Uniting Twist
Developer Documentation
by
Sam Lincoln

Description

Uniting Twist is an Android game developed by Litun, and can be download from https://github.com/Litun/UnitingTwist. This game is about grouping colored hexagons into groups of three. These hexagons will be flying from the edges of the device to the center, and you must use the device's gyroscope to guide these hexagons into a position in the center. If the flying hexagon hits another hexagon, it gets stuck in a cluster. When a group of three is created, the hexagons get cleared from center cluster of hexagons. You get a point for each hexagon you clear from the cluster. Each ~~point~~ new flying hexagon causes the next flying hexagon to speed up. If the cluster gets too big, the game is over. You can try again to beat your high score after a game over.

Hardware and OS Specification

- Sensors: Gyroscope
- Minimum SDK: API Level 15, Android 4.0.3 (Ice Cream Sandwich)

Programming Languages

- Mostly Android Java
- Some classes written in Kotlin

Classes

ru.litun.unitingtwist (androidTest)

The only class here is the default ApplicationTest.java that Android Studio will create for you. The ApplicationTest class extends the deprecated ApplicationTestCase<Application> class. It has one method: public ApplicationTest(). That one method just calls super(Application.class). This ApplicationTest currently has nothing in it to actually test the application.

ru.litun.unitingtwist (test)

The only class here is the default ExampleUnitTest.java that Android Studio will create for you. The ExampleUnitTest has only one method in it: public void addition_isCorrect() that throws Exception. That one method makes sure that when the Android app adds 2 and 2 together, you end up with four. This is another class that doesn't really do anything for Uniting Twist as a whole.

ru.litun.unitingtwist

AngleListener Interface

This Interface states that all AngleListeners must have a function for void setUp(float x, float y). This method is overridden in the MainActivity class. The AngleListener is used in the GyroscopeListener class. It is given as a parameter in GyroscopeListener's constructor, and is used inside onSenorChanged(SensorEvent) to send the cosine and sine of the latest Z axis change through setUp(cosine, sine). The Z axis changes when the device is turned like a steering wheel.

Circle

This class uses the functions of OpenGL ES to create drawable circles. It contains its own shader codes for vertexes and fragments (vertextShaderCode and fragmentShaderCode Strings). Its constructor, Circle(), sets up the vertexes that would make up the circle of the device's screen. The same constructor loads all the vertexes into a FloatBuffer, loads the shader codes, then creates an OpenGL ES program and adds the created shaders to the program. The shader codes are loaded through Circle's loadShader(int type, String shaderCode). The loadShader method use OpenGL ES to create the shader of the type, compile the shader code to make the shader, and then returns the new shader (its int Id for that shader). The final method, public void draw(float[] mvpMatrix) is what is called to get OpenGL ES to actually draw the circle to the device's screen.

ColorUtils

This class is used to set up the colors that will be used for the hexagons. Public static void init(Context) is first called from ColorUtils to get the hexadecimal value of the colors from /app/res/values/colors.xml, turns those into ints, sets up a float[4] array for each color for the Red Green Blue Alpha values for each color, uses a loop to turn those into floats that the device uses for colors, and then puts them into a float[][] for later use. Public static float[] getColor(int n) returns that float array for the color in the index of n modulus colors.length (length of the float[][] array from init). Public static int colorsCount() just returns colors.length.

Drawable Interface

This interface makes sure that every class that implements Drawable has a void update(long) and a void draw(float[] mvpMatrix). It is implemented in FieldGraph to draw the hexagon cluster, GameField to draw the center and flying hexagons, GameHexagon to draw the actual shape of each hexagon, and the GameHexagonContainer to just call the GameHexagon's draw method.

Engine

This class has an inner class called LoopThread that extends Thread. LoopThread's overridden public void run() function will run a loop that continues to run until interrupted. The loop's job is to make sure enough time has passed for MIN_DELTA_TIME, and then updates the screen by running the Engine class's update() command. The LoopThread class also uses public void updated() to set date to

the recent time, public void setRunning(boolean running) to store that value in its private running variable, and public void preInterrupt() to set its private boolean interrupted to true. Engine is constructed from a given Scene and GLSurfaceView to set its scene and surface private variables. Engine's update() function calls scene.update() to update the positions of the hexagons and then calls surface.requestRender() to draw the hexagons on the device's screen.

