Term 3 Project Descriptions

Overview:

In this final term of your first year, there is just a single programming task - a project that is designed to bring together many of the programming techniques that you have learned over the year. This project represents the final piece of practical coursework for SCC110.

You have a choice of which project to undertake, to be selected from the 4 choices below, and you need to **pick just one** of these projects. Read each of them carefully before choosing which project to undertake.

For each project you need to complete three pieces of work: a design document due in week 22 and an implementation and reflective report both due in week 25.

Each project also has a **start-up task** associated with it, to be completed in Week 21. You should complete this task for the project that you initially favour, in order to get a better feel for the task and to give you valuable insight for your design document.

Project Design (15%)

- You should initially produce a two page design document detailing the following: (i) which project you have selected and what features you hope to implement, (ii) a description of how you intend to break the project down into smaller components, i.e. identify the functional decomposition, (iii) identify the relationships between different components, i.e. what will call what, and (iv) pay special attention to where there are possibilities for code re-use.
- Deadline for Moodle submission: Friday 4th May 2012, 6pm (Wk 22).

Project Implementation (70%) & **Reflective Report** (15%)

- As mentioned above, there are 4 projects to choose from (you choose one), all of which need the basic programming skills that you have learned this year, but they also intentionally encourage you to **learn additional programming skills** for yourself – to make your

project into something meaningful for you, and so that you end up with a piece of code that you can be proud of!

- The **reflective report** should be two pages long and should take as a starting point your project design, and reflect on how closely your final project followed this design (including if/how/why things deviated from your design, what you forgot to include in your design, and if you found anything helpful by having thought through the design first), plus importantly what lessons you should take into the next project that you undertake.
- Deadline for Moodle submission: Tue 22nd May 2012, 10am (Wk 25).
- PLUS Wk 25 demonstration – your code must be demonstrated (and marked) in your allocated Week 25 practical session.

Submission:

As with the term 2 assessment, you must submit all your work[1] electronically to Moodle AND also demonstrate your project in your allocated Week 25 practical session ...

... failure to submit electronically, or failure to demonstrate your work, will automatically mean that you will be assigned a mark of zero.

Time management:

Use your time wisely! Whichever project you choose, use your first 4 practical sessions to help you in your project design and development.

We have learned from many years of experience that when we give students just over 4 weeks to do a task, a handful of them think that they can have a 'break' for over 3 weeks, and then expect to be able to achieve the task (and get endless help from us tutors) over the last few days before submission. From what we see, this never works.

