điều khiển vòng lặp while trong phương thức run của luông GameThread.

Lớp GameThread có một phương thức dựng dùng để gán giá trị cho đối tượng SurfaceHolder mHolder bằng đối tượng được trả về bằng cách gọi phương thức GamePanel.this.getHolder();. Đây là cách gọi phương thức getHolder của lớp GamePanel. Phương thức này sẽ trả về đối tượng nắm giữ một bề mặt hiển thị của GamePanel.

Trên lớp GameThread ta ghi đè lại phương thức run(…). Đây chính là phương thức được thực thi khi GameThread được khởi động. Phương thức run của GameThread có một vòng lặp while với điều kiện lặp là giá trị của biến mRun. Trong vòng lặp while này ta lần lượt thực thi các lệnh:

```
canvas = mHolder.lockCanvas();
if (canvas != null) {
     GamePanel.this.changeGamePanel();
     GamePanel.this.drawGamePanel(canvas);
     mHolder.unlockCanvasAndPost(canvas);
}
```

Như đã giải thích ở trên, khi khởi tạo đối tượng GameThread ta đã gán giá trị mHolder bằng đối tượng nắm giữ bề mặt hiển thị của GamePanel. Để vẽ lên bề mặt của mHolder ta sẽ cần phải sử dụng một đối tượng canvas. Tuy nhiên để đảm bảo trong quá trình GameThread vẽ lên bề mặt mHolder không ai vẽ lên đó nữa ta thực thi lệnh lockCanvas(…). Phương thức này sẽ trả về một đối tượng canvas. Ta có thể dùng đối tượng canvas này để vẽ lên mHolder như cách vẽ lên CustomView ở mục 1 của chương này. Tiến hành gọi GamePanel.this.changeGamePanel(); để thay đổi vị trí của trái banh và gọi GamePanel.this.drawGamePanel(canvas); để vẽ trái banh ra phần màn hình bằng cách sử dụng đối tượng canvas ở trên. Sau khi vẽ xong tiến hành unclockCanvasAndPost đốit ượng canvas lên mHolder để các nét vẽ trong canvas được thể hiện lên màn hình mHolder đang nắm giữ.

Tiến hành bổ xung thêm 02 phương thức đã nhắc đến ở trên:

```
public void drawGamePanel(Canvas canvas) {
     canvas.drawColor(Color.BLACK);
     synchronized (ball) {
          ball.drawBall(canvas);
     }
}
```

Phương thức drawGamePanel() sẽ vẽ lại màn hình GamePanel bằng cách tô màu toàn bộ màn hình bằng màu đen và gọi lệnh ball.drawBall(…) truyền theo đối tượng canvas để vẽ. Bằng cách này phương thức drawBall sẽ sử dụng đối tượng canvas được cung cấp dưới dạng tham số để vẽ ra trái banh tại vị trí (x,y) là giá trị thuộc tính của nó.

```
public void changeGamePanel() {
     synchronized (ball) {
          ball.moveBall();
     }
}
```

Phương thức changeGamePanel() sẽ gọi ball.moveBall() để di chuyển trái banh tới vị trí mới dựa vào vận tốc theo phương dọc và phương ngang của trái banh hiện tại.

Đến thời điểm hiện tại, ta gần như đã hoàn tất chương trình theo yêu cầu đặt ra ban đầu. Tuy nhiên ta cần phải bổ sung thêm một chút bằng cách cài đặt lại các phương thức sau:

```
@Override
public void surfaceCreated(SurfaceHolder holder) {

     ball = new Banh(getResources(), 0, 0, this.getWidth(),
this.getHeight());
     ball.ganVanToc(5, 5);

     if (!gameThread.isAlive()) {
          gameThread = new GameThread();
          gameThread.setRunning(true);
          gameThread.start();
    }
}
```

Phương thức surfaceCreated() sẽ được gọi khi GamePanel được dựng hoàn tất. Ở đây ta cần bổ sung vài dòng lệnh để tạo mới đối tượng ball cũng như khởi động gameThread để trái banh bắt đầu di chuyển.

```
@Override
public void surfaceDestroyed(SurfaceHolder holder) {
     if (gameThread.isAlive()) {
          gameThread.setRunning(false);
        }
}
```

Phương thức surfaceDestroyed sẽ được gọi khi GamePanel đã hủy xong. Ở đây đơn giản là ta gán giá trị cho biến mRun bằng false để vòng lặp while trong phương thức run của GameThread dừng không lặp tiếp. Khi đó GameThread sẽ kết thúc công việc.

Lưu ý: Ở đây có sử dụng hai đối tượng là ball và gameThread được khai báo như là một thuộc tính của GameThread:

```
private GameThread gameThread = new GameThread();

private Ball ball;
```

```
@Override
public boolean onTouchEvent(MotionEvent event) {
     switch (event.getAction()) {
          case MotionEvent.ACTION_DOWN:
          {
               lastX = event.getX();
               lastY = event.getY();
               synchronized (balls) {
                    newBall = new Ball(getResources(),lastX,lastY);

                         newBall.setPosition(lastX, lastY);
                         newBall.setVelocity(0,0);
                         balls.add(newBall);
               }
          }
          return true;

          case MotionEvent.ACTION_MOVE:
          {
               synchronized (newBall) {
                    newBall.setPosition(event.getX(), event.getY());
               }
          }
          return true;

          case MotionEvent.ACTION_UP:
          {
               if(newBall!=null)
               {
                    synchronized (newBall) {
                         newBall.setVelocity((float)((event.getX()-
lastX)/10.0),(float)((event.getY()-lastY)/10.0));
                    }
               }
          }
          return true;

          default:

          break;
          }

        return super.onTouchEvent(event);
}
```