CS 301 **Project Assignment 6** Tuesday, November 7, 2017

To Deliver: 1. Release 6.0 in your SVN repository

To Download: —
Hard & final submission deadline: November 20, at midnight (11.59 PM)

Learning Objectives:
- Experience in designing a user interface.
- Learn to work with a UI design in XML and connecting it with Java code
- Experience in software development on an Android platform: working with activities and intents

Motivation/Background information

We want to port our maze java application to Android. This requires us to redesign the user interface for a start.

User interface design is very important for a successful app. It needs to be intuitive and self explanatory as you cannot expect a user being willing to ever read a manual page on how to use your wonderful software.

Design decision

It is up to you to decide on the particular design, however as **minimum requirements** we will follow the 4-stage automaton implemented in Falstad's maze program:

State Title (class name: AMazeActivity):
1. Displays the welcome page and takes parameter settings to start with the maze generation.
2. Your code should include options to set the following parameters:
   2.1. the complexity (size) of the maze. Consider using a SeekBar for this. Pick a meaningful, commonly used default value (0 may be a good choice during development).
   2.2. the selection of either generating a new maze or loading a maze from file. One possibility is to have two buttons "Revisit" and "Explore".
      2.2.1. if user clicks "Explore", take into account what is selected for the maze generation algorithm (Backtracking, Prim's, and Eller's algorithm). Consider using a Spinner for this to show the selection of generation algorithms, pick a commonly used default value.
      2.2.2. if user clicks "Revisit", loads the last maze that matches with the selected complexity (size) from file to play it again.
         2.2.2.1. Note: You do not have to implement file access for this project. File access is a bit tricky, so let's keep this simple for a start: the plan is to store only 3 files, one for each skill level 0, 1, and 2. For a maze of size 0, 1, or 2: when you generate a new maze, then store it (possibly replacing an existing file) before the user plays the game.
   2.3. the selection of one of the following ways to operate the robot: a) Manual, b) Wall Follower, c) Wizard, d) Pledge, e) Explorer (optional). Pick manual as a default value.
3. Navigation:

    3.1. Possible transitions are to state Generating (user clicks on the start button after selecting parameter values).
4. Hint: check slides for lecture “Android User Interface Basics” for an example. Please note that this example is not the one possibility, feel free to do your own design.

State Generating (class name GeneratingActivity):
1. Intermediate page that shows <u>progress</u> of the maze generation algorithm and informs a user. Once the generation algorithm is finished the display switches to State Play.
2. Navigation
    2.1. Pressing the back button will stop the maze generation and return to State Title.

State Play (class name PlayActivity):
1. Displays the maze and lets the user either watch a robot exploring the maze or allows the user to manually navigate the robot through the maze.
2. Detailed features
    2.1. Displays the remaining energy, consider using a ProgressBar for this.
    2.2. Provides a feature to toggle visibility of the map plus functionality to toggle visibility of the solution on the map. Support the following choices:
        2.2.1. show the whole maze from top or not (toggle).
        2.2.2. show the solution in the maze or not (toggle).
        2.2.3. show the currently visible walls or not (toggle)
        2.2.4. Note: this matches existing functionality in the maze: keyboard input m,s,z.
    2.3. If in manual exploration mode: screen provides navigation buttons (up, down, left, right).
    2.4. If in robot exploration mode: screen provides a start/pause button to start the exploration and to pause the animation.
3. Navigation
    3.1. If the robot stops (no energy, at exit) the screen switches to the finish screen.
    3.2. Pressing the back button returns to State Title to allow the user to choose different parameter settings and restart.
    3.3. As an intermediate step only for this project introduce a Button "Shortcut" to directly move the UI to State Finish (possibly enhanced with a selection mechanism for: success, failure due to lack of energy, i.e. you may use 2 buttons).

State Finish (class name FinishActivity):
1. Displays the finish page and informs the user what happened and how to restart the game.
2. Shows the overall energy consumption and length of path.
3. Visualizes if robot stopped for lack of energy, or if it is broken, or if it reached the exit (consider using <u>audio</u> or <u>vibrations</u> to enhance the user experience).
4. Navigation
    4.1. Pressing the back button returns to State Title.

As we do not use the maze code at this point, you need to imitate the behavior of the final app at certain points:
- In general: have your user interface respond to any user input by a corresponding message (with a so-called toast or snackbar) **and** a Log.v() output into the Eclipse Logcat window such that you can recognize that the bits and pieces of your app correctly communicate. While the toast or snackbar messaging is just to try this out, the Log.v output is something you need to retain and use for P7.

- State Generating: imitate the delay in this state and the update to the progress visualization by some simple code that just shows progress but does not generate a maze.
- State Play: you should represent the graphics part for the maze in your layout with 2 simple buttons (the Shortcut buttons, one for winning, one for loosing) that when clicked, the display moves on to state Finish. Also prepare for a user interface in state Play that allows a user to navigate the game. Check if buttons for right, left, top, bottom are pressed. You can also decide to have two activities for the playing state: one for the animation with the driver algorithm and one for the manual operation.

To retain a reasonably uniform design, I would like to fix some further naming conventions for classes:

- Call your new Android project **AMazeBy<YourName>** and
- use a path **edu.wm.cs.cs301.<yourname>** and
- create three packages:
    - **ui** for user interface and
    - **falstad** and **generation** for the code that we will incorporate at a later point in time.