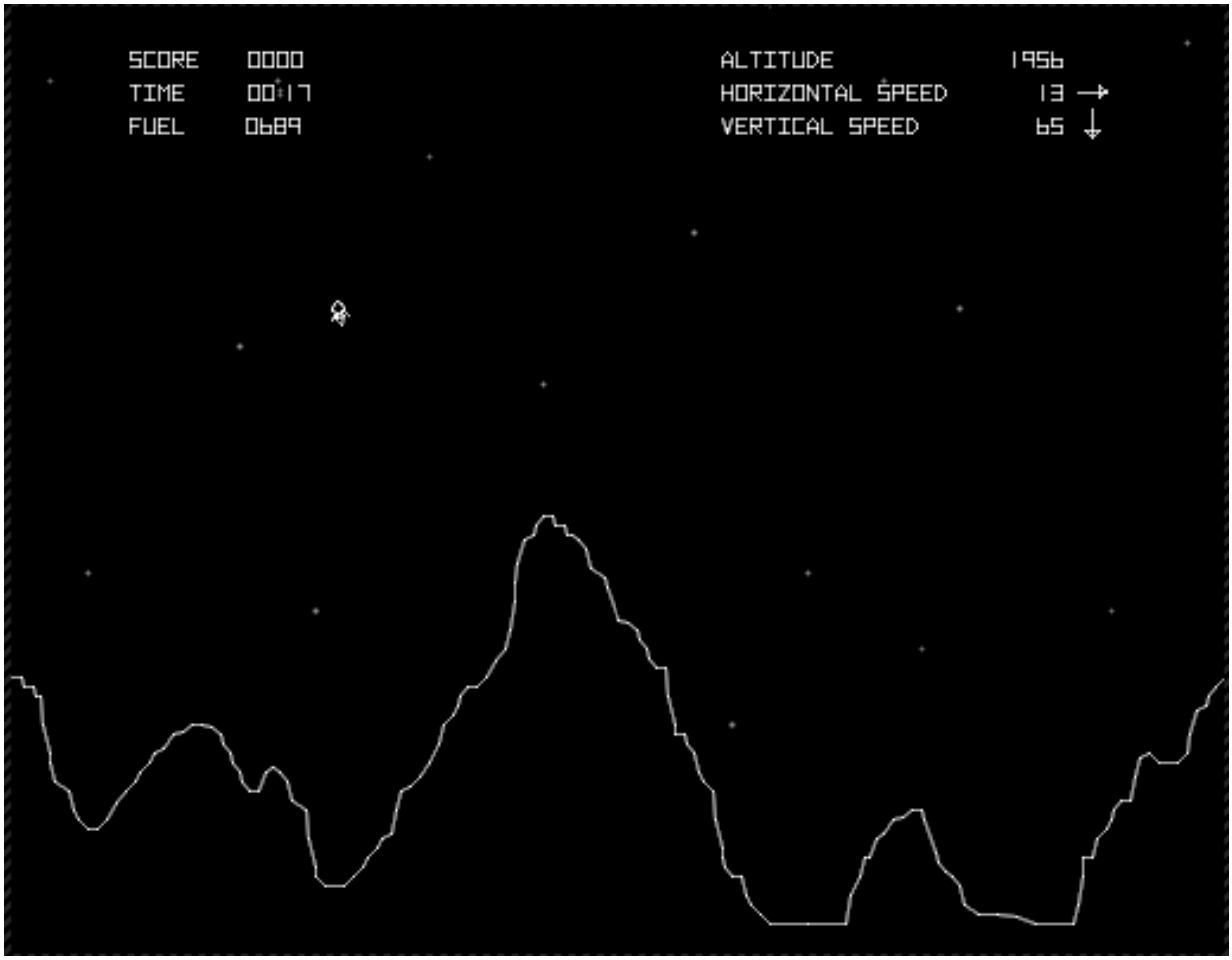
This project is worth a third of the practical component of 110, which means a whole 20% of your final 110 grade ... don't mess it up by poor time management!

Have fun, and good luck!

[1] This includes your design document in Week 22, and all of your coding and your reflective report in Week 25.

Project 1: C – Arcade Classic – Lunar Lander

Lunar Lander started as a text-based computer game written in FOCAL on the DEC PDP-8 in 1969. Later revisited and popularised by Atari in 1979 using a vector display, it became one of the early arcade classics (shown below).



This deceptively simple game challenges the player to control a landing spacecraft onto a number of landing pads under the influence of a simple physics model (gravity, momentum, thrust). Landing pads are graded for difficulty and more difficult pads act as a multiplier on the score. Score is calculated using some heuristic based on time used and fuel remaining. Naturally, if you run out of fuel you will cease to be able to use your rocket engine. If you hit the ground too hard, then your lander crashes horribly!

The aim of this project is recreate your version of this classic on the UNIX command line creatively using ASCII characters (the terminal, not graphics, vector or otherwise). The simplest version of the game will feature a ship, somewhere to land, simulated gravity, keyboard controls for

thrust and direction and a score. The rest, including how to generate a landscape, multiple levels with different gravities, etc. is up to you.

