Exploration approfondie du dessin personnalisé dans Android

Cet article de blog a été rédigé avec l'aide de ChatGPT-4o.

Introduction

Dans ce billet de blog, nous allons explorer la classe `DrawActivity`, un exemple complet de mise en œuvre d'une vue de dessin personnalisée dans une application Android. Nous décomposerons chaque composant et les algorithmes utilisés, en expliquant en détail comment ils fonctionnent ensemble pour réaliser les fonctionnalités souhaitées.

Table des matières

Aperçu de DrawActivity

L'activité `DrawActivity` est une composante essentielle de l'application, conçue pour permettre aux utilisateurs de créer et de manipuler des dessins. Elle offre une interface intuitive où les utilisateurs peuvent utiliser différents outils de dessin, tels que des pinceaux, des formes géométriques, et des options de couleur. L'activité est également équipée de fonctionnalités pour sauvegarder, partager et modifier les dessins existants.

Fonctionnalités principales :

- **Outils de dessin** : Pinceaux, gomme, formes géométriques, etc.
- **Gestion des couleurs** : Palette de couleurs personnalisable.
- **Sauvegarde et partage** : Options pour enregistrer les dessins localement ou les partager sur les réseaux sociaux.
- **Modification** : Possibilité de modifier les dessins existants avec des outils avancés.

Code d'exemple :

```
public class DrawActivity extends AppCompatActivity {
    private CanvasView canvasView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_draw);

        canvasView = findViewById(R.id.canvas);
        // Initialisation des outils de dessin
        setupDrawingTools();
    }

    private void setupDrawingTools() {
        // Configuration des outils de dessin
    }
}
```

Cette activité est conçue pour être facilement extensible, permettant aux développeurs d'ajouter de nouvelles fonctionnalités ou de personnaliser les outils existants selon les besoins de l'application.

`DrawActivity` est l'activité principale qui gère les opérations de dessin, le recadrage des images ainsi que les interactions avec d'autres composants tels que les fragments et le téléversement d'images. Elle fournit une interface utilisateur où les utilisateurs peuvent dessiner, annuler, rétablir et manipuler des images.

```
public class DrawActivity extends Activity implements View.OnClickListener {
  // Constantes pour les codes de requête et les ID de fragment
```

```
    public static final int CAMERA_RESULT = 1;
    public static final int CROP_RESULT = 2;
    public static final int DRAW_FRAGMENT = 0;
    public static final int RECOG_FRAGMENT = 1;
    public static final int RESULT_FRAGMENT = 2;
    public static final int WAIT_FRAGMENT = 3;
    public static final int MATERIAL_RESULT = 4;
    public static final String RESULT_JSON = "resultJson";
    public static final int INIT_FLOWER_ID = R.drawable.flower_b;
    public static final int LOGOUT = 0;
    public static final int IMAGE_RESULT = 0;

    // Variables pour la gestion des images et des opérations de dessin
    String baseUrl;
    DrawView drawView;
    Bitmap originImg;
    public static DrawActivity instance;
    View dir, clear, cameraView, materialView, scale;
    ImageView undoView, redoView;
    View upload;
    String cropPath;
    Tooltip toolTip;
    int curFragmentId = -1;
    int serverId = -1;
    private Bitmap resultBitmap;
    private RadioGroup radioGroup;
    Fragment curFragment;
    int curDrawMode;
    RadioButton drawBackBtn;
    private Activity cxt;
    Uri curPicUri;
}
```

Initialisation de l'Activity

Lors de la création de l'Activity, diverses opérations d'initialisation sont exécutées, telles que la configuration des vues, le chargement des images initiales et la configuration des écouteurs d'événements.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  instance = this;
  cxt = this;
  cropPath = PathUtils.getCropPath();
  setContentView(R.layout.draw_layout);
  findView();
  setSize();
  initOriginImage();
  toolTip = new Tooltip(this);
  initUndoRedoEnable();
  setIp();
  initDrawmode();
}
```

Note: Le code Java reste en anglais, car il s'agit d'un langage de programmation et les noms de méthodes, variables, etc., ne sont généralement pas traduits.

findView()

Cette méthode initialise les vues utilisées dans l'Activity.

