

Computer Science Large Practical

Stephen Gilmore (Stephen.Gilmore@ed.ac.uk)

School of Informatics

Document version 1.0.4.

Issued on: November 15, 2017

About

The Computer Science Large Practical is a 20 point Level 9 course which is available for Year 3 undergraduate students on Informatics degrees. It is not available to visiting undergraduate students or students in Year 4 or Year 5 of their undergraduate studies. It is not available to postgraduate students. Year 4, Year 5 and postgraduate students have other practical courses which are provided for them.

Scope

The Computer Science Large Practical consists of one large design and implementation project, done in three parts. The first part is administrative only, requiring setting up and populating a source code repository for the practical. The second part is a design document, presenting the plan of the implementation work which will realise the design. The third part is the implementation, which should be a well-engineered implementation of the previously-supplied design.

Part of the CSLP	Deadline	Out of	Weight
Part 1 (Admin)	16:00 on Friday 13th October	0/1	0%
Part 2 (Design document)	16:00 on Friday 10th November	100	50%
Part 3 (Implementation)	16:00 on Friday 15th December	100	50%

Please note that although Part 1 of this practical is not weighted, you are strongly encouraged to do it both in order to ensure that you are keeping up with the material of the course as it progresses and because your use of your source code repository is a factor in the assessment for Part 3 of the practical. Parts 2 and 3 are equally weighted and constitute the assessment for the Computer Science Large Practical. There is no exam paper for this course.

Introduction

The requirement for the Computer Science Large Practical is to use the *Android Studio* development environment to create an app implemented for an Android device. The app implements a location-based mobile phone puzzle game which allows users to follow a map and collect words which have been scattered at random around the University of Edinburgh's Central Area. The words make up the lyrics of a well-known song and the puzzle aspect of the game is to guess the song from the words which have been found. Given that it is a song-based puzzle, the game is called *Songle*.

— ◊ —

The implementation language of the app is to be *Kotlin* (<https://kotlinlang.org>), a modern, strongly-typed programming language with functional language features which runs on the Java Virtual Machine. User-interface parts of the app are coded in XML. For a brief introduction to Kotlin, please visit <http://kotlinbyexample.org>

— ◊ —

In the Songle game, words are collected by visiting their location on the map, by which we mean that the player literally moves to that location with their mobile phone. There are five different versions of the map for each song, each giving progressively more hints to help the user guess the song more easily.

Map version 1: No additional information is given, words are *unclassified*, and only 25% of the words are displayed.

Map version 2: Words are classified as either *boring* or *notboring*, and only 50% of the words are displayed.

Map version 3: Words are classified as either *boring*, *notboring* or *interesting*, and only 75% of the words are displayed.

Map version 4: Words are classified as either *boring*, *notboring* or *interesting*, and all of the words are displayed.

Map version 5: Words are classified as either *boring*, *notboring*, *interesting* or *veryinteresting*, and all of the words are displayed.

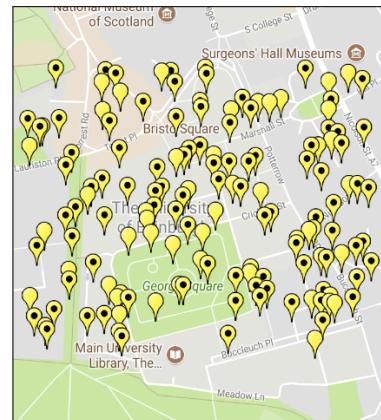
Words which are classified as *boring* are short, common words (such as ‘the’, ‘an’, ‘and’, etc.) which are no help in trying to guess which song the words come from. Words which are classified as *veryinteresting* are longer, less common words (such as ‘Scaramouche’, ‘thunderbolts’ and ‘lightning’) which are much more likely to help the user guess the song the words come from. Knowing which words are interesting and which are not allows the user to prioritise collecting certain words instead of others, making it quicker to identify the song from the words which have been found. The five different kinds of maps are shown in Figure 1.