FieldGraph

The FieldGraph class handles the adding and removal of hexagons on the screen. The FieldGraph(int n) constructor uses int n as the number of hexagons can fit in the radius of the center area. The constructor doubles n and creates points for each hexagon that can fit in the center area, and then fills a 2d GraphGameHexagon array with GraphGameHexagons. We got getters for List<GraphGameHexagon> getEndpoints() (hexagons where flying hexagons can collide into) and getOpened() (spots where flying hexagons can be put in to for the cluster). A getter for the center GraphGameHexagon getCenter(). Public void put(GraphGameHexagon hexagon) will check for any that GraphGameHexagons by hexagon, and including hexagon, need to be removed. It also finds the amount to add to score and notifies the GameListener. If the method finds out that the latest hexagon goes outside the bounds, the GameListener is also notified that the game is over. Void removeCluster(List<GraphGameHexagon> list) removes the hexagons from the cluster. Public void totalRecount() goes through all the hexagons by the center one, and makes sure that all the open spots are opened and all the endpoints are set correctly. Void resetVisits() makes sure none of the GraphGameHexagons are marked visited, they get marked visited as they are scanned by put(GraphGameHexagon). Public void rotate(float angle) rotates all the GraphGameHexagons' Point by angle so that they are drawn properly. Public void setGameListener(GameListener listener) sets FieldGraph's listener to the given listener. Overridden public void draw(final float mvpMatrix) sends mvpMatrix to all the GraphGameHexagons' draw function to put them all on the screen. Private void iteratePoints(Action action) goes through all the GraphGameHexagons in the 2d array called points. Private void iterateNear(GraphGameHexagon hexagon, Action action) goes through all the GraphGameHexagons by hexagon. Private void iterateNearHexagons(GraphGameHexagon hexagon, final Action action) adds an action to Private void iterateNear(GraphGameHexagon hexagon, Action action) that makes sure that hexagon's hasHexagon() is true before calling action.act(hexagon). This class also has interface Action which makes sure that any Action must have an overridden method void act(GraphGameHexagon h).

FlyingGameHexagon

This class extends GameHexagonContainer, and it adds things related to movement and positioning. Public FlyingGameHexagon(GameHexagon h) sets the GameHexagonContainer's hexagon to h. Overridden public void update(long deltaTime) will move hexagon's position according to a float value derived from deltaTime. Public void setVector(float x, float y) will put the value of x into private float vectorX and y into private float vectorY.

GameField

This class implements Drawable, and handles generating new FlyingGameHexagons and placing those FlyingGameHexagons into open spaces in FieldGraph when they collide into the cluster. The Public GameField() sets up the GraphGameHexagon at the center of FieldGraph. The overridden public void update(long deltaTime) changes the position for each FlyingGameHexagon according to deltaTime and then checks for collision through collisionDetect(). The overridden public void draw(float[] mvpMatrix) calls FieldGraph's draw method using the mvpMatrix, and then it grabs the GameHexagon from each FlyingGameHexagons from its final list and calls the draw method using mvpMatrix for GameHexagon. Private TimerTask newGenerator() generates a random angle, sets that random angle to the vectors of a new FlyingGameHexagon, gives it a velocity, and then adds this new FlyingGameHexagon to GameField's final list. The velocity does not increase for each point, but it starts out at 0.1 and increases by 0.005 every time a FlyingGameHexagon is added to the list. Public void startGenerating() creates a new Timer and schedules newGenerator() to be executed every three seconds. Public void stopGenerating() cancels the timer and performing newGenerator() if startGenerating() was called before calling this method. Private void collisionDetect() will go through every endpoint in the FieldGraph and checks each FlyingGameHexagon in the list to see if their point distance is less than 0.1, if it is, then it calls findOpen on that FlyingGameHexagon. Private void findOpen(FlyingGameHexagon flyingHexagon) searches for a suitable open location to store the flyingHexagon, sets the open location hexagon to the GameHexagon of flyingHexagon, and then the flyingHexagon is removed from GameField's list.