```
private void findView() {
  drawView = findViewById(R.id.drawView);
  undoView = findViewById(R.id.undo);
  redoView = findViewById(R.id.redo);
  scale = findViewById(R.id.scale);
  upload = findViewById(R.id.upload);
  clear = findViewById(R.id.clear);
  dir = findViewById(R.id.dir);
  materialView = findViewById(R.id.material);
  cameraView = findViewById(R.id.camera);
```

```
dir.setOnClickListener(this);
materialView.setOnClickListener(this);
undoView.setOnClickListener(this);
scale.setOnClickListener(this);
redoView.setOnClickListener(this);
clear.setOnClickListener(this);
cameraView.setOnClickListener(this);
upload.setOnClickListener(this);
initRadio();
}
```

setSize()

Définit la taille de la vue de dessin.

```
private void setSize() {
  setSizeByResourceSize();
  setViewSize(drawView);
}
```

Note : Le code reste en anglais car il s'agit d'un extrait de code Java, et les noms de méthodes et de variables ne sont généralement pas traduits pour maintenir la cohérence et la lisibilité du code.

```
private void setSizeByResourceSize() {
  int width = getResources().getDimensionPixelSize(R.dimen.draw_width);
  int height = getResources().getDimensionPixelSize(R.dimen.draw_height);
  App.drawWidth = width;
  App.drawHeight = height;
}

private void setViewSize(View v) {
  ViewGroup.LayoutParams lp = v.getLayoutParams();
  lp.width = App.drawWidth;
  lp.height = App.drawHeight;
  v.setLayoutParams(lp);
}
```

initOriginImage()

Charge l'image initiale qui sera utilisée pour le dessin.

```
private void initOriginImage() {
  Bitmap bitmap = BitmapFactory.decodeResource(getResources(), INIT_FLOWER_ID);
  String imgPath = PathUtils.getCameraPath();
  BitmapUtils.saveBitmapToPath(bitmap, imgPath);
  Uri uri1 = Uri.fromFile(new File(imgPath));
  setImageByUri(uri1);
}
```

Manipulation des images

L'Activity gère diverses opérations sur les images, telles que la définition d'une image via un URI, le recadrage et l'enregistrement des bitmaps dessinés.

setImageByUri(Uri uri)

Charge une image à partir de l'URI donné et la prépare pour le dessin.

```
private void setImageByUri(final Uri uri) {
  new Handler().postDelayed(new Runnable() {
    @Override
    public void run() {
      curPicUri = uri;
      Bitmap bitmap = null;
      try {
        if (uri != null) {
          bitmap = BitmapUtils.getBitmapByUri(DrawActivity.this, uri);
        }
      } catch (Exception e) {
        e.printStackTrace();
      }

int originW = bitmap.getWidth();
int originH = bitmap.getHeight();
if (originW != App.drawWidth || originH != App.drawHeight) {
```

```
  float originRadio = originW * 1.0f / originH;
  float radio = App.drawWidth * 1.0f / App.drawHeight;
  if (Math.abs(originRadio - radio) < 0.01) {
    Bitmap originBm = bitmap;
    bitmap = Bitmap.createScaledBitmap(originBm, App.drawWidth, App.drawHeight, false);
    originBm.recycle();
  } else {
    cropIt(uri);
    return;
  }
}
ImageLoader imageLoader = ImageLoader.getInstance();
imageLoader.addOrReplaceToMemoryCache("origin", bitmap);
originImg = bitmap;
serverId = -1;

      drawView.setSrcBitmap(originImg);
      showDrawFragment(App.ALL_INFO);
      curDrawMode = App.DRAW_FORE;
    }
  }, 500);
}
```

cropIt(Uri uri)

Lance l'activité de recadrage d'image.

```
public void cropIt(Uri uri) {
  Crop.startPhotoCrop(this, uri, cropPath, CROP_RESULT);
}
```

saveBitmap()

Enregistre le bitmap dessiné dans un fichier et le téléverse sur le serveur.

```
public void saveBitmap() {
  Bitmap handBitmap = drawView.getHandBitmap();
  Bitmap originBitmap = drawView.getSrcBitmap();
  saveBitmapToFileAndUpload(handBitmap, originBitmap);
}
```

saveBitmapToFileAndUpload(Bitmap handBitmap, Bitmap originBitmap)

Enregistre les bitmaps dans un fichier et les téléverse de manière asynchrone.