Key to the symbols used in the maps:

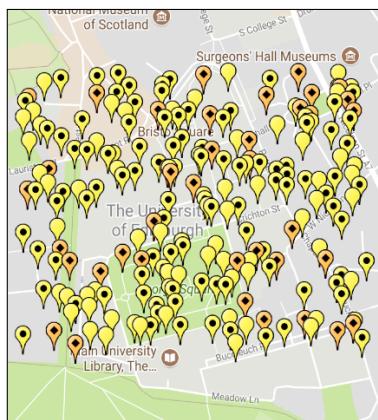
unclassified (♀); *boring* (♂); *notboring* (♂); *interesting* (♂); *veryinteresting* (♂)



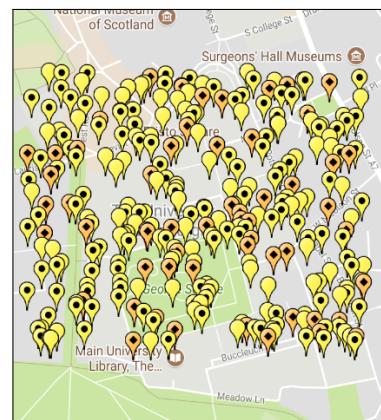
Map 1: 25% of the words;
unclassified



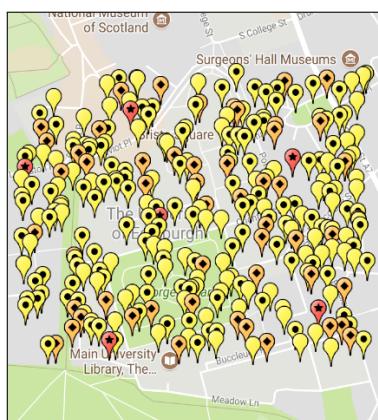
Map 2: 50% of the words
boring / notboring



Map 3: 75% of the words;
boring / notboring / interesting



Map 4: 100% of the words
boring / notboring / interesting



Map 5: 100% of the words;
boring/notboring/interesting/veryinteresting

Figure 1: The five different kinds of maps, giving progressively more information.

The game is backed by a collection of songs with associated maps. The list of available songs is on-line at

- <http://www.inf.ed.ac.uk/teaching/courses/csdp/data/songs/songs.xml> and
- <http://www.inf.ed.ac.uk/teaching/courses/csdp/data/songs/songs.txt>

This XML document contains the information which the player of the game has to guess. It is a numbered list of songs together with the name of the artist and the name of the song. A YouTube link to the song is also supplied.

— ◊ —

This collection of songs is periodically extended with additional songs in order to keep players interested in the game as time goes by. Your app should check for a new version of this XML document every time the user begins a new game. The XML document contains a timestamp which is updated every time a new version of the document is released. The `data/songs/songs.xml` file may be updated at any time so it is important to use the on-line version to ensure that you are providing the player with new songs and maps as they are released. Downloading and bundling this XML document with your application would not achieve the desired result. The format of this XML file is illustrated in Figure 2.

```

<?xml version="1.0" encoding="UTF-8"?>
<Songs timestamp="2017-09-10T20:10:22.563+01:00[Europe/London]"
      root="http://www.inf.ed.ac.uk/teaching/courses/csdp/data/songs/">

  <Song>
    <Number>01</Number>
    <Artist>Queen</Artist>
    <Title>Bohemian Rhapsody</Title>
    <Link>https://youtu.be/fJ9rUzIMcZQ</Link>
  </Song>

  <Song>
    <Number>02</Number>
    <Artist>Blur</Artist>
    <Title>Song 2</Title>
    <Link>https://youtu.be/SSbBvKaM6sk</Link>
  </Song>
  ...
</Songs>

```

Figure 2: Sample of the XML format used for the list of Songle songs. The first song is “Bohemian Rhapsody” by Queen. The second song is “Song 2” by Blur. New songs will be added at the end of the document. Blank lines have been added in this figure to increase readability.