The exercise focuses on your ability to:

- **Solve the challenge** of creating a simple game (game loop, score etc.);
- Exercise your **problem solving skills**, especially in the simulation of the spaceship and its physics;
- **Work out algorithms** to control the ship, generate a landscape, detect collisions, perform the game logic;
- **Generate a GUI** for the user to interact with the ship (using ASCII art & the UNIX **curses** library)
- Demonstrate good code modularity and style.

Start-up task:

This task is **to be completed in your Week 21 practical** in order to give you a better feel for this project and to give you valuable insight for your design document.

- Visit the lunar lander website and play the game a couple of times! http://www.atari.com/arcade/arcade/lunar-lander
- Make notes on the features of the game you can see – pay particular attention to the phases of gameplay (start, middle, end). What happens when you run out of time, fuel or land too hard?
- Read the introduction to the ncurses library (sections 1-1.3): http://tldp.org/HOWTO/NCURSES-Programming-HOWTO/intro.html
- Look back over term 1 and remind yourself of how to log into 'UNIX' using putty and compile C programs.
- Try the 'hello world' example: http://tldp.org/HOWTO/NCURSES-Programming-HOWTO/helloworld.html - you'll need to `#include <curses.h>` in your C file and compile using '`-lcurses`' with `gcc`.
- Now try to create a C program that uses the curses library to let you control the position of a symbol (e.g. '*') on the screen using keyboard input. Tips: lookup `mvprintw`, `getch` and `timeout` in the man pages/online.

Project 2: Java – 'Let it rain!'
(Rainfall data: processing and visualisation)

The aim of this project is to be able to process a large real-world data set and to be able to manipulate/ visualise the data in meaningful ways.

The data that you will use is the set of **rainfall data** from the University's field station at the back of campus, operated by LEC (Lancaster Environment Centre). The data has been collected at 9am every morning going back to 1974, and you have been given access to the entire data set! You may prefer to start by working on a smaller data set covering just two years (see the project resource file RainfallSample.txt) rather than the full data set (see RainfallDataLanc.txt).

The exercise focuses on your ability to:

- Handle a data file;
- Use appropriate **data structures** for the storage of the real-world data;
- **Work out algorithms** to allow accurate (and error-free) manipulation of the data (see some possible questions below);
- **Generate a GUI** for the user to be able to interact with the data;
- Allow the user to select certain **data visualisation options**, e.g. as provided by calling classes/ methods from one of the powerful Java graphing tools such as JFreeChart (http://www.jfree.org/jfreechart/);
- Demonstrate good documentation skills through the use of **Javadoc**.

For a similar (although simpler) exercise, look back at the Week 13 practical notes when we looked at earthquake data. As a starting point, you may like to re-use the SimpleFileReader class and the guidelines from that exercise.

You will need to write **algorithms** that allow you to process the data to give answers to questions like:

- What was the wettest/ driest day in a particular month?
- What was the total/ average rainfall for a month?
- What was the wettest/ driest month from the records?

- Have the seasonal[2] or yearly statistics changed notably over the years (i.e. are any seasonal or yearly trends starting to emerge)?
- ... etc. (you can be as creative as you like here!)

A note on the Lancaster data layout: The data sets contain 3 header rows followed by the data rows – one row per month. Each of the data rows is laid out as follows: Year, then Month, then 31 Day fields. Note that if any of the day fields are not appropriate for the given month (e.g. April 31st doesn't exist), or the data was not available (e.g. the latter part of 2012), then the data cell will contain -99.99.

Additional data sets: In order to go further with your project, you may like to investigate how the Lancaster data compares with the average North West data (or other UK regions)[3]?

Using larger data sets: As a more advanced task, you may like to research the degradation in speed with larger data sets. For example, have a look at the relative efficiency of different mechanisms that you could use to read in data from a file. Which ones are slow for large data sets? Make sure you implement your project using the more efficient ones!

Start-up task:

This task is **to be completed in your Week 21 practical** in order to give you a better feel for this project and to give you valuable insight for your design document.