```
private void saveBitmapToFileAndUpload(Bitmap handBitmap, Bitmap originBitmap) {
  final String originPath = PathUtils.getOriginPath();
  BitmapUtils.saveBitmapToPath(originBitmap, originPath);
  final String handPath = PathUtils.getHandPath();
  BitmapUtils.saveBitmapToPath(handBitmap, handPath);
  new AsyncTask<Void, Void, Void>() {
    boolean res;
    Bitmap foreBitmap;
    Bitmap backBitmap;

@Override
protected void onPreExecute() {
    super.onPreExecute();
    showWaitFragment();
}

@Override
protected Void doInBackground(Void... params) {
  try {
    if (baseUrl == null) {
      throw new Exception("baseUrl est null");
    }
    String jsonRes = UploadImage.upload(baseUrl, serverId, Web.STATUS_CONTINUE, originPath, handPath, nu
    getJsonData(jsonRes);
    res = true;
  } catch (Exception e) {
    res = false;
    e.printStackTrace();
  }
  return null;
}

private void getJsonData(String jsonRes) throws Exception {
    JSONObject json = new JSONObject(jsonRes);
```

```
    if (serverId == -1) {
        serverId = json.getInt(Web.ID);
    }
    String foreUrl = json.getString(Web.FORE);
    String backUrl = json.getString(Web.BACK);
    String resultUrl = json.getString(Web.RESULT);
    foreBitmap = Web.getBitmapFromUrlByStream1(foreUrl, 0);
    backBitmap = Web.getBitmapFromUrlByStream1(backUrl, 0);
    resultBitmap = Web.getBitmapFromUrlByStream1(resultUrl, 0);
}

@Override
protected void onPostExecute(Void aVoid) {
  super.onPostExecute(aVoid);
  if (res) {
    showRecogFragment(foreBitmap, backBitmap);
  } else {
    Utils.toast(DrawActivity.this, R.string.server_error);
    recogNo();
  }
}

  }.execute();
}
```

Gestion des Fragments

L'Activity gère différents fragments pour traiter les divers états de l'application, tels que le dessin, la reconnaissance et l'attente.

showDrawFragment(int infoId)

Affiche le fragment de dessin.

```
private void showDrawFragment(int infoId) {
  curFragmentId = DRAW_FRAGMENT;
  curFragment = new DrawFragment(infoId);
```

```
  showFragment(curFragment);
}
```

Le code reste en anglais, car il s'agit d'un extrait de code Java. Voici une explication en français :

Cette méthode `showDrawFragment` est utilisée pour afficher un fragment spécifique dans une application Android. Voici ce qu'elle fait étape par étape :

1. `curFragmentId = DRAW_FRAGMENT;` : Elle assigne une constante `DRAW_FRAGMENT` à la variable `curFragmentId`, qui représente l'identifiant du fragment actuellement affiché.
2. `curFragment = new DrawFragment(infoId);` : Elle crée une nouvelle instance de `DrawFragment` en passant un identifiant `infoId` comme argument. Ce fragment est ensuite stocké dans la variable `curFragment`.
3. `showFragment(curFragment);` : Enfin, elle appelle une méthode `showFragment` pour afficher le fragment créé.

En résumé, cette méthode configure et affiche un fragment de type `DrawFragment` dans l'interface utilisateur de l'application.

showWaitFragment()
Affiche le fragment d'attente.

```
private void showWaitFragment() {
  curFragmentId = WAIT_FRAGMENT;
  showFragment(new WaitFragment());
}
```

showFragment(Fragment fragment)
Remplace le fragment actuel par le fragment spécifié.

```
private void showFragment(Fragment fragment) {
  FragmentTransaction trans = getFragmentManager().beginTransaction();
  trans.replace(R.id.rightLayout, fragment);
  trans.commit();
}
```

Gestion des événements

L'Activity gère diverses interactions utilisateur, telles que les clics sur les boutons et les sélections de menu.

onClick(View v)

Gère les événements de clic pour différentes vues.

```
@Override
public void onClick(View v) {
  int id = v.getId();
  if (id == R.id.drawOk) {
    if (drawView.isDrawFinish()) {
      saveBitmap();
    } else {
      Utils.alertDialog(this, R.string.please_draw_finish);
    }
  } else if (id == R.id.recogOk) {
    recogOk();
  } else if (id == R.id.recogNo) {
    recogNo();
  } else if (id == R.id.dir) {
    Utils.getGalleryPhoto(this, IMAGE_RESULT);
  } else if (id == R.id.clear) {
    clearEverything();
  } else if (id == R.id.undo) {
    drawView.undo();
  } else if (id == R.id.redo) {
    drawView.redo();
  } else if (id == R.id.camera) {
    Utils.takePhoto(cxt, CAMERA_RESULT);
  } else if (id == R.id.material) {
    goMaterial();
  } else if (id == R.id.upload) {
    com.lzw.commons.Utils.goActivity(cxt, PhotoActivity.class);
  } else if (id == R.id.scale) {
    cropIt(curPicUri);
```

