Curtin University — Discipline of Computing

Mobile Application Development (COMP2008)

Semester 2, 2019

Assignment

Due: Thursday 24 October, 23:59:59.

Weight: 25% of the unit mark.

Your task is to create a (very simple) city simulator game. You can reuse code from any of the practical worksheets as appropriate (provided that it is either your original code, or code sourced from this unit's Blackboard area). In particular, there are significant similarities to worksheet 3.

1 Requirements Flexibility

Before we get into what you're making, here are some broader points about the nature of this assignment:

- You are expected to write an Android app, using the standard Android framework. However, you are free to use *any version* of the standard API. You are free to work in Java, Kotlin, or both.

  You are *not* free to use alternative frameworks or libraries not covered in the unit.

- You may use your own personal laptop, or the Linux lab machines in building 314, to demonstrate your assignment. (This is more lenient than in many computing units, due to the complexities of Android Studio and the difficulties of moving between environments in this situation.)

- Where this document says "**Optional**", it indicates valid alternative approaches or additional features that are *not worth any extra marks*. They are simply a suggestion of what else you could do if you prefer.

2 Overall App Structure

Your app must have the following screens:

A "title" screen. This should be the default screen that is initially shown when the app first starts up. It will simply show the title of the app, your name (as the author of the app), and buttons to launch the settings screen and the map screen.

A "settings" screen. The game will have various settings that affect its difficulty; see the `Settings` class in the diagram in Section 3, and also Section 4.1.

A "map" screen. This will resemble worksheet 3, showing a scrollable (in one direction) 2D landscape with houses and roads, and is the core part of the game. In worksheet 3, you simply had a list of structures to place on the map. In this assignment, you will need more sophisticated functionality, and this is described below in sections 4 and 5.1.

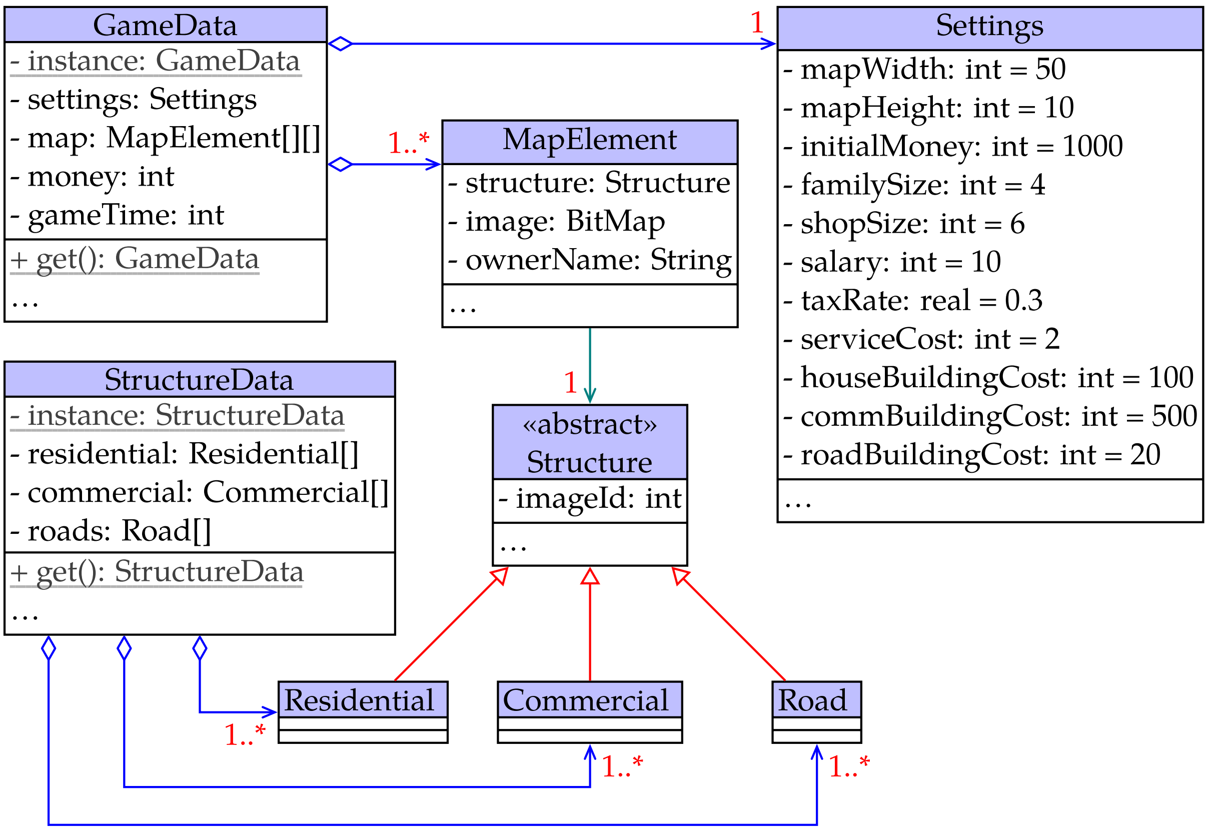
A "details" screen. This will be launched from the map screen in order to present editable information on a particular grid cell in the map.