There is a *Songle Word Map* for each song, made available in the Keyhole Markup Language (KML) format used by Google Earth and other geographic visualisation software. The word maps are also available in HTML and text formats for ease of reading and browsing online. (Most web browsers do not display KML format natively.) The word maps and song lyrics for song number 1 in the database are available at the following locations in the folder `data/songs/01`:

- <http://www.inf.ed.ac.uk/teaching/courses/csdp/data/songs/01/lyrics.txt>
- <http://www.inf.ed.ac.uk/teaching/courses/csdp/data/songs/01/map1.html>
- <http://www.inf.ed.ac.uk/teaching/courses/csdp/data/songs/01/map1.kml>
- <http://www.inf.ed.ac.uk/teaching/courses/csdp/data/songs/01/map1.txt>
- <http://www.inf.ed.ac.uk/teaching/courses/csdp/data/songs/01/map2.html>
- <http://www.inf.ed.ac.uk/teaching/courses/csdp/data/songs/01/map2.kml>
- <http://www.inf.ed.ac.uk/teaching/courses/csdp/data/songs/01/map2.txt>
- <http://www.inf.ed.ac.uk/teaching/courses/csdp/data/songs/01/map3.html>
- <http://www.inf.ed.ac.uk/teaching/courses/csdp/data/songs/01/map3.kml>
- <http://www.inf.ed.ac.uk/teaching/courses/csdp/data/songs/01/map3.txt>
- <http://www.inf.ed.ac.uk/teaching/courses/csdp/data/songs/01/map4.html>
- <http://www.inf.ed.ac.uk/teaching/courses/csdp/data/songs/01/map4.kml>
- <http://www.inf.ed.ac.uk/teaching/courses/csdp/data/songs/01/map4.txt>
- <http://www.inf.ed.ac.uk/teaching/courses/csdp/data/songs/01/map5.html>
- <http://www.inf.ed.ac.uk/teaching/courses/csdp/data/songs/01/map5.kml>
- <http://www.inf.ed.ac.uk/teaching/courses/csdp/data/songs/01/map5.txt>
- <http://www.inf.ed.ac.uk/teaching/courses/csdp/data/songs/01/words.txt>

The file `words.txt` contains the song lyrics with line numbers which count the lines of the song. To prevent ambiguity, blank lines between verses are also numbered. The line number and the line of the song are separated by a TAB character (`\t`).

— ◊ —

The format of the KML files for the Songle word maps is outlined in Figure 3. A KML document is a list of Placemarks. Each Placemark contains a name giving the unique numerical identifier of the place, a description identifying which word of the song is available here, and a Point. A Point has coordinates in the format $\langle \text{longitude}, \text{latitude}, \text{height} \rangle$ where the height is always 0. In designing your game you should decide how near a Placemark the player physically needs to be before they can be considered to have collected that word. GPS-based devices cannot determine your true location perfectly but the Android LocationManager API at least attempts to determine the accuracy of its estimated location.

```

<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
<Document>

  <Style id="unclassified">
    <IconStyle>
      <scale>1.75</scale>
      <Icon>
        <href>http://maps.google.com/mapfiles/kml/paddle/wht-blank.png</href>
      </Icon>
    </IconStyle>
  </Style>

  <Placemark>
    <name>1:2</name>
    <description>unclassified</description>
    <styleUrl>#unclassified</styleUrl>
    <Point>
      <coordinates>-3.1920514249267513,55.94610672008997,0</coordinates>
    </Point>
  </Placemark>

  ...
  <Placemark>
    <name>26:4</name>
    <description>unclassified</description>
    <styleUrl>#unclassified</styleUrl>
    <Point>
      <coordinates>-3.188107575106777,55.94423294710405,0</coordinates>
    </Point>
  </Placemark>

</Document>
</kml>

```

Figure 3: Sample of the KML format used in Songle word maps. The first map point is the second word on the first line of the song (1:2). The final map point is the fourth word on line 26 of the song (26:4). This is an extract from `data/songs/01/map1.kml`, the first version of the map for the first song in the database so the words at positions 1:2 and 26:4 in the song are found by referring to `data/songs/01/words.txt`. Blank lines have been added in this figure to increase readability.

The tags `<Document>` and `</Document>` were previously omitted in error, leading to invalid KML. These tags have been added to all KML files stored on the webserver.

All points on every map have a latitude which lies between 55.942617 and 55.946233. All points on every map have a longitude which lies between -3.184319 and -3.192473 .