```
  }
}
```

onActivityResult(int requestCode, int resultCode, Intent data)

Gère les résultats d'autres activités, comme la sélection ou le recadrage d'images.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
  if (resultCode != RESULT_CANCELED) {
    Uri uri;
    switch (requestCode) {
      case IMAGE_RESULT:
        if (data != null) {
          setImageByUri(data.getData());
        }
        break;
      case CAMERA_RESULT:
        setImageByUri(Utils.getCameraUri());
        break;
      case CROP_RESULT:
        uri = Uri.fromFile(new File(cropPath));
        setImageByUri(uri);
        break;
      case MATERIAL_RESULT:
        setImageByUri(data.getData());
    }
  }
}
```

Fonctionnalités d'annulation et de rétablissement

L'Activity offre des fonctionnalités d'annulation et de rétablissement pour les opérations de dessin.

initUndoRedoEnable()

Initialise les fonctionnalités d'annulation et de rétablissement en définissant des fonctions de rappel.

```
void initUndoRedoEnable() {
  drawView.history.setCallBack(new History.CallBack() {
    @Override
    public void onHistoryChanged() {
      setUndoRedoEnable();
      if (curFragmentId != DRAW_FRAGMENT) {
        showDrawFragment(curDrawMode);
      }
    }
  });
}
```

Note: Le code reste en anglais, car il s'agit d'un extrait de code Java et les noms de méthodes, variables, etc., ne sont généralement pas traduits pour des raisons de cohérence et de compréhension technique.

```
void setUndoRedoEnable() {
  redoView.setEnabled(drawView.history.canRedo());
  undoView.setEnabled(drawView.history.canUndo());
}
```

Personnalisation de DrawView

`DrawView` est une vue personnalisée utilisée pour gérer les opérations de dessin, les événements tactiles et le zoom.

onTouchEvent(MotionEvent event)

Gère les événements tactiles pour le dessin et le zoom.

```
@Override
public boolean onTouchEvent(MotionEvent event) {
  if (!scaleMode) {
    handleDrawTouchEvent(event);
```

```
  } else {
    handleScaleTouchEvent(event);
  }
  return true;
}

private void handleDrawTouchEvent(MotionEvent event) {
  int action = event.getAction();
  float x = event.getX();
  float y = event.getY();
  if (action == MotionEvent.ACTION_DOWN) {
    path.moveTo(x, y);
  } else if (action == MotionEvent.ACTION_MOVE) {
    path.quadTo(preX, preY, x, y);
  } else if (action == MotionEvent.ACTION_UP) {
    Matrix matrix1 = new Matrix();
    matrix.invert(matrix1);
    path.transform(matrix1);
    paint.setStrokeWidth(strokeWidth * 1.0f / totalRatio);
    history.saveToStack(path, paint);
    cacheCanvas.drawPath(path, paint);
    paint.setStrokeWidth(strokeWidth);
    path.reset();
  }
  setPrev(event);
  invalidate();
}
```

Ce code gère les événements tactiles pour dessiner sur une surface. Voici une explication en français :

- `action` : Récupère le type d'action de l'événement tactile (appui, déplacement, relâchement).
- `x` **et** `y` : Coordonnées du point touché.
- `MotionEvent.ACTION_DOWN` : Si l'utilisateur appuie sur l'écran, le chemin commence à ce point.

- `MotionEvent.ACTION_MOVE` : Si l'utilisateur déplace son doigt, une courbe quadratique est dessinée entre le point précédent et le point actuel.
- `MotionEvent.ACTION_UP` : Lorsque l'utilisateur relâche son doigt, le chemin est transformé en utilisant une matrice inverse, l'épaisseur du trait est ajustée, le chemin est sauvegardé dans l'historique, dessiné sur le canvas, puis réinitialisé.
- `setPrev(event)` : Met à jour les coordonnées précédentes.
- `invalidate()` : Force la mise à jour de la vue pour afficher les modifications.