Activities are the obvious (and probably simplest) way to implement these screens.

Optional: You may instead use fragments to implement different screens.

3 Game Data

The structure of the data is similar to worksheet 3, except there are now officially three different kinds of structures, each with different uses. The following UML diagram gives an overview of the expected model classes:



Optional: You are free to improve on this design if you like, provided that your solution can do everything that the above class structure can. In particular, the `Residential`, `Commercial` and `Road` structures do not necessarily need to be represented as subclasses.

3.1 Changeable Data and Database Storage

In the above diagram, GameData and MapElement represent the data that changes as the game is played. Settings contains values that affect gameplay, at least some of which can be changed by the player *before starting* a game. This information must be stored in a database, such that, if the app is restarted, the player can resume from where they left off.

That said, you do not need to store any *unchangeable* settings; see Section 4.1. You also do not need to store the image field in MapElement, and by default this should be null; see Section 5.2.

3.2 Structure Data

The Structure, Residential, Commercial and Road classes, simply represent the available *types* of structures, including the integer ID for a drawable image. StructureData is simply a place to keep them all together.

None of this changes at runtime – it is all constant – and so there is no need to store it in a database.

Optional: You could allow the user to edit it, if you like, and then store it in a database. This is likely to represent a large amount of extra work!

Optional: You can use artwork sourced from elsewhere if you like, provided you have permission from the artist, and you cite your source(s).

4 Game Logic

The game starts when the map screen is first shown. The map must be a grid with the width and height specified by the Settings object, and the player must start with the amount of money given by initialMoney.

There is no particular "win" condition. It is assumed that the player just wants to build the best city possible, for their own amusement. However, the player can lose if their money falls below $0. It will suffice to display a "Game Over" message, somewhere on the screen, if this happens. You can allow the mechanics of the game to continue after this point nonetheless.

There must be a way to get back to the title screen, even if this is simply via the standard Android "back" button.

4.1 Settings

Prior to the game starting, the player may adjust various numeric settings within the settings screen. At a minimum, these must include mapWidth, mapHeight and initialMoney.

They have default values as per the above diagram.

Optional: You can make any or all of the *other* settings player-configurable as well. You can also allow the player to adjust any/all of them *after* the game has started if you like.

4.2 Construction and Demolition

On the map screen, the player will have the ability to build structures: houses ("residential"), shops/factories/etc. ("commercial") and roads. Unlike in worksheet 3, each type of structure costs a certain amount of money to build. However, the player can also demolish any existing structure for free.

The `GameMap` object defines how much money the player has, and `Settings` defines the cost of building each each kind of structure (`houseBuildingCost`, `commBuildingCost` and `roadBuildingCost`).

Similar to worksheet 3, it is suggested that your app first gets the user to select from a list of possible structures, and then gets them press the grid cell where they want that structure to be built. To enable demolition, it is suggested that you make the player press a "Demolish" button, and then press the grid cell they want to demolish.

You can reuse the drawable XML files from worksheet 3 to show the different structures: "`ic_building1.xml`" to "`ic_building8.xml`", and the "`ic_road_<direction>.xml`" files (also provided alongside this assignment specification). By default, the entire map can simply be blank green, and every grid cell can be built upon.

Optional: You can retain or modify the background map layout from worksheet 3 if you like – the sea, coast and land – and specify that only land (and perhaps coast) cells can be built upon.

However, there are a few key building constraints:

- The player cannot build anything that they do not have enough money for.
- The player cannot build anything in a grid square that contains an existing structure. They must explicitly demolish the existing structure first.
- The player cannot build residential or commercial structures that are *not* directly adjacent to a road.

  (If you like, you may instead implement stricter requirements that try to ensure that the player must build something that looks like a functional town/village.)

4.3 Time and Money

The game will have an internal notion of "game time", expressed as an integer, starting from zero. You should add a "Run" button to the map screen to step forward one "time unit" (`gameTime++`). The player will need to press it regularly (as often as they like) to create the effect of time passing.

Optional: You can instead have the game automatically step forward one "time unit" every second (or so), by using a `Timer` object. It's perhaps best to avoid this unless you already know how `Timers` work, as they're not taught in this unit.

