School of Computer Science – Coursework Issue Sheet

Session	16-17	**Semester**	Autumn
Module Name	Software Maintenance	**Code**	G52SWM
Module Convenor(s) **(CW Convenor in Bold)**	Andrew French, **Peer-Olaf Siebers**		

Coursework Name	Understanding and improving other people's software	**Weight**	25%
Deliverable (a brief description of what is to be handed-in; e.g. 'software', 'report', 'presentation', etc.)	1000 word report reflecting on the process of understanding third party software including relevant examples, class diagrams and explanations.		
Format (summary of the technical format of deliverable, e.g. "C source code as zip file", "pdf file, 2000 word max", "ppt file, 10 slides max", etc.)	PDF file, **1000 words max**. Word count on front page. This should be in the form of a number of screenshots of problem code (5 examples) along with a brief description of any coding problem, and how you would change it. There should also be a class diagram and explanation (see below)		

Issue Date	Issued w/b 17th October
Submission Date	Deadline Weds 2nd Nov at 3pm. Standard penalty applies for late submissions. If you have ECs please contact Peer or Andy.
Submission Mechanism	Electronic submission of PDF via Moodle
Late Policy **(University of Nottingham default will apply, if blank)**	Standard policy
Feedback Date	w/b 21st November
Feedback Mechanism	Individual comments via grading system on Moodle. General feedback in lecture.

Instructions	You will be given the source code of **Diamond Hunter, but this time some changes have been made, and they are not all good.** Use this code for this coursework (Called DiamondHunterCW1.zip). Your task is to find and read some material (we will give you an article to start with but you should find some more material by yourself online) on the topic of "**Making Bad Code Good Code**" and then apply what you learned from your studies to portions of the given code. Below we provide you with some pointers to classes which we think might be good targets for improvement but in the end your examples can come from **anywhere** in the code. The expected output of the coursework is a report (1000 words max.) where you describe how you would improve the provided code, based on what you have learnt about good programming practices and object oriented software

development to date. This should be in the form of a number of screenshots of problem code (covering **5** significantly different types of problems) along with a brief description of any coding problem, and how you would change it to overcome these problems.

Also provide a high level class diagram from parts of the source code provided (it is enough to consider the following game packages: GameState, Main, and Manager). Doing this and providing an explanation of how you derived it from the source code (e.g. how you identified the type of relationship between classes) will have a positive impact on your mark.

Some tips:

- Only providing a class diagram without explanations of how you derived it is not sufficient**. The class diagram only needs to be high level** – i.e. providing class names and relationship but not listing all fields and methods. In this case, the diagram should be **manually derived**, rather than using software to interpret the code.

- When you pick examples you always need to provide an explanation of why your alteration is an improvement to the project. Just providing examples is not enough. Picking the same kind of examples will not bring you any benefits. You need to look out for different kinds of examples (tackling different types of problems with the source code)

- In Workshop 2 we provided examples different categories of maintenance: Basic Maintenance and More Advanced Maintenance. If you want to get high marks you should consider examples from both categories. The more of your examples are "More Advanced" refactoring examples, the better.

You should start by reading the article "Make Bad Code Good" (see link below). If you want to have a better grade you should also consider some ideas from additional sources (on the internet).

http://www.javaworld.com/article/2075129/testing-debugging/make-bad-code-good.html

Good starting points for your investigations would be:

- *com.neet.DiamondHunter.TileMap/TileMap.java*
- *com.need.DiamondHunter.Manager/Keys.java*
- *com.neet.DiamondHunter.Entity/Player.java*

IMPORTANT: Make sure you understand what you are writing in your essay. We reserve the right to briefly interview you if we think that you do not understand what you write about. So, when you write your essay do not simply copy/paste large portions of text from existing articles or other resources – as this does not demonstrate your understanding of the topic. You might also run into issues with plagiarism.

If you use information from other resources please remember to **provide a reference** to the website / book you found the information in. A quick guide for correct referencing can be found here:
https://www.uhi.ac.uk/en/libraries/how-to/UHI-mini-Student-referencing-guide-en.pdf

	*A link to the game code of **Diamond Hunter CW1** will appear on **Moodle** on Wednesday **19th October**.*
Assessment Criteria	Quality and challenge of the code improvements will be assessed. Evidence of further reading and investigation of the code will be particularly rewarded. MARKING: A **pass** mark requires: • Evidence that you read the supplied article and that you have an understanding of some elements of good and bad coding practice A **high** mark requires • Evidence that you read and understood the article, and have done some **further reading** • Provision of a wide range of **insightful** examples from the supplied code, related to recognised concepts, and suggesting improvements • Clear details of **why the examples could be improved**, including citations to source material • Provision of a high level class diagram (including a clear explanation of how it was derived) • Demonstration of an understanding of the more complex aspects of OO design improvements will achieve the highest marks • Clear, justified examples and clear descriptions of your improvements to the code.