Ce code est typiquement utilisé dans une application de dessin pour gérer les interactions tactiles et dessiner des formes sur un écran.

```
private void handleScaleTouchEvent(MotionEvent event) {
  switch (event.getActionMasked()) {
    case MotionEvent.ACTION_POINTER_DOWN:
      lastFingerDist = calFingerDistance(event);
      break;
    case MotionEvent.ACTION_MOVE:
      if (event.getPointerCount() == 1) {
        handleMove(event);
      } else if (event.getPointerCount() == 2) {
        handleZoom(event);
      }
      break;
    case MotionEvent.ACTION_UP:
    case MotionEvent.ACTION_POINTER_UP:
      lastMoveX = -1;
      lastMoveY = -1;
      break;
    default:
      break;
  }
}

private void handleMove(MotionEvent event) {
  float moveX = event.getX();
  float moveY = event.getY();
  if (lastMoveX == -1 && lastMoveY == -1) {
```

```
    lastMoveX = moveX;
    lastMoveY = moveY;
  }
  moveDistX = (int) (moveX - lastMoveX);
  moveDistY = (int) (moveY - lastMoveY);
  if (moveDistX + totalTranslateX > 0 || moveDistX + totalTranslateX + curBitmapWidth < width) {
    moveDistX = 0;
  }
  if (moveDistY + totalTranslateY > 0 || moveDistY + totalTranslateY + curBitmapHeight < height) {
    moveDistY = 0;
  }
  status = STATUS_MOVE;
  invalidate();
  lastMoveX = moveX;
  lastMoveY = moveY;
}
```

Ce code est une méthode Java qui gère le mouvement d'un événement tactile (`MotionEvent`). Voici une explication en français de ce qu'il fait :

1. **Récupération des coordonnées du mouvement** : Les coordonnées `moveX` et `moveY` sont extraites de l'événement `MotionEvent`.
2. **Initialisation des dernières positions** : Si `lastMoveX` et `lastMoveY` sont encore à leur valeur initiale (-1), elles sont mises à jour avec les coordonnées actuelles du mouvement.
3. **Calcul de la distance parcourue** : La distance parcourue en X (`moveDistX`) et en Y (`moveDistY`) est calculée en soustrayant les dernières positions enregistrées (`lastMoveX` et `lastMoveY`) des positions actuelles.
4. **Vérification des limites** : Si le mouvement dépasse les limites de l'écran ou de l'image (en tenant compte de la translation totale et de la taille de l'image), la distance parcourue est réinitialisée à 0 pour éviter de sortir des limites.
5. **Mise à jour du statut** : Le statut est mis à jour à `STATUS_MOVE` pour indiquer que l'utilisateur est en train de déplacer l'élément.
6. **Rafraîchissement de l'affichage** : La méthode `invalidate()` est appelée pour forcer la vue à se redessiner.

7. **Mise à jour des dernières positions** : Les dernières positions enregistrées (`lastMoveX` et `lastMoveY`) sont mises à jour avec les coordonnées actuelles du mouvement.

En résumé, cette méthode gère le déplacement d'un élément à l'écran en fonction des mouvements de l'utilisateur, tout en s'assurant que l'élément ne dépasse pas les limites de l'écran ou de l'image.

```
private void handleZoom(MotionEvent event) {
  float fingerDist = calFingerDistance(event);
  calFingerCenter(event);
  if (fingerDist > lastFingerDist) {
    status = STATUS_ZOOM_OUT;
  } else {
    status = STATUS_ZOOM_IN;
  }
  scaledRatio = fingerDist * 1.0f / lastFingerDist;
  totalRatio = totalRatio * scaledRatio;
  if (totalRatio < initRatio) {
    totalRatio = initRatio;
  } else if (totalRatio > initRatio * 4) {
    totalRatio = initRatio * 4;
  }
  lastFingerDist = fingerDist;
  invalidate();
}
```

onDraw(Canvas canvas)

Dessine l'état actuel de la vue.

```
@Override
protected void onDraw(Canvas canvas) {
  super.onDraw(canvas);
  if (scaleMode) {
    switch (status) {
      case STATUS_MOVE:
        move(canvas);
        break;
```

```
      case STATUS_ZOOM_IN:
      case STATUS_ZOOM_OUT:
        zoom(canvas);
        break;
      default:
        if (cacheBm != null) {
          canvas.drawBitmap(cacheBm, matrix, null);
          canvas.drawPath(path, paint);
        }
    }
  } else {
    if (cacheBm != null) {
      canvas.drawBitmap(cacheBm, matrix, null);
      canvas.drawPath(path, paint);
    }
  }
}
```

move(Canvas canvas)

Gère les opérations de déplacement pendant le zoom.

```
private void move(Canvas canvas) {
  matrix.reset();
  matrix.postScale(totalRatio, totalRatio);
  totalTranslateX = moveDistX + totalTranslateX;
  totalTranslateY = moveDistY + totalTranslateY;
  matrix.postTranslate(totalTranslateX, totalTranslateY);
  canvas.drawBitmap(cacheBm, matrix, null);
}
```

zoom(Canvas canvas)