During each time unit, when `population > 0`, the player will gain or lose money according to the following formula:

```
money += population * (employmentRate * salary * taxRate - serviceCost)
```

Where:

- The initial value of money is configurable, and by default 1000;
- `population = familySize * nResidential;`
- `employmentRate = min(1, nCommercial * shopSize / population)` (noting that this is real division, and that the value is undefined when there are no people, and otherwise is in the range [0, 1]);
- `nResidential` and `nCommercial` are the number of grid cells in the map that are occupied by residential and commercial structures, respectively;
- `familySize`, `shopSize`, `salary`, `taxRate` and `serviceCost` are in the `Settings` object.

5 In-Game Information

5.1 Status Information

The map screen must show the following information, in some reasonable form:

- The current game time; i.e., the number of time units elapsed since the start.
- The player's current amount of money.
- The player's most recent income; e.g., "+$40" if the player's money increased by that amount in the last time unit.
- The current population.
- The current employment rate (as a percentage).

5.2 Structure Details and Customisations

The final screen, "details", must be accessible via the map screen. It must show information on a particular structure in a particular grid cell. The screen only needs to be shown for grid cells that actually contain structures.

It is suggested that you make the player press a "Details" button, and then press the grid cell they want to show details for.

It should contain the following:

- The grid coordinates of the structure; e.g., “row = 4, col = 3” or “5 north, 1 east” (making sure the player can distinguish the horizontal and vertical numbers).
- The (uneditable) structure type – “Residential”, “Commercial” or “Road”.
- An editable name. By default, this is the same as the structure type, but the player should be able to set it to anything they want. There is no requirement to display this name outside of the “details” screen though.
- A feature for taking a *thumbnail* photo, by invoking a camera app. This is what the `BitMap` “`image`” field is for in `MapElement`. When taken, this photo should be used to represent the structure on the map (in just this particular grid cell), in place of the standard structure image.

  There is no requirement to save this photo in the database.

  Optional: Ideally, of course, it *would* be saved in the database, and you could do this using a “BLOB”-typed column.

  You could also allow the player to pick images from a photo gallery app.

6 Submission

Submit your assignment electronically, via Blackboard, before the deadline. To submit, do the following:

(a) Fill out and sign a declaration of originality. A photo, scan or electronically-filled out form is fine. Whatever you do, ensure the form is complete and readable! Place it (as a .pdf, .jpg or .png) inside your project directory.

(b) Zip up your entire project directory as-is (as a .zip or .tar.gz file). Leave nothing out.

(c) Submit your zip/tar.gz file to the assignment area on Blackboard.

(d) Re-download, open, and run your submitted work to ensure it has been submitted correctly.

You are responsible for ensuring that your submission is correct and not corrupted. You may make multiple submissions, but only your newest submission will be marked. The late submission policy (see the Unit Outline) will be strictly enforced.

Please note:

- DO NOT use WinRar.
- DO NOT have nested zip/tar.gz files. One is enough!
- DO NOT try to email your submission as an attachment. Curtin’s email filters are configured to *silently discard* emails with potentially executable attachments.

In an emergency, if you cannot upload your work to Blackboard, please instead upload it to Google Drive, or a private Github repository, or another online service that *preserves immutable timestamps* and is *not publicly accessible*.

7 Marking

7.1 Demonstration

You will be required to demonstrate and discuss your application with a marker in-person. You may use your own laptop OR the Linux lab machines in building 314 to do this.

Most of the marks for your assignment will be derived from this demonstration. You must either be available during the practical sessions in week 14, or schedule an alternate time (in agreement with the marker/lecturer) *before* the start of the exam weeks.

Be prepared for the demonstrator to ask that you copy/run your assignment off a particular USB stick, or ask that you *re-download* it from the submission area on Blackboard, or similar. The demonstrator needs to be certain that the version demonstrated is the version submitted, so you cannot simply set everything up beforehand in preparation.

The demonstrator will ask you to rebuild and run your application, and to demonstrate the major features of it. They may ask you about any aspect of your submission.

7.2 Inspection

A smaller proportion of your assignment marks will come from a marker inspecting your code. This does not require your attendance, and is separate to the demonstration.

The marker will be looking to ensure that your code is reasonably well commented, well structured, well formatted, and isn't obviously defective. They may also confirm whether aspects of the functionality are present and correct, whether or not they have already been demonstrated.

8 Academic Integrity

This is an assessable task. If you use someone else's work, or obtain someone else's assistance, to help complete part of the assignment that is intended for you to complete yourself, you will have compromised the assessment. You will not receive marks for any parts of your submission that are not your own original work.

Further, if you do not *reference* any external sources that you use, you are committing plagiarism and/or collusion, and penalties for Academic Misconduct may apply.

Please see Curtin's Academic Integrity website for information on academic misconduct (which includes plagiarism and collusion).

The unit coordinator may require you to provide an oral justification of, or to answer questions about, any piece of written work submitted in this unit. Your response(s) may be referred to as evidence in an Academic Misconduct inquiry.

End of Assignment