CS 301 **Project Assignment 7** November 19, 2017

To Deliver: 1. Release 7.0 in your SVN repository

To Download: —
Submission deadline: Monday Dec 4, at midnight (11.59 PM)

Learning Objectives:
- Porting code across platforms
- Benefits of encapsulation, (targeted reimplementation of MazePanel)
- Experience in the differences between the general Java environment and the specific Android Java environment.
- Working with threads (the maze generation operates in a separate thread to keep the UI responsive)
- Working with simple graphics in Android
- How Android handles graphics and screen updates in its UI thread with a message queue

Topic: Android App for the Maze game

Motivation/Background information

We want to port our maze java application to Android. This requires us to redesign the user interface and to port the Falstad and generation packages into the new Android project.
Preparation done so far:
- Project 6: new Android UI design. Your newly created Android UI provides you with screens and navigation between them. This is a great start. Since the new UI from Project 6 handles the different screens with activities, we do not need certain classes from the Falstad package anymore, for example MazeApplication.java is not used anymore. We also need to reorganize MazeController.java since the different screens and states that it currently manages are now handled by the new UI activities. In consequence, MazeController.java will loose some of its responsibilities.

1st Challenge in Porting the Maze code base to Android

The current Falstad package uses AWT and Swing packages for graphics. Of course Android supports similar drawing capabilities as AWT and Swing, so we basically need to reimplement our MazePanel class. In doing so, there are different choices, one is that of a View. The following description focuses on that solution but you can also pick a different strategy.

Suggested strategy:
1. Reference http://developer.android.com/guide/topics/graphics/2d-graphics.html).
2. If you choose to work with a View, then
   1. Make your MazePanel a subclass of View and integrate it into the layout for your PlayActivity. Dimension it such that its area is a square.
   2. See also: http://developer.android.com/training/custom-views/create-view.html
   3. The new MazePanel class assumes the responsibility to draw anything necessary on the screen in your PlayActivity in the little square area where you see that maze from a first person view and as a map.

4. Adjustments:
    1. Comment out awt and swing import statements and all corresponding lines of code in MazePanel. This will often lead to methods with no code inside.
    2. You need to implement the drawing capabilities from the MazePanel skeleton to draw lines, polygons etc on the new MazePanel view.
        1. The good thing is that you can test this in isolation. (You can work on the draw a line feature and hard code it into the AMaze app to draw a thick red line at a 45 degree angle till you finally see it on the screen).
        2. For this purpose it makes sense to temporarily make the PlayActivity the starting screen in the Android Manifest such that testing this is faster, once the drawing works, you change the Manifest file back to what it should be for the AMaze app.
5. Move additional classes over from the maze code base into the Android project as needed.
    1. For compiler errors: I would initially just comment out existing code that the compiler complains about, put the corresponding lines on a todo list and make the compiler stop complaining that way.

At the end of this step, you have an Android project were the UI operates but is not connected much to the Maze code and you have code in place that draws on your MazePanel view. You pulled all necessary Maze class over and the compiler is able to digest this.

2nd Challenge: Generating a maze in the Generating Activity

Generating the maze takes place in a separate thread to retain a UI that responds to user input. This was the case in the Maze code already and we need to keep it this way. Now, this takes place in the Generating activity which receives its parameter settings with its starting intent being sent by the Starting activity. In the end the new maze configuration needs to be made accessible to the Playing activity.

- There are different design options on how the MazeFactory may deliver the maze configuration to the StatePlay activity. The recommendation is to use a simple shared variable, e.g. a static variable or a static field. See also the discussion of options here http://stackoverflow.com/questions/4878159/android-whats-the-best-way-to-share-data-between-activities.
    1. One can define a static variable, use the singleton pattern (application singleton) or some other way of using a global variable to share data across activities that reside in the same application.
    2. One can serialize the data and communicate it within an intent (serializable or parcelable). This is an interesting learning experience for message passing, but it creates substantial amount of work and the solution does not scale well. Not recommended for this particular assignment.
    3. One can transfer the data into the applications preferences. A solution based on file i/o, not recommended either.

