University of Derby
Department of Electronics,
Computing and Mathematics

Assignment Specification

Module Code and Title: 4CC511 Programming 2	
Assignment No. and Title: Critterworld	
Assessment Tutors: Dave Voorhis	**Weighting Towards Module Grade:** 100%
Date Set: Friday, 22nd February, 2019	**Hand-In Deadline Date:** May 3th, 2019, 4.00pm for final submission on Course Resources; 3.00pm for DLL submission to the Critterworld competition.

Conditions for Late Submission

Recognising that deadlines are an integral part of professional workplace practice; the University expects students to meet all agreed deadlines for submission of assessments. However, the University acknowledges that there may be circumstances that prevent students from meeting deadlines. There are now 3 distinct processes in place to deal with differing student circumstances:

1. Assessed Extended Deadline (AED): Students with disabilities or long-term health issues are entitled to a Support Plan.
2. Exceptional Extenuating Circumstances (EEC): The EEC policy applies to situations where serious, unforeseen circumstances prevent the student from completing the assignment on time or to the normal standard. http://www.derby.ac.uk/eec
3. Late Submission: Requests for late submission will be made to the relevant Subject Manager in the School (or Head of Joint Honours for joint honours students) who can authorise an extension of up to a maximum of one week.

Work that is submitted late without being covered under one or more of the above processes to deal with student circumstances will be awarded a failing grade of 0%. You must submit your work; otherwise, a grade of NS (non-submission) will be recorded. Under the University regulations, a non-submission may be considered evidence of non-engagement, which may lead to your being removed from the course.

Level of Collaboration

This is an individual assignment. No collaboration with other students or anyone else is allowed.