Forrest Hill (55.946233, -3.192473)		KFC (55.946233, -3.184319)
	<i>Songle is played here</i>	
(55.942617, -3.192473) Top of the Meadows		(55.942617, -3.184319) Buckleuch St bus stop

Bonus features

In addition to the game features described above you should design and implement some *Bonus Features*, which set your app apart from others. These may be enhancements which are intended to make the game more interesting to play, or more rewarding, causing the user to play more frequently, or for longer sessions. What the bonus features are is up to you but you could consider enhancements in areas such as:

- user preferences, allowing the user to decide whether to solve already-solved problems, or be presented with a new song every time;
- giving hints such as free words, or even displaying a line from the song;
- recording the distance walked by the user while playing the game;
- timing-based play (play against the clock);
- play modes (easy, moderate, difficult) or attainment levels (beginner, advanced, expert);
- a scoring system which reflects the attainment level, or play mode; or
- ‘background’ mode where words are collected without user intervention while walking, without the user needing to click to confirm collection.

You are not limited to the items above; this list is only to prompt you to think about your own bonus features. You should aim to add *two* additional bonus features of your own invention, but games with more than two additional bonus features are also welcome.

Computer science aspects of the practical

This practical helps you to improve three useful skills in computer science:

- **learning new programming idioms:** you will learn some new programming concepts and gain experience of a functional programming language on the JVM;
- **using version control systems:** you are to use the Git version control system to manage the source code of your application—learning how much and when to commit code is a useful skill;
- **writing readable source code (in an unfamiliar language):** the Kotlin source code which you submit will be inspected for clarity and readability (as well as correctness) so you should try to write clear, easy-to-read code.

Frequently asked questions

- *I don't have an Android device. I've never written an app before. How can I do this practical?*
 - You don't need to have an Android device to do this practical exercise. The software which you develop will run on an emulator which is freely available for Windows, Mac OS X, and GNU/Linux platforms. There is no expectation that you have written an app before: you will learn how to do this in the course of this practical.
- *Can I develop my app on my laptop?*
 - Yes. You are strongly encouraged to do this because it will encourage you to investigate the Android SDK and related libraries. Of course, we recommend taking regular, well-organised backups, which is an important service provided by a version control system.
- *I don't want to learn a new programming language; can I implement my app in Java instead?*
 - That's what's being done in the Software Engineering Large Practical, so you can just transfer to that course.
- *Instead of Kotlin, can I implement my app in Ruby/Python/Scala/C#?*
 - No, not for this practical. We need all students to be working in the same programming language in order to make a fair assessment.

However, Kotlin is not an arbitrary choice. Kotlin is the latest-and-greatest programming language for the Android platform and there are many people believe that it may be the future of Android development, becoming used in more and more projects.

- *Do I have to develop in Android Studio? I much prefer Eclipse/Emacs/vi etc.*
 - You are required to submit an Android Studio project so we strongly recommend developing in Android Studio for this practical exercise. An Android project developed in Eclipse uses a different build system from Android Studio and can require some significant effort to be made to work in Android Studio. Android development in Eclipse has been unsupported since mid-2015 so you will not be able to access the latest libraries and Android services. Android Studio also contains tools to convert Java code to Kotlin.
- *Is there a specified device for this practical or a specified Android version?*
 - No. You can choose an Android device and an Android version. If you have an Android device then you could choose a suitable specification for that device, to allow you to test your app on a real device. If you do not have an Android device then choose the emulator for a relatively recent device and a relatively recent version of the Android platform.
- *Who owns the intellectual property of the code that I submit? Can I make it public or extend it after the practical is over?*
 - You own the code which you produce for this practical; the University has no claim on any of it. After the practical is over you can extend or use your code in any way that you like. You can make it a part of your public portfolio of work and release it open-source or under any other software licence. However, please wait at least *eight calendar days* after the submission deadline for Part 3 of the practical before making your repository public because the University allows late submission (with daily penalties) for up to seven calendar days after a deadline; see Appendix A and the link therein for details.
- *I have a server where I can host web services. Can I transfer some of the app's functionality to the server side?*
 - In principle, yes, but the server-side code will also have to be written in Kotlin. You will also need to submit your server-side code for assessment, and it too should be readable and clear.

Part 1

Computer Science Large Practical

Stephen Gilmore (Stephen.Gilmore@ed.ac.uk)

School of Informatics

1.1 Introduction

This part of the CSLP is zero-weighted: all of the assessment is based on Part 2 and Part 3 of the practical. Nonetheless, you are strongly encouraged to complete this part to ensure that you are keeping up with the material of the course.