- The MazeBuilder thread updates a visualization for the progress that it makes in the Generating activity. One possible visualization is a progress bar. This may require a Handler to handle messages being sent between the MazeBuilder thread and the main thread to inform the main thread about the progress being made in the MazeBuilder thread.

    - For communication between threads in Android with a so-called Handler, follow this link to a Handler; here is another example with a handler and a discussion on possible issues with a handler.
- Finally, either the generating activity or the MazeFactory needs to start the playing activity when the maze generation is done. It makes sense to let the GeneratingActivity finish() such that the activity stack does not contain it and the back button leads back to the starting activity.

3rd Challenge: Getting the manual operation of the maze game to work, i.e., adjusting the role of MazeController.java.

MazeController.java looses quite a bit of its complexity and its responsibilities. Organizing the generation of a maze according to the user given parameter settings moves into the StateGenerating activity class such that MazeBuilder will interact with the State Generating Activity. MazeController.java's main responsibility is reduced to handle the maze visualization with notifications to registered viewers in state Play where the maze is on display and either the user manually navigates the robot through the maze or the robot driver and robot explore the maze. So the user input for buttons (forward, left, right) on the UI need to connect to the corresponding method calls for the keydown method on the Maze side.

To handle the Maze graphics, there are different options. One is to draw on a Canvas using a View as described here as we continue the way we dealt with the first challenge above. (Note: you can pick other options if you find those more appealing.)
In the Canvas case, drawing the graphics is implemented in an OnDraw method in MazePanel and one can cause the Android environment to call OnDraw by a call to the invalidate method. The main UI thread has a so-called looper and handles requests for redrawing a view in a queue. For the manual operation of the maze, where the user clicks buttons, the integration of the maze graphics and controlling when an update is necessary is reasonably straightforward.

4th Challenge: Getting the animation to work

We want to complete our maze application on Android by integrating the automated maze exploration algorithms, i.e. the various implementations of the RobotDriver.java interfaces.

The key difference to the manual operation is that the automated maze exploration does not produce a sequence of events (user input, screen update, user input, screen update, …) but a sequence of screen updates as an effect of a single user input. So, basically, our robot driver produces a sequence of images much like an animation, a movie.

The remaining changes will basically touch 2 screens:
State Play:
- **What does not work I:** feeding a sequence of screen update requests (method: invalidate) to the UI thread is noticed by the UI thread. It optimizes its own performance by skipping intermediate updates and simply goes to the last screen. The visual effect is that the maze exploration starts and basically reaches the exit immediately.
- **What does not work II:** as a fix to the UI thread optimization, one may think that slowing down the UI thread with sleep commands may yield the desired effect. This is a very bad idea because Android expects the UI thread to be responsive. Android may decide to bring

up a user dialog (ANR) and offer to kill the app as it does not seem to respond (assuming it hangs and thus better be killed).

- **What works:** For the automated exploration with the RobotDriver, it is advisable to consider using a Handler for the UI thread and feeding runnable tasks for robot steps into the handler such that the UI thread faithfully performs all requested updates (check method **postdelayed**). Perform an internet search for "Android postdelayed UI update", see e.g. the section http://android-developers.blogspot.com/2007/11/stitch-in-time.html and http://stackoverflow.com/questions/3765161/updating-ui-with-runnable-postdelayed-not-working-with-timer-app
- **What also works:** the previous solution is not necessarily the only one. Feel free to come up with something else, your solution is graded simply by checking if it works or not.