Gère l'opération de zoom.

```
private void zoom(Canvas canvas) {
  matrix.reset();
  matrix.postScale(totalRatio, totalRatio);
  int scaledWidth = (int) (cacheBm.getWidth() * totalRatio);
```

```
  int scaledHeight = (int) (cacheBm.getHeight() * totalRatio);
  int translateX;
  int translateY;
  if (curBitmapWidth < width) {
    translateX = (width - scaledWidth) / 2;
  } else {
    translateX = (int) (centerPointX + (totalTranslateX - centerPointX) * scaledRatio);
    if (translateX > 0) {
      translateX = 0;
    } else if (scaledWidth + translateX < width) {
      translateX = width - scaledWidth;
    }
  }
  if (curBitmapHeight < height) {
    translateY = (height - scaledHeight) / 2;
  } else {
    translateY = (int) (centerPointY + (totalTranslateY - centerPointY) * scaledRatio);
    if (translateY > 0) {
      translateY = 0;
    } else if (scaledHeight + translateY < height) {
      translate

Y = height - scaledHeight;
    }
  }
  totalTranslateX = translateX;
  totalTranslateY = translateY;
  curBitmapWidth = scaledWidth;
  curBitmapHeight = scaledHeight;
  matrix.postTranslate(translateX, translateY);
  canvas.drawBitmap(cacheBm, matrix, null);
}
```

Gestion de l'historique

La classe `History` gère l'historique des dessins pour permettre les fonctionnalités d'annulation et de rétablissement.

saveToStack(Path path, Paint paint)

Sauvegarde le chemin actuel et le pinceau dans la pile.

```
public void saveToStack(Path path, Paint paint) {
  Draw draw = new Draw();
  draw.path = new Path(path);
  draw.paint = new Paint(paint);
  saveToStack(draw);
}
```

Remarque : Le code reste en anglais car il s'agit d'un extrait de code Java, et les noms de variables, méthodes et classes sont généralement conservés en anglais dans la programmation.

```
public void saveToStack(Draw draw) {
  curPos++;
  while (histroy.size() > curPos) {
    histroy.pop();
  }
  histroy.push(draw);
  if (callBack != null) {
    callBack.onHistoryChanged();
  }
}
```

getBitmapAtDraw(int n)

Retourne un bitmap représentant l'état à un point donné dans l'historique.

```
public Bitmap getBitmapAtDraw(int n) {
  Canvas canvas = new Canvas();
  Bitmap bm = Utils.getCopyBitmap(srcBitmap);
  canvas.setBitmap(bm);
  for (int i = 0; i <= n; i++) {
    Draw draw = histroy.get(i);
```

```
    canvas.drawPath(draw.path, draw.paint);
  }
  return bm;
}
```

undo()

Exécute l'opération d'annulation.

```
public Bitmap undo() throws UnsupportedOperationException {
  if (canUndo()) {
    curPos--;
    if (callBack != null) {
      callBack.onHistoryChanged();
    }
    return getBitmapAtDraw(curPos);
  } else {
    throw new UnsupportedOperationException("Aucun enregistrement à annuler");
  }
}
```

redo()

Exécute l'opération de rétablissement.

```
public Bitmap redo() throws UnsupportedOperationException {
  if (canRedo()) {
    curPos++;
    if (callBack != null) {
      callBack.onHistoryChanged();
    }
    return getBitmapAtDraw(curPos);
  } else {
    throw new UnsupportedOperationException("Aucun enregistrement à refaire");
  }
}
```

canUndo()

Vérifie s'il est possible d'annuler.

```
public boolean canUndo() {
  return curPos > 0;
}
```

canRedo()

Vérifie s'il est possible de refaire une action.

```
public boolean canRedo() {
  return curPos + 1 < histroy.size();
}
```

Conclusion

`DrawActivity` et ses classes associées fournissent un exemple complet pour implémenter une vue de dessin personnalisée dans Android. Il démontre diverses techniques, y compris la gestion des événements tactiles, la gestion de l'historique des dessins, ainsi que l'intégration avec d'autres composants tels que les fragments et les tâches asynchrones. En comprenant chaque composant et algorithme, vous pouvez exploiter ces techniques dans vos propres applications pour créer des fonctionnalités de dessin puissantes et interactives.