An Object-Oriented *Education Establishment Game (EEG)*

Group Project 2015

Regard the following text as your customer's requirement statement. Your customer wants a good end-product, but also one that, in software terms, is well documented and well tested.

The aim of the 2015 group project is to create an object-oriented software implementation of an *Education Establishment Game (EEG)*. The game will have something of the flavour of the well-known board game *Monopoly* from Hasbro/Parker. However, your game, while it should take its inspiration from *Monopoly*, will definitely not be a straight copy of the original. For one thing – and this is very important – it will be thematically different! Choose premises, locations, incidents and opportunities that are related to university. Give different groups of premises the characteristics of the places with which you are familiar (places can include teaching buildings, labs, halls of residence, restaurants, etc.). The players could be fictional Managers or Subject Leaders, with ambitions of expanding their portfolios on and around campus! The look and feel can be very different from *Monopoly*: you may have your own unique types of squares, and you may have some brand-new gimmicks that are unlike anything in a shop-bought board game: acquiring several groups of premises might give a player new powers – for example, the power to rename and set new usage fees for the combined group. Such additions will attract extra marks – up to ten of them – awarded at the marker's discretion (and his or her decision will be final) for enhanced playability, entertainment value, or technical sophistication!

The Education Establishment Game must be developed in Java. Eclipse should be used predominantly as the development environment. SVN should be used to facilitate versioning and collaboration between team members.

The end product is only part of what the Group Project involves. The software engineering process – and you will also be given credit for this – includes the documented analysis of the problem, the documented design of the solution, the documented testing of the implemented software, the manner in which it has been maintained and built, the manner in which team working has been facilitated, as well as, of course, as the quality of the resulting software system itself. While the finished product should work as intended, and show something of the complexity of an actual board game, adherence to process is also important in a serious software engineering exercise.

How complex does the game have to be? The game should include at least the features mentioned between the broken lines below (▪▪▪▪▪▪▪▪▪).

Even a relatively simple *Education Establishment Game* is likely to be underpinned by quite a complex object mechanism, a suite of interacting software objects that have attributes and

behaviour. *Players* may take turns, roll dice, and land on squares. Depending on the square they land on, they may have obligations and opportunities. Players may have the opportunity to acquire squares that represent premises and may eventually acquire a whole group of premises. Once a group has been acquired, the player may wish to add new facilities to one or more of the premises in the group. Once several additional facilities are on a square, the square may be converted into a Centre of Excellence. Acquiring premises and adding facilities costs money. However, the better the facilities on a square, the more it costs for other players to use that square! Prices will differ from square to square. As players pay to acquire or to use a square or facility, the amount of money they have diminishes. Players' funds are topped up when they pass *Go*, or when other players pay them a usage fee. Certain squares (other than premises) may trigger events that have an associated action, cost or income. According to the 'house rules' you decide upon, the end of the game may be signalled when players concede defeat, or when one or more have no more funds, or when a time limit is reached, or simply when the players decide to stop. The system should be able to provide a summary of the final state of play (and indeed throughout the game it should remind players of the opportunities available to them, what they have to pay, what income they have received, and the state of their finances generally).

The 'marked-off' features between the broken lines above (▪▪▪▪▪▪▪▪▪) represent a basic *Education Establishment Game*. For a basic edition, the board may not be as extensive as a real-life board game like *Monopoly*. The look and feel may be much simpler. (A sample Java/Swing-based *Property Trading Game* is available as an executable jar file for you to look at and try: it shows how some of the very basic features of a board game can be implemented as a simple GUI-based application. Of course, you'll probably want to try for more involving gameplay and a more sophisticated look-and-feel in your *Education Establishment Game*.)

Value-added features might include: excellent playability, even with large numbers of players; many and varied additional premises, facilities, and 'surprise' costs and incomes; a very original interpretation of the university theme; an exciting reinterpretation of board game features for the computer; suitable celebrations when a player wins. You may want to look at the rules of some of the many *Monopoly* variants that are now available to get ideas for additional features to add to your *Education Establishment Game*. You may have some great original ideas of your own that make the game more exciting or entertaining, or exploit opportunities that software gives you over cardboard and plastic!

That said, a system that implements the 'basic' marked-off features above is likely to involve quite complex object interaction (enough for a worthwhile software engineering exercise), assuming that the developers have attempted to identify the most obvious candidate objects and object behaviours in the problem domain.

... So, your challenge as a team will be to implement your interpretation of an *Education Establishment Game*. The very best versions will give more than the basics outlined between the

broken lines – but be careful not to overstretch your abilities as a team. Again, the basic version will already present you with many design and implementational challenges.

What you have to provide…

By week 6 you must produce an *initial report*. Your team should produce a use case analysis, a software analysis of the main functionality of your proposed solution, and a development plan. Your proposal may suggest additional functionality (e.g. some of the functionality described outside the broken lines above) that will lend added value to your project.

By week 11 you must have finished *the full working system and its accompanying documentation*. This will be assessed by the quality of the design presented in *a final design document*, by the thoroughness of *a completed and documented test plan*, and, of course, by the extent to which the full working system is *robust*, *does what it was designed to do*, and *incorporates value-added features*.

For the *initial report*, **20 marks** will be available.

Of the **40 marks** available for *the full working system and its accompanying documentation*, **10** will be awarded to the group for value-added features, and **10** for adherence to process (see the Deliverables document).

As good preparation for 'adherence to process', you should consider carefully the management structure of your team from the outset of the project. In a team of ten, a possible structure would be: a manager and a deputy manager, 2 analysts, 2 software designers, 2 programmers, and 2 test engineers. These roles should not be tightly 'ring-fenced' (an analyst might suggest a high-level software design, a designer might implement a high-level coded solution, etc.), but the roles are intended to set particular individuals' focus and give them particular responsibility within a team. All developers should be engaged throughout the development process. At times the manager may ask an individual to take the lead in a task, or may request that an individual switch roles in support of another team member. Allocation of roles should help individuals work to their strengths within an overall software engineering process. Once the team decides who will manage it, the manager should ensure that the necessary development roles are filled. Development will be iterative in nature so that all team members have an opportunity to contribute to regular working prototypes and to the project documentation. Even as they prepare the Week 6 deliverable, teams should aim to create exploratory prototypes that inform their analysis and design. Weekly team minutes (circulated to and agreed by the team members) should show what tasks have been allocated to each team member and what contributions have been made by each team member. Teaching associates and student advisers will be available at the weekly advisory sessions to provide technical and product advice.

Full details of what you have to produce during the Software Engineering and Group Project can be found in *CSC2018 Spring 2015 Deliverables***.**