5th Challenge: Loading and storing mazes

The “revisit” option in combination with the skill-level selection on the starting screen allows the user to replay the maze of same size that was played before. This feature can be implemented by loading a maze from file in the generating activity if the user chooses to revisit a maze. If the user chooses to explore a newly generated maze, that maze should be stored in a file right after its generation in the generating activity. Loading the maze takes place in the generating activity as it takes time. In order to make this happen:

1. You can use the existing MazeFileReader and MazeFileWriter for loading/storing a maze configuration. There is only one adjustment needed in order to store the color of a segment which you hopefully did for the previous project.
2. You need to figure out how to access the file system such that you can read/write files. This is quite particular in Android as different kinds of storage are supported. For our maze file, it makes sense to have them private to the application and readable and writable.
3. You will have to store at most 1 maze per skill-level. You most likely have files like maze0.xml, maze1.xml, …, maze3.xml. For large mazes, the required storage space is huge, which makes file access time consuming. Thus we will secretly limit this feature to mazes of size 0-3. For higher skill-levels, the app simply generates a new maze instead of loading/storing it.

Options for Bonus Points: Make the maze game more appealing

There are plenty of ways to make this game more appealing. Some of these options have been successfully explored by students in previous classes:

- Background pictures in general and more attractive graphics for the walls in the FirstPersonDrawer to create a theme such as the “Corn Maze”, “Zelda”, “James Bond”… (hint: check documentation for BitmapShader)
- Sounds, vibrations
- Music (make sure music starts and stops with activities, beware of the lifecycle of an activity)
- Swipe gestures
- Voice control
- Provide orientation to a user, e.g., compass rose (North,South,East,West) or background pictures (visibility of sun).

If you do something in this direction, please make sure you let the grader now in order to get bonus points for your efforts and achievements.

Other options for Bonus points:

- Explore technical aspects of Android development that are of interest to you
- Other UI concepts such as fragments, animated transitions
- Providing Junit test code for other classes outside of the current assignments, e.g. the BSPnodes

Notes

- **Temporary feedback on what the program is doing**: Use the Log.v(), Log.e()… methods and LogCat to obtain output from the Maze program.
- **As a general advice: do not attempt a big bang integration**, identify individual steps, e.g. the integration of the maze generation and make this one work before integrating the graphics part or vice versa. For the reimplementation of the MazePanel code: reimplement the individual functions and test them before you attempt to port the FirstPersonDrawer and the MapDrawer into the Android project.

Grading

- SVN
    - You will receive points for a subversion release 7.0 that includes source code that can be checked out by the grader.
- You will receive points if your Android app can be executed on an AVD Nexus 4 for a particular API level (pick one in the range 19 or higher) specifically if
    - one can select options for the maze generation (size, algorithm)
    - one can move from the initial activity to the generating activity and a random maze gets generated with the MazeFactory in the generating activity.
    - a newly generated maze (size 0,1,…,3) is automatically stored in a file (one file per size) such that one can play it again at a later point in time
    - one can move from the initial activity to the generating activity and a maze is loaded from file according to the chosen skill level (size 0,1, …,3 are supported)
    - one can select if one sees the maze from top, its walls, and its solution.
    - one can move from the generating screen to the playing screen once the maze is generated.
    - on the playing screen, the maze is visualized in the first person perspective and one can move left, right, forward and the screen updates for a manual exploration.
    - one can manually play the game and get to the exit.
    - one can select automated drivers (Pledge, WallFollower, Wizard) and observe how the driver tries to find the exit with in the first person perspective and on the map
    - while the driver explores the game, one can pause and subsequently continue the animation
    - one gets to the winning finish screen if one reaches the exit and sees energy and path information there.
    - one gets to the loosing finish screen if the driver runs into a wall or if the driver runs out of energy before reaching the exit
    - on the finish screen, on can go to the starting screen and can play another round.
- You will receive points for feedback produced at runtime with Log.v / Log.e etc commands such that one can see how the app internally works.
- As a general remark: the emphasis is on code that works, not on elegance, not on efficiency, not on the latest & greatest & coolest Android feature.
- **Bonus points:**

- Please provide information to the grader what extras you did and how the grader can operate your app to see this in action.
- If you have plans for this and you are not sure if those would pay off, please ask!