- Look back at your solution to Week 13's exercise to see how you achieved that task;
- Investigate the rainfall data sets, and work out what you will need to process the data provided. For example, start writing pseudo-code for reading in and parsing the data, and think about what algorithms you will need to start answering the above questions;
- Investigate JFreeChart and decide what data visualisation tools you would like to be able to offer to the end-user of your system.

---

[2] Note that the Met Office identifies the different British seasons as being:
Winter = Dec - Feb, Spring = Mar - May, Summer = June - Aug, Autumn = Sept – Nov.

[3] e.g. http://www.metoffice.gov.uk/climate/uk/datasets/Rainfall/ranked/England_NW_and_N_Wales.txt

Project 3: Java – 'A matter of style?'
(File processing: layout and style issues)

The aim of this project is to be able to allow a user to choose a piece of code that s/he has already developed, and to provide the user with a simple file editor that incorporates layout and style options.

The exercise focuses on your ability to:

- Handle a source code file;
- **Generate a GUI** for the user to be able to interact with the source code;
- Handle **file input** (reading in the required source code file) and **file output** (writing out to file any changes made to source code);
- Be able to offer **'smart indentation'** based on the location of brackets in your code;[4]
- Process a separate file that contains a set of key words for the particular language being supported, and **colour code any key words** that occur in your source code file;
- Demonstrate good documentation skills through the use of **Javadoc**.

As with project 2, as a starting point, you may like to re-use the SimpleFileReader class that you have seen early in term 2 (including Week 13).

Tabbed editing: To go deeper with your project, you may like to investigate offering 'tabbed editing' where you support multiple open files using different tabbed panes to hold the contents of separate files.

Find and replace functionality: How about trying to offer the user the ability to find/ replace text? And perhaps you could also explore how to use **regular expressions** in searching.

[4] For example, as you read through your code, you should keep track of what 'level' you are at based on how many { and } you have encountered so far. For each subsequent 'level', you should indent your code by an additional tab (or set of spaces).

Linking to a compiler: As a much more advanced challenge, investigate offering a 'compile' option, which will take a piece of source code and pass it to the (external) javac/ gcc compiler.

For this advanced task, it will be necessary to move outside Java, and launch an **external** program using system-level commands.[5] There are various ways that this could be achieved, but you may like to investigate the Runtime class[6] or the Process/ ProcessBuilder classes[7]. For further advice, look at the java-tips page[8] that compares these two approaches, or read through 'When Runtime.exec() won't – JavaWorld' to get an idea of some of the right/ wrong ways that Runtime exec can be used![9]

Start-up task:

This task is **to be completed in your Week 21 practical** in order to give you a better feel for this project and to give you valuable insight for your design document.

- Decide on what GUI elements you will offer to the user, and find/ read through any tutorials and/ or examples to discover how any that you have not previously used should be implemented;
- Look back at your Java code from earlier in term 2 to refresh your memory on how to read in files;
- Start writing pseudo-code for the algorithms that you will need in order to offer smart indentation, colour coded keywords, and find/replace functionality.

---

[5] Once you enter the world of system level commands, your application needs to know what platform it is running on. There are obvious differences such as whether to use a forward or backward slash as a file separator, but also what utility gives you a shell through which to enter system commands (e.g. command.exe/ cmd.exe on Windows 95/ later, or terminals on macs). Details about accessing system properties are given on the Oracle tutorial page: http://docs.oracle.com/javase/tutorial/essential/environment/sysprop.html

[6] http://docs.oracle.com/javase/7/docs/api/java/lang/Runtime.html

[7] http://docs.oracle.com/javase/7/docs/api/java/lang/ProcessBuilder.html

[8] http://www.java-tips.org/java-se-tips/java.util/from-runtime.exec-to-processbuilder.html

[9] http://www.javaworld.com/javaworld/jw-12-2000/jw-1229-traps.html - keep reading through to the 4th page, i.e. through to the "Gobbler" class on the last page!