GameHexagon

This class implements Drawable, and it just holds the properties of any hexagon on the screen: the Point position, the angle of rotation, and color. Public GameHexagon(Point p) creates a new GameHexagon with the Point point set to p. The overridden public void draw(float[] mvpMatrix) grabs an instance of a Hexagon using Hexagon.getInstance(), rotates it to GameHexagon's angle, translates it to GameHexagon's point x and y, sets its color to GameHexagon's color, and then calls that instance's draw using mvpMatrix. Public void setAngle(float angle) sets the GameHexagon's angle to the one in the parameter. Public void newPoint(Point p) sets GameHexagon's point to p. Public void setColor(int color) sets the GameHexagon's color to the one in the parameter; this is an index to a color stored in ColorUtils. Public void move(float x, float y) will increase GameHexagon's point values x and y by the amount in the parameters. Public Point getPoint() returns GameHexagon's point. Public int getColor() returns GameHexagon's color; an index to a color stored in ColorUtils.

GameHexagonContainer

This class implements Drawable, and provides abstraction to a protected GameHexagon. Public GameHexagon getHexagon() returns the protected GameHexagon. Public void setHexagon(GameHexagon hexagon) sets the protected GameHexagon to the one in the parameter. Public Point getPoint() returns the protected GameHexagon's point. The overridden public void

draw(float[] mvpMatrix) calls the protected GameHexagon's draw method using mvpMatrix if the protected GameHexagon is not null.

GameListener Interface

This interface makes sure that any class that implements it contains overridden methods: void onCut(int n) and void onLose(). These methods are overridden in MainActivity.kt.

GraphGameHexagon

This class extends GameHexagonContainer and implements a Comparable for GraphGameHexagon, and this class is used to add features used in searching and managing the hexagons in the cluster. Public GraphGameHexagon(int I, int j, Point p) is used to create a GraphGameHexagon with its I and j values set to the given parameters and sets its graphPoint to p. The overridden public void setHexagon(GameHexagon hexagon) calls GameHexagonContainer's setHexagon(hexagon) and then sets that hexagon's Point to graphpoint. Public float distance(Point other) returns the result of the distance formula where this GraphGameHexagon's graphPoint is the first point and parameter other is the second point. Public Point getPoint() returns graphPoint. Public boolean isVisited() returns visited (which starts as false, but can be marked so by methods in GameField). Public void setVisited(boolean visited) sets the value of GraphGameHexagon's visited to the value in the parameter. Public boolean hasHexagon() returns true if GameHexagonContainer's getHexagon() returns a non-null Hexagon, else it is false. Public int getI() returns I, and Public in getJ() returns j. The overridden public int compareTo(@NonNull GraphGameHexagon other) comes from Comparable, and it compares the I and j values of the two GraphGameHexagon to find where it stands in an 2d array of GameGraphHexagons in the FieldGraph class. Public void removeHexagon() sets the GameHexagon from GameHexagonContainer to null.

GyroscopeListener

This class implements SensorEventListener and it is responsible for handling the SensorEvent for changing Gyroscope; something that is triggered whenever the device's title changes on any of the axises. Public GyroscopeListener(AngleListener l) creates a new GyroscopeListener that sets its AngleListener listener to that of l. The overridden public void onAccuracyChanged(Sensor sensor, int accuracy) does nothing, but is there to implement the SensorEventListener. The overridden public void onSensorChanged(SensorEvent event) acts when the event deals with the gyroscope, and then it compares the date that this event happened with the previous date that this event took place to find the angle it should use to get sin and cos values. Once it has those values, it passes those to AngleListener using listener.setUp((float) cos, (float) sin).

