CSC 122: Gravity Simulator

Advanced User Interface

1 Introduction

In this lab, you will create a gravity simulator. The simulator will respond to user touch events in order to create planets of varying size and exude limited control over their motion. You will accompish this by designing a GUI layout, handling touch events initiated by the user, and rendering the planets. A custom GUI element will be provided to aid the touch event handling and animation rendering.

2 Objective

The purpose of this lab is to present the student with an advanced user interface experience including complete control of the GUI layout and interactive functionality. The student will learn how to handle touch events and render simple animations on an Android system.

3 Activity

This lab will be fairly freeform in execution due to the large control you have over the implementation. A theory section is provided to aid in your design of the project solution. It is recommended that you look over the theory section at least once before beginning to code. Details for the implementation can be found in their own section after Theory.

3.1 Theory

This section provides background information for various parts of the lab. Refer to this section as needed during your implementation.

3.1.1 Kinematics

This section will include formulas and brief introductions for gravitational interaction, motion under constant acceleration, and elastic and inelastic collisions. The student will have the option to modify formulas to alter the physics, though there are some restrictions: the gravitational force must be attractive and proportional to mass, and at least one (elastic or inelastic) collision must be modeled.

3.1.2 Touch Events

An explanation of touch events will be given with highlights of their usage and handling, such as how to detect the beginning, middle, and end of motions. The student will also be directed toward the `VelocityTracker` class for approximating the velocity of the user's touch gesture.

3.1.3 Rendering Animation

A brief description of the classes and methods necessary for animation will be given. Images for the background and planets will be provided with instructions for how the student may replace them if desired.

3.2 Implementation

This section will provide a general guide to getting your app running.

3.2.1 Research

If you have not encountered enumerated types or the `final` keyword before, look up their usage in Java. In addition, review their usage in the code provided, specifically inside of the class `GravCanvas` and its inner class `GravThread`.

3.2.2 Design the layout

You may use the XML Layout editor in Eclipse to aid you in designing the GUI. The layout must provide an ample amount of space for a `GravCanvas SurfaceView`. In addition, it must provide buttons or some other device to control pausing/playing of the simulator and locking/unlocking of the screen for touch events.

3.2.3 Make an Activity

Make an activity to hold references to the `GravCanvas`, its `GravThread`, and buttons. This activity will handle button clicks and their results as they pertain to the `GravCanvas`. Button labels should be kept up to date with their current function if any are multipurpose ("Play" or "Pause"). Override `onDestroy()` such that it contains a call to `GravThread.setRunning(false).`

3.2.4 Implement Kinematics

Decide which physical laws your program will simulate and give an initial value to `GravThread.GRAVITATIONAL_CONST`. Inside of class `Planetoid`, add and modify methods for the planet to react to stimuli according to your physical laws. Minimum recommended methods include force application and movement for given durations and set methods for momentum and area.

3.2.5 Model Gravitational Interaction

Tasks contained in this section should only occur if animation is set to play. Inside of the function `GravThread.gravitate()`, apply forces based upon the gravitational field each planet exerts upon the others . These forces should be applied for a certain duration equal to the time elapsed since the last time `gravitate` was called. After all the forces have been applied to a planet, move it based upon one or more of the formulas given for motion under constant acceleration. If any planets collide (if they would overlap eachother when drawn), either apply an elastic (they bounce off eachother) or inelastic (they merge into one) collision to each. You are in control of which of the two types of collision takes effect. If a planet would move off of the screen, you may either let it continue or make it bounce off the edge at an appropriate angle (alter the direction of its velocity while retaining the magnitude). Planets that continue off the edge of the screen may very well return if the gravitational force is strong enough.

3.2.6 Handle Touch Events

Tasks contained in this section should only occur if the screen is unlocked. In `GravCanvas.onTouchEvent()`, handle touch events initiated by the user. You will need to interact with `GravCanvas.mThread` in order to affect the animation. If one of the motion pointers is positioned over empty space when it begins, create a `Planetoid` at that location and associate it with the pointer. Otherwise, associate the existing `Planetoid` at that location with the pointer. In addition, the `Planetoid` should be *selected* such that it obeys the following constraints: its position is the same as its associated motion pointer, its image is different from unselected `Planetoids`, and its size and mass slowly increase over time. Note, however, that the size and mass should only increase if the `Planetoid` was newly generated, i.e. its associated motion pointer started over empty space. When a motion pointer ends, the `Planetoid` is deselected and continues along the trajectory set by the motion pointer's last estimated velocity. The speed of a `Planetoid` released this way should be inversely proportional to its mass. Flinging of existing `Planetoids` may optionally be allowed on locked screens (but not `Planetoid` generation).

3.2.7 Debug

Try running the app, performing both positive and negative tests. For example, do any `Planetoids` stay selected after a motion event ends or simultaneous finger movements end? In addition, does it perform to your desire? You may wish to tweak the strength of flinging or your gravitational constant if the planets move too slowly or quickly for your liking. Experiment until you find a balance that you enjoy.

3.2.8 Optional Refinements

If you would like to save your progress (at least for as long as the app is running either on the screen or in the background), then you should do the following in your activity. Override `onPause()` and `onResume()`. Inside of `onPause()`, call `GravThread.quit()` with a `static Bundle`. Inside of `onResume()`, call `GravThread.restart()` with the same `Bundle`. Look inside these functions to discover how your buttons should be labeled.

4 Conclusion

Upon completion of this lab, you should be capable of designing Android applications that can handle touch events. In addition, you should have gained a greater understanding of how to render images and animations.

5 Deliverables

Deliverables include a project export from Eclipse. When running, the simulator should create, move, and grow planets according to user touch events only if the screen is unlocked. Planets should only move under their own influence when the simulator is playing. Pausing should not prevent touch events, and upon resuming behavior should be as though the pause never occurred.