— ◇ —

In this project you are creating an Android app. The first part of your work on this practical is the necessary admin of installing the Android Studio software, creating a simple Android project, committing this to a version control system, and notifying the course lecturer when this is done.

1.2 Installing Android Studio

The recommended IDE for Android development is Android Studio, freely available for Mac, Windows and Linux systems. Android Studio is available for download from <https://developer.android.com/studio/index.html>. Android Studio is a platform, not a single application, so when you have installed Android Studio on your system and begin to use it you will see that it periodically asks to download other components such as Android software libraries, Android device emulators, and the like. It will also periodically suggest that you upgrade components to the latest versions. We recommend following these suggestions and installing/upgrading these components when Android Studio prompts you to do so.

— ◇ —

- ⚠ For this practical you will need to use Android Studio 3.0 Beta 6 or higher, available from <https://developer.android.com/studio/preview/index.html>.

1.3 Creating a sample project

Create a new Android project in your installation of Android Studio following the instructions at <https://developer.android.com/studio/projects/create-project.html>. Choose the form factors and API level. Unless you have a compelling reason not to, we recommend choosing Android 4.4 (KitKat) API level 19 or higher. Add an activity to your project such as a Google Maps Activity. Check whether you can run your project using the Android emulator or an Android device.

1.4 Using version control

In the Computer Science Large Practical you will be creating Android project resources such as XML documents and Kotlin files which will form part of your implementation, to be submitted in Part 3 of this practical. These resources should be placed under version control in a source code repository using the Git version control system. The hosting service which we recommend is *BitBucket* (<https://bitbucket.org/>), because this offers free, private Git repositories for small projects. Create a login on the BitBucket service and create a Git repository there referring to Figure 1.4 for an example of how to set up such a repository.

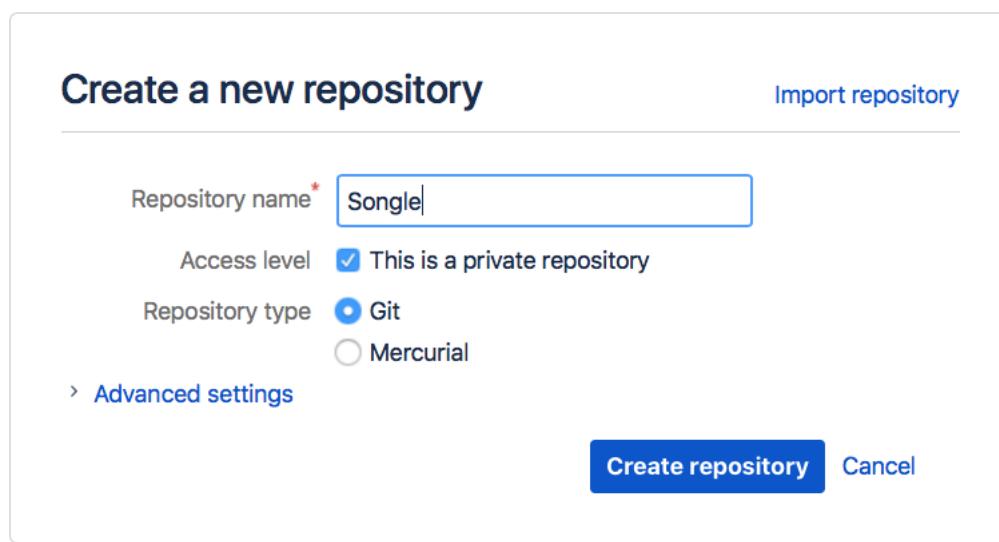


Figure 1.4: Creating a private Git repository (<https://bitbucket.org/repo/create>)

Check your current Android project into your GitHub repository. It is not important what your project does at this stage, only that your repository contains a valid Android project.

Caution: Although Git repositories created on BitBucket are private by default, Git repositories created on GitHub are *public* by default, meaning that anyone and everyone can see and make use of your source code. Private repositories can be created on GitHub if you pay for the service or you have a registered student account.

— ◇ —

Please contact the course lecturer with details if you would like to use a different Git repository host.

1.5 How to submit

Add a standard `README.md` Markdown file to your repository which identifies you by name and university matriculation number.

