School of Computing Sciences

Module: CMP-7009A
Advanced Programming Concepts and Techniques
Assignment: Project

Set by: Rudy J. Lapeer – r.lapeer@uea.ac.uk
Checked by: Stephen D. Laycock
Date set: 28 September 2020
Value: 60%

Date due: Report: Assessment Week 1 2021
Demo: Assessment Week 1 2021
Returned by: January 2021
Submission: Blackboard

Learning outcomes
To understand different programming language paradigms.
To be able to program in established general-purpose programming languages such as Java, C++ and C#, as well as learn new languages such as F#.
To understand and be capable of applying the principles of modern software engineering to develop an industry-grade desktop application.

Specification

Overview

To develop a software product following the principles of modern day software engineering and using modern day development platforms and programming languages.

Description

- To develop an industry grade desktop software solution using one or more programming languages on one or more platforms with (an) IDE(s) of your choice.
- The UI can be based on higher level platforms or engines such as wxWidgets, WPF, Unity, SDL, etc. but these should **not be the integral part** of the project and should be considered as a supporting framework.
- No standalone web apps or mobile apps or database apps or projects of which these three are central to the development. However, they can be used as an add-on to one of the projects described below.

Relationship to formative assessment
Formative assessment is based on weekly laboratory (follow-up) meetings with the Module Organiser (MO) and an AT (Associate Tutor).

Deliverables

- Report of no more than 4,000 words but no limit on pages. The use of UML diagrams, pseudo code, explanatory diagrams and figures are strongly encouraged.
- Source code and executable.

Examples/Resources

Generic graphics simulator

Create a software platform which can run graphics (e.g. game, simulation, object rendering) using a variety of different API's (e.g. OpenGL and DirectX) and/or Window Managers (e.g. Win32 and X) and/or on different platforms/OS's (Windows and Apple OS). The platform should be well engineered, i.e. each of the components e.g. Window Manager, Graphics Object(s) and/or Scene Graphs, Graphics Engine, Physics Engine, File loader, etc. should be decoupled from each other and ideally show multiple levels of abstraction.

Visualisation (or tutorial) software for maths/stats with interpreter

- **Maths**: Create a software that can be used to visualise functions of two variables. The software should allow the user to enter particular functions using a certain syntax which is then processed by a lexer (word level, i.e. breaking an input string into tokens) and parser (grammatical level, i.e. interpret the tokens produced by the lexer). Intuitive and interactive visualisation of functions should be provided. Additional functionality could include zero crossings, function differentiation and integration. Examples of maths software are Matlab, Mathematica and Maple (though none of these are actually 'tutorials').
- **Stats**: Create a software to explain a variety of statistical methods ranging from very basic, e.g. average, standard deviation to more advanced topics such as regression, hypothesis testing, Bayesian inference, etc. Examples of stats software are R, SPSS and SAS.

Dynamic simulation software

The Simulator should be capable of simulating a specific dynamical process but also suggest improvements to the infrastructure that is part of the process.
Examples:

- **Traffic flow**: Simulate the traffic flow in a city (e.g. Norwich) at certain locations and (algorithmically) suggest improvements of the infrastructure (road layout, traffic lights, etc.)
- **Aircraft boarding**: Simulate the boarding of an aircraft using various strategies (see http://www.vox.com/2014/4/25/5647696/the-way-we-board-airplanes-makes-absolutely-no-sense)
- **Evacuation**: E.g. in buildings when there is a fire. The software should suggest improvements to buildings to speed up evacuation.
- **Ecology / AI**: E.g. Simulating a (finite) number of species in a natural environment (see for example the bug software) using FSM (Finite State Machines). Alternatively, this could be a simulation of a human society,

where humans with different occupations and/or personalities interact with each other.

Programming the NAO robot
Quite open-ended at the moment. Needs a team of strong and adventurous programmers.

Computer Game with advanced concepts such as PCG (Procedural Content Generation), AI (Artificial Intelligence), MoCap (Motion Capture), advanced lighting and shadowing, …

Own project subject to approval by the teaching staff.

Marking scheme

1. Minimal requirements to obtain a good honours mark (60%):

- A working software product.
- Evidence of adequately using IID (Iterative and Incremental Development - as opposed to a waterfall approach) or AP (Agile Programming).
- Well-tested and well-documented programming code which does not contain major bugs (e.g. freeze or crash, unpredictable or incorrect output, unintentional or inoperative functionality, etc.).
- Implementation of an appropriate (numerical) algorithm.

2. Additional marks for (weight* from 1-4; 4 being the highest):

- Advanced and user-friendly UI (2-3)
- Multi-threaded or parallel implementation (2-4)
- Use of plugins or DLLs (2-4)
- Generic and flexible architecture (i.e. easy to customise or extend; multi-platform without using a specific tool, etc.) (3-4)
- Programming code optimisation in terms of speed and memory usage. (2-3)
- Advanced software development methods including correct implementation of design patterns, the use of templates or generics, etc. (2-3)
- Evidence of good project management (task distribution, time management, communication) (1-3)
- Use of multiple languages or multiple language paradigms. (4)
- Other (1-4)

A ‘weight’ does not convert directly into an additional mark!

3. Marks are deducted for:

- Flawed architecture, e.g. not flexible, hard-coded, etc.
- Specifications are not met or are incorrectly implemented.
- Major Bug in a main component of the software (with or without crash).
- Poor project management including time management, task distribution and communication.

4. Guidelines for marks:

< 50 A software product that does not have the essential features as outlined in Section 1 and/or which has substantial flaws as outlined in Section 3.

50-60 A software product that just about meets the essential specs (Section 1) but with little or no additional features as outlined in Section 2 but has no flaws as outlined in Section 3.

60-70 A software product that fully meets the essential specs (Section 1) and may have well-implemented lower-weighted additional features as outlined in Section 2 and has no flaws as outlined in Section 3.

70-85 A solid software product that meets the essential specs (Section 1) and has well-implemented higher weighted additional features as outlined in Section 2 and has no flaws as outlined in Section 3.

85+ A stellar software product with all essential features (Section 1) no flaws (Section 3) and a substantial number of additional high weighted features as outlined in Section 2 and preferably additional well-implemented features not specifically mentioned in Section 2.