Hexagon

This class is responsible for telling OpenGL ES how to draw the hexagon shape. Private static Hexagon init() sets up this class's private static Hexagon instance to a new Hexagon and then returns instance. Public static Hexagon getInstance() returns instance. Private final String vertexShaderCode is the shader code for the vertexes of a Hexagon for OpenGL ES to use. Private final String fragmentShaderCode to set up how a Hexagon's color is used in OpenGL ES's shaders. Static final int COORDS_PER_VERTEX sets the number of coordinates for each vertex of a Hexagon, and static float hexagonCoords[] sets all the float coordinates for a complete Hexagon. Private final short drawOrder[] sets the order of vertexes to be drawn. Private final int vertexStride states how many Bytes a Hexagon will use. Public Hexagon() sets up the drawing object data to be used in OpenGL ES: it initializes the vertex Byte buffer for shape coordinates, another buffer for the draw list, and then prepares the shaders and finishes the OpenGL ES program to draw a Hexagon. Public void initVertexBuffer() is called in the middle of public Hexagon(), but it clones a master coordinate array and shrinks each coordinate to a scale. Once initVertexBuffer() scales the coordinates, it then takes the ByteArray set up in public Hexagon and creates a FloatBuffer vertexBuffer filled with the scaled coordinates. Public void draw(float[] mvpMatrix) loads the ID of the Hexagon's OpenGL ES program, stored in private final int mProgram after calling public Hexagon(), into OpenGL ES's environment for rendering. Public void draw(float[] mvpMatrix) will set the right angle to start the drawing, load mProgram, prepare the vertexes of the Hexagon, setup up the Hexagon's color, setup the view transformation matrix through uMVPMatrix, applies the transformation, and then tells OpenGL ES to draw that stuff as well as disable the vertex array. Public void translate(float x, float y) sets the Hexagon's float x and float y to the values from the parameters; this move the position that the Hexagon should be located. Public void rotate(float angle) sets the Hexagon's rotation angle (float) to the one given in the parameter. Public void setColor(int colorInd) sets the Hexagon's float array for color to the one in ColorUtils at the index of colorInd.

MainActivity

This class was written in Kotlin, extends AppCompatActivity, implements AngleListener, and is the actual Android Activity where you play the game, but not the Activity that is first shown when you start the app. This class grabs information from the gyroscope sensor and sets up the GyroscopeListener and the OpenGL ES Engine. The overridden public void onCreate(Bundle savedInstanceState) sets the layout for the screen to the one in /app/res/layout/activity_main.xml, creates an OpenGL ES context for the screen, and overrides the functions of GameListener and puts it on the GameField field to play the game: public void onCut(int n) adds n to the score which is used when a group of Hexagons is removed from the cluster and public void onLose() calls MainActivity's lose(). Public void onStart() calls AppCompatActivity's onStart() and Engine engine's create(), which starts up a loop that wakes up after a minimum delta time to redraw the objects on the screen; this is to start generating the Hexagons for the game. The overridden public void onResume() calls AppCompatActivity's onResume, calls the same method for the surface, and re-registers the GyroscopeListener and calls Engine engine's resume(); assuming that onPause() was called before

onResume(). Also, during onResume(), it also sets the flags to keep the screen on and the flags for a full screen app where navigation and UI are hidden. This stuff with the onResume() method is so the game can start generating Hexagon's again after an onPause(). The overridden public void onPause() calls both AppCompatActivity's onPause() and surface's onPause(), unregisters the GyroscopeListener, removes the flag to keep the screen on, and call's Engine engine's pause(); this will pause the game and stop generating Hexagons. The overridden public void setUp(float x, float y) calls MyGLRenderer renderer's setUp(x, y) function, GameField field's newUp(x, y) and Engine engine's forceUpdate() all to setup the screen to the new x and y positions and redraw all the object on it. The overridden public void onStop() calls AppCompatActivity's onStop() and Engine engine's destory() to completely stop generating Hexagons by removing the infinite loop thread in Engine; the activity will have to be re-created to set up the Engine again. Public void lose() sets up an intent with the score and result code of RESULT_OK, to finish this activity with a result for StartActivity. The result from MainActivity will be handled by StartActivity's onActivityResult(int requestCode, int resultCode, Intent data).