- If you have used BitBucket for your project then please give BitBucket user `stephengilmore` read access to your repository. (Click the “Send invitation” button in BitBucket.)
- If you have used GitHub for your project then please give GitHub user `sgilmore` read access to your repository. (Settings → Collaborators → “Add collaborator” in GitHub.)

Part 2

Computer Science Large Practical

Stephen Gilmore (Stephen.Gilmore@ed.ac.uk)

School of Informatics

2.1 Introduction

This part and the next part of the CSLP are for credit: all of the assessment is based on Part 2 and Part 3 of the practical, weighted equally. Please now read Appendix A for information on good scholarly practice and the School’s late submission policy.

2.2 Description

The second part of the CSLP involves the creation of a design document, presenting the plan of the implementation work which will realise the design. The design document presents your ideas for your version of the game and is a record of the design decisions which you have taken so far in the practical. It commits you to certain decisions, which are to be realised in the implementation.

— ◊ —

Your design document should record your **design decisions for your app** including a **definitive list of the bonus features** which are offered by your app. Your implementation will be expected to provide these bonus features, in addition to being a playable implementation of the Songle game.

— ◊ —

Your design document should represent the visual layout of your app by being illustrated with **screenshots of the views which you have designed** for the activities in your app. These can be taken either from the Android emulator running a (partially complete) version of your app, or from the visual editor in Android Studio, rendering your XML layouts, with no Kotlin code attached. Include as many as possible, to give the clearest picture of how your app will work in practice. You should aim to show a typical play of the game including at least the user viewing a map, guessing which song this is, and being informed whether or not they are correct.

The expected length of this design document is between 6 and 10 pages. The choice of font, font size, and margins is up to you but please consider the readability of your submission. The submission format is PDF only.

2.3 Typical questions for the design document to answer

[This material previously appeared in Lecture 5 of the course, and is reproduced here for convenience.]

- How is the song that the player has to identify chosen?
- What does a player have to do to collect a word? Does anything happen to the placemark when the word is collected?
- After a player has collected several words can they review them? If so, how do they do that? What does the “review screen” look like?
- When a player thinks that they can guess the song how do they enter their guess?
 - What happens if their guess is correct?
 - What happens if their guess is incorrect?
- When a player thinks that they *can't* guess the song how do they indicate that they give up? What happens then?
- What determines which of the five versions of the map is shown?
- Can the player set the “level of difficulty” of the game? If so, how?
- After a player has identified a song, could that song be chosen again as the puzzle to solve?
- After a player has identified several songs, can they review their list of solved puzzles? What does that “review screen” look like?
- What does the game do if a data connection (4G) is not available? Can it be played at all?

2.4 Allocation of marks

Marks are allocated in this practical according to the following weighting.

Design decisions (40%): Your design document should provide evidence that you have moved forward from the high-level specification of the app which is given on pages 2–7 of this document, making progress towards your implementation. Your design document should answer questions such as those featured in Section 2.3 above and record other design decisions that you have made.

Bonus features (40%): Recall (from page 7) that you are expected to add *two or more* bonus features to your app. Your design document should clearly identify what these are. Marks will be allocated here based on the added value provided by the bonus features which you offer. Are they just some additional decorations or do they actually make the game more interesting to play, or more rewarding?

Design of views (20%): Screenshots are included in your design document to demonstrate your progress and to clarify aspects of your design expressed in your design decisions. Based on the screenshots which you provide, marks are allocated here in recognition of the coherency and consistency of the screens which you show from your application. Do they clearly belong to the same app, giving the impression of an app which has been designed thoughtfully? Do they represent aesthetically-pleasing design of a game which is fun to play?

2.5 How to submit

Please submit your design document from your DICE account using the command:

```
submit cslp cw2 design.pdf
```

(Assuming that your design is in a document called `design.pdf`.)

2.6 Things to consider

- If in doubt, leave it out. Don't promise to deliver bonus features which you have no idea how to implement. Think ahead to your implementation and try to be at least 80% sure that you know how to implement the features that you are promising to deliver. It is better to promise less and deliver more than to promise more and deliver less, so don't make your list of bonus features too long.
- You will need to investigate Android concepts and programming in order to be able to make informed decisions about the bonus features which you will implement.
- Although this is a design document, and you do not have to supply any code at this stage, you are *strongly encouraged to begin coding your app now*, to clarify your ideas about various details of Android application structure and to ensure that you have made a start on the implementation of the app comfortably in advance of the Part 3 deadline for the CSLP.