Project 4: Android – 'Sliding Tile Puzzle'

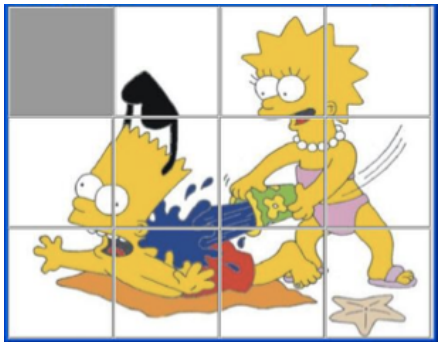
The aim of this project is to provide an additional project for those of you who already feel pretty confident with your Java abilities and wish to explore Google's Android platform. In this project, you will implement a Sliding Tile Puzzle suitable for playing on any Android-based mobile phone.

Making this choice, part 1: Although Android is based around the Java language, it introduces many other totally new structural elements to your coding, and also a totally new way of building GUIs (using XML instead of Swing). For these reasons, we only recommend this project if you are already a reasonably good Java programmer and are prepared to play around with something completely different!

Making this choice, part 2: You will have seen an introduction to Android in Week 21's 110 lecture. If you are still intrigued, try out the start-up task below and see how you get on. If you find things too challenging or too confusing, you still have time to change your project. BUT, whichever project you end up choosing, you will still need to develop your design document by the end of Week 22. So make sure you make the most of the lab time this week (Week 21), so that you can explore Android as fully as possible, and ensure that you are making a properly informed choice!

The task itself:

The aim of the Sliding Tile Puzzle is to allow the player to moves tiles left, right, up or down within the puzzle, the challenge being that you can only move tiles into the empty space. Ultimately, a player tries to reform the original image by moving tiles one by one until every tile is in the correct location.



The individual tiles necessary to re-create this Simpson's tile puzzle have been provided for you in the zipped file, TilePuzzleResources.zip.

This exercise focuses on your ability to:

- Understand the (very different) format of an Android project;
- Create a new project to display the tiles in an appropriate layout;
- Develop code to allow the tiles to be clicked to move them around, and control the game logic so that tiles will only swap place when

they are adjacent (next to) the blank tile;[10]

- Add a 'scramble' button to randomize the positioning of the tiles on the board, ready for the player to re-attempt the puzzle; you may also like to automatically detect when the game has been won;
- Demonstrate good documentation skills with comments/ Javadoc.

Handling multiple activities and intents: To go deeper with your project, you may like to demonstrate your use of multiple activities and intents. For example, you may like to implement a separate start-up screen that allows you to see the current high score, and invites you to start the game.[11] On completion of the game, the number of moves taken is stored in the High Scores table and control then returns to the start-up screen.

Start-up task:

This task is **to be completed in your Week 21 practical** in order to give you a better feel for this project and to give you valuable insight for your design document.

- Download a copy of the sample HelloAndroid project and unzip it;
- In the labs, open up the CSM004 VM which includes the Eclipse/Android environment already installed – see instructions on VMWare View: http://scc-intranet.lancs.ac.uk/Virtualisation/Connecting;
- Start Eclipse (you will need to browse to C:\Program Files\Eclipse) and import your unzipped HelloAndroid project (tick the box that says import these files into my workspace);
- Examine the structure, particularly res/ src, and run the project (note you'll need to create an instance of the emulator for the 1st time);
- Add an additional button to the main.xml file (try editing both via the graphical layout screen and directly into the xml code). Add code in HelloAndroid.java to handle when your new button is clicked;
- You could now try adding an image to your application.

[10] You SHOULD NOT aim to swap actual buttons around. INSTEAD you should make it look like the tiles change place by swapping the images displayed on the buttons.

[11] Perhaps you could also allow the user to choose from more than one tiled image? (You would need to provide the individual tiles for any other image for yourself!)