MyGLRenderer

This class implements a GLSurfaceView.Renderer and is used to setup OpenGL ES to draw everything on the screen. Public MyGLRenderer(Scene scene) creates a new MyGLRenderer with it's Scene scene being set to the one in the parameters. The overridden public void onSurfaceCreated(GL10 unused, EGLConfig config) sets the background frame color in OpenGL ES, and then it initializes Hexagon and creates a circle for the MainActivity screen. The overridden public void onDrawFrame(GL10 unused) draws the background on the screen, calculates how to transform all the Hexagons so that they look correct at the current angle, draws the circle and the scene, and then calls Matrix's setRotateM and multipleMM to apply these transformations in OpenGL ES. The overridden public void onSurfaceChanged(GL10 unused, int width, int height) adjusts the viewport in OpenGL ES to match the screen rotation by recreating the viewport and then calls Matrix's frustumM to change the object coordinates on the screen. Public void setUp(float x, float y) sets the MyGLRenderer's upX to x and upY to y, and MyGLRenderer's mAngle to the Arc Tangent of the two in degrees. Public static int loadShader(int type, String shaderCode) is a utility method for compiling OpenGL shader; which, is done by using GLES20's glCreateShader to create a blank shader, then loads the shaderCode into the blank shader, and then compiles and returns the ID tag for that shader. Public static void checkGlError(String glOperation) is a utility method used to debug OpenGL calls and it checks if the previous call had an error and shares the same name as glOperation.

Point

This class was written in Kotlin and is called a data class. That means that Point does nothing except hold data. Point just holds the x, y, and z coordinates used for Hexagons. The x and y coordinates are used, but z is not since this is a 2d game. All three variables are floats.

Scene

This class is used to handle grabbing the current date, calculating the time delta, and then sending that off to its GameField field. Public Scene(final GameField field) creates a new Scene and sets its field to the one in the parameter. Public void draw(float[] mvpMatrix) calls GameField's draw(mvpMatrix) to draw the Hexagons. Void update() will grab the current date, calculate time delta, updates Scene's Date date to the current date, and then sends the time delta to GameField field through update(delta) if Scene's date is not null. Void resume() will set the Date date to a new Date and will tell GameField field to start generating FlyingHexagons again, assuming void pause() was called before resume(). Void resume() sets Date date to null and tells GameField field to stop generating FlyingHexagons.

StartActivity

This class is written in Kotlin, extends AppCompatActivity, and is the class that sets up the first screen that you see when you start the app. It reads the best score from a shared preference called prefs. The overridden public void onCreate(Bundle savedInstanceState) sets the layout to that stored in /app/res/layout/activity_start.xml, adds startGame() to the onClickListener for the white Hexagon in the center, and then it grabs best (the best score) from prefs and calls setText(-1, best) to display it at the bottom, but does not show Score since the User hasn't played a round yet. If best is not found in prefs, then -1 is used instead and nothing is displayed at the bottom. Public void startGame() creates an intent to start MainActivity for a result (requestCode 3) and then uses the transition created from /app/res/anim/fade_in and /app/res/anim/fade_out; this will bring us to the screen where we actually play the game. The overridden public void onActivityResult(int requestCode, int resultCode, Intent data) is assumed to be called after losing a round in MainActivity. It checks to make sure the requestCode is 3 (set in startGame()) and the resultCode is RESULT_OK before extracting the score from data. If the retrieved score is higher than best, then best is updated both as a variable and in the shared preference. Then setText(score, best) is called to display the new score and best values at the bottom of the screen. Public void setText(int score, int best) sets the TextView at the bottom of the layout to display last rounds score and the best score; if -1 is sent as either parameter, that will disable the display of the last round score or best score respectively.

TwistApp

This class extends Application, and it is called when the app first starts up to set up a few things before arriving at StartActivity's screen. The overridden public void onCreate() calls Application's onCreate and ColorUtils.init(this) to set up the colors stored in ColorUtils when the app starts and before MainActivity's screen.