Part 3

Computer Science Large Practical

Stephen Gilmore (Stephen.Gilmore@ed.ac.uk)

School of Informatics

3.1 Introduction

As noted above, this part and the previous part of the CSLP are for credit: all of the assessment is based on Part 2 and Part 3 of the practical, weighted equally. Information on good scholarly practice and the School’s late submission policy is provided in Appendix A.

— ◇ —

The third part is the implementation. This should be a well-engineered implementation of the previously-supplied design. The code which you submit for assessment should be readable, well-structured, and thoroughly tested.

3.2 Report and documentation

Please submit a report describing the following, and any other special features of your implementation which would be relevant for the marker to know.

- *Your experience of Kotlin as a development language.* Give a report on your experience of using Kotlin to develop your app. You can report on:
 - good features of the language which you found helpful;
 - bad features of the language which you found to be problematic or confusing; and
 - whether or not you would recommend that Java developers should start to migrate to Kotlin.
- *Algorithms and data structures used for the core functions of the implementation.* Describe the algorithms and data structures which are used in your implementation for such features as:

- download and parsing of the XML-based Songle song list from the server;
 - persistent storage of the user’s progress in playing the game; and
 - detection of the words which can be collected at the user’s current location.
- *Parts of your design which have not been realised in the implementation.* It could be that you have not been able to implement something which you had planned in your design. If so, document that here.
 - *Additional features of your implementation which were not described in your design.* It could be that you have implemented something which you had not planned in your design. If so, document that here.
 - (Only if relevant¹.) *Instructions for installing the server-side part of your project, if you have one.* If you have a server-side component to your project, then provide details of how to install and run this component. Note that your server-side component (if any) must also be implemented in Kotlin.
 - *Acknowledgements:* Include an acknowledgement for any part of your code which comes from a publicly-available source such as Android tutorial examples, Android sample projects, or open-source projects hosted on GitHub, BitBucket or elsewhere. You do not need to include an acknowledgement for any code which was presented in the course lecture slides.

— ◇ —

The expected length of this document is between 4 and 6 pages, but shorter or longer submissions will also be accepted. The choice of font, font size, and margins is up to you but please consider the readability of your submission. The submission format is PDF only.

3.3 Allocation of marks

Marks are allocated in Part 3 of this practical according to the following weighting.

Documentation (20%): As described in Section 3.2 above, you are to provide a document describing your implementation. Your document should be a clear and accurate description of your implementation. An important part of your document is your report on your experience of using Kotlin as a development language. Do not neglect this part of your document.

Functionality (30%): Your submission should be a playable implementation of the *Songle* game, extended with bonus features of your choosing. Your submission should be based on your previously-submitted design although you can modify

¹Depending on the bonus features which you decided to implement, a server-side component might have been required; if you do not have one then you need not include anything on this in your document.

and improve your design in whatever way you wish to add improvements. Do not feel limited by the design which you submitted for Part 2, you can add whatever improvements you wish before this submission. Your game should be usably efficient, without significant stalls while playing. Your game should correctly handle other songs being added to the song database, and should not assume a specific number of songs.

Bonus features (20%): Bonus features implemented in your game will be evaluated on whether they make the game more interesting to play, or more rewarding. More significant bonus features are more credit-worthy. If you have had ideas for additional bonus features which were not described in your Part 2 submission then add them here, and document them in your submission document.

Interface (10%): The user interface of your app is an important feature in appealing to potential users. The user interface for your submission should be based on your previously-submitted user interface as seen in screenshots in your Part 2 document — although you can modify and improve the user interface of your app in whatever way you wish to add improvements. As before, do not feel limited by the user interface design which you submitted for Part 2, you can make whatever alterations or improvements you wish before this submission.