Hint: good starting point is: https://developer.android.com/training/basics/firstapp/index.html

Task list:

1. **Prerequisite**: Install the Android Studio and the Android SDK on your machine, create an AVD for a Nexus 4 API level 23 or higher as a reference architecture.
    1. Install Android Studio, follow the installation instructions on developer.android.com or watch a lynda.com video for it (see CS301 course website on blackboard). For HAXM: install the installer with the SDK manager but also RUN the installer to actually have HAXM installed (common pitfall, at least in previous years).
    2. Create an AVD for a Nexus 4 API level 23 or higher as a reference architecture. Choose an Intel processor to be able to benefit from the accelerator.
2. **UI Design**
    1. Do a paper prototype of the design
        1. Draw each individual screen, button etc on paper. Simulate common usage scenarios to convince yourself that the design works and is convenient & appealing.
    2. Perform a class design with CRC cards for the different Android activities in your user interface.
        1. Pick one activity per state and use the class names as suggested above, e.g. AMazeActivity.java for the title screen. Make sure you recognize collaborators from the existing Maze project with whom your new classes will interact with.
        2. Put your crc design in writing, simply list each class, its responsibilities and collaborators. For collaborators on the existing Maze project, denote what changes will be necessary there. Create a report named P6.pdf and include this in your svn repository release.
3. **Getting started with Android Studio**
    1. Exercise the tutorial on ConstraintLayouts from https://codelabs.developers.google.com/codelabs/constraint-layout/index.html?index=..%2F..%2Findex#0 and go through the constraint-layout-start example project.
    2. Create a new Android project in Android Studio for the AMaze app.
        1. Any new project will be created as a fully functioning hello world implementation.

        2. Follow the Getting Started Guide on developer.android.com with sections "Building your first app" and "Adding the action bar".

4. **Getting started with Android programming**
    1. To get more familiar with Android projects, there are different ways to go from here.
        1. You can follow the Getting Started Guide on developer.android.com with sections "Building your first app" and "Adding the action bar".
        2. Other alternative tutorials are for instance this one developing a simple UI and starting another activity or this one by Lars Vogel.
        3. With the a HelloWorld that actually works, purposely add code that is faulty to see what error messages this creates. Try some syntax errors. Try calling a method on a variable with value null to see how a null pointer runtime exception is reported and so forth. This makes it easier to recognize what is wrong if your code has errors.
5. **Implementation**
    1. Start with your freshly created AMaze project that comes as a Hello World implementation.
    2. Define each activity:
        1. Follow the naming conventions for classes and UI elements (like buttons) listed under "Design Decisions" above.
    3. Some hints:
        1. Check on the use of the action bar: design concepts and how to work with the action bar.
        2. Check tutorials on how to add more activities and how to make a switch from one activity to another with an intent and parameter settings (e.g. described here or here).
        3. Serialization: A tutorial that particularly deals with the communication of data from serializable objects is this. Serialization in a general Java setting is also discussed in Horstmann's book, Chapter 7.
        4. Add a corresponding layout xml file and specify its content in directory res/layout. For the first activity you can refine the pregenerated file main.xml there but for others you need to add a new file for each new activity.
        5. All strings that you use need to be defined in file res/values/strings.xml.
        6. Basic widgets are for instance described here.
    4. Update file AndroidManifest.xml to take the new activities into account.
    5. Add a corresponding java file to src/edu.wm.cs.cs301/ui/ for each activity. Each activity will need its own java class file there.
    6. Add navigation between activities by using intents such that perform the navigation as described under "Design Decisions" above.
6. SVN
    1. Share your project with your SVN repository.
    2. Update your subversion repository whenever you make progress and got a particular feature or method going.
        1. There is no need to create 10 revisions, just use SVN as you find it helpful.
    3. When you have finished the project assignment, create a tag with a Release 6.0 before the due date.
    4. Add the CRC design description to the repository under reports/P6.pdf.

Aside: Porting the falstad and generation package and making it work with the new Android user interface is the topic of the next and final project P7.

5. **Optional for a head start for P7**: If for some reason you want to go ahead, please note that Java AWT and Swing packages are not supported on Android. These need to be replaced. So let's put the graphics issues aside for now. A good start without the graphics is to copy the generation package over to Android and make the GeneratingActivity work with a background thread to create the necessary maze information (the MazeConfiguration). Make the MazeConfiguration accessible via a static variable such that the GeneratingActivity can access it as well as the PlayActivity. One of the challenges will be to make the background thread feed data into the progress bar.

Grading:
You will receive points for:

1) Your SVN repository:
    1) your subversion repository is set up correctly and the grader can download your release6.0 or higher from its tags folder.
2) Your design:
    1) You will receive points for your CRC class design (file: reports/P6.pdf).
3) Your implementation:
    1) **Activities:** You will receive points for implementing each of the 4 activities such that the Android Studio project does not show any syntactical or configuration errors. Code is expected to have comments, at least a statement per method (using the javadoc format with /** …. */).
    2) You will receive points for each activity that the grader is able to run on the emulator and by moving from the starting main activity to that activity.
    3) **Feedback:** You will receive points if your implementation provides feedback with the help of Log.v() message in the Logcat window to demonstrate that it correctly received user input AND if messages popup either as toasts or snackbar messages.
    4) **Navigation:** You will receive points for a working navigation between screens, i.e.,
        1) a user can start with the title, click a button to move to the generating screen.
        2) The generating screen switches to the playing screen once the progress bar hits a 100% (or there is a button to switch).
        3) The playing screen switches to a final screen with a button click (the placeholder for the missing graphics part).
        4) Plus the back button navigation gets you back from various screens as described in the text above.