Code quality (10%): Your code should be readable and clear, making use of private values, variables and functions, and encapsulating code and data structures. Note that Kotlin encourages the use of (immutable) values over (mutable) variables so consider if a `var` in your Kotlin code could be a `val` instead. All else being equal, code with comments should receive a higher mark than code without comments. Everyone thinks that their code is self-documenting, but it isn't. Note that use of deprecated methods, interfaces, or classes will *not* be a factor in determining code quality; no penalty will be applied for using deprecated methods, interfaces, or classes.

Use of your repository (10%): All else being equal, a project where version control using a source code repository such as BitBucket or GitHub has been used to manage the project code and resources throughout the duration of the project should receive a higher mark than one where version control has been used only a little (or has not been used at all).

Please read also Section 3.6 below for further information on the marking criteria for Part 3.

— ◊ —

3.4 Preparing your submission

- Create a new folder to contain your submission. Assuming that your folder is called `songle`, create two sub-folders `songle/doc` and `songle/android`.

Place a copy of your documentation in the `songle/doc` folder and place a copy of your Android Studio project in the `songle/android` folder.

(Only if relevant.) If your submission has a server-side component then create a third sub-folder called `songle/server` and place a copy of your server-side code in there.

- (Optional, you can skip this step if you wish to.) You can remove any keys which you have generated for Google Maps or other APIs. Replace these with the text `YOUR_KEY_HERE`.
- Make a compressed version of your folder using ZIP compression.
 - On Linux systems use the command `zip -r songle.zip songle`.
 - On Windows systems use **Send to > Compressed (zipped) folder**.
 - On Mac systems use **File > Compress “songle”**.

3.5 How to submit

Please submit your implementation work from your DICE account using the command:

```
submit cslp cw3 songle.zip
```

(Assuming that your implementation is in a ZIP archive called `songle.zip`.)

3.6 Things to consider

- Your submitted Kotlin (and other) code will be read and assessed by a person, not a script. It is not a waste of time to add comments to your code, documenting your intentions. Your submitted code should be readable and clear.
- Logging statements (using `Log.d` and friends) and diagnostic print statements (using `println` and friends) are useful debugging tools for Android apps. You do not need to remove them from your submitted code. It is fine for these to appear in your submission.
- The game should be robust. Failing with a `NullPointerException` or other Kotlin run-time error will be considered a serious fault.
- *[This point has been incorporated into Section 3.3 above.]*

Appendix A

Coursework Regulations

Good scholarly practice

Please remember the University requirement as regards all assessed work for credit. Details about this can be found at:

<http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct>

Furthermore, you are required to take reasonable measures to protect your assessed work from unauthorised access. For example, if you put any such work on a public repository then you must set access permissions appropriately (generally permitting access only to yourself, or your group in the case of group practicals).

— ◇ —

The Computer Science Large Practical is not a group practical, so all work that you submit for assessment must be your own, or be acknowledged as coming from a publicly-available source such as Android tutorial examples, Android sample projects, or open-source projects hosted on GitHub, BitBucket or elsewhere.

Late submission policy

It may be that due to illness or other circumstances beyond your control that you need to submit work late. The School of Informatics late submission policy aligns with the university's Assessment Regulations which applies a penalty of *5% per day* for a maximum of *7 calendar days* after the deadline. For more information, see

<http://web.inf.ed.ac.uk/infweb/student-services/ito/admin/coursework-projects/later-coursework-extension-requests>

The staff of the Informatics Teaching Organisation will approve extension requests up to 7 days in accordance with the “good reasons” defined in assessment regulation 28.6.

— ◇ —

Please note that things that would *not* be considered good reasons include (among other things) last-minute computer problems and loss of work through failing to backup your source code. Frequent commits of your work into a remote source code repository such as BitBucket or GitHub provide an off-site backup of your work which can allow you to continue from where you left off in the case of a disk or other hardware failure, so we recommend committing your work frequently.

Appendix B

Document version history

Version 1.0.1: Fixed a bug in the `submit` command which had the wrong exercise number.

Version 1.0.2: Corrected the KML files on the server to be valid KML. Updated the example in Figure 3 to reflect the new state of the files.

Version 1.0.3: Added Section 2.3 with the list of design questions which were presented in Lecture 5 of the course and Section 2.4 giving the allocation of marks for Part 2 of the practical.

Version 1.0.4: Added Section 3.3 giving the allocation of marks for Part 3 of the practical. Changes to the text are marked with blue change bars in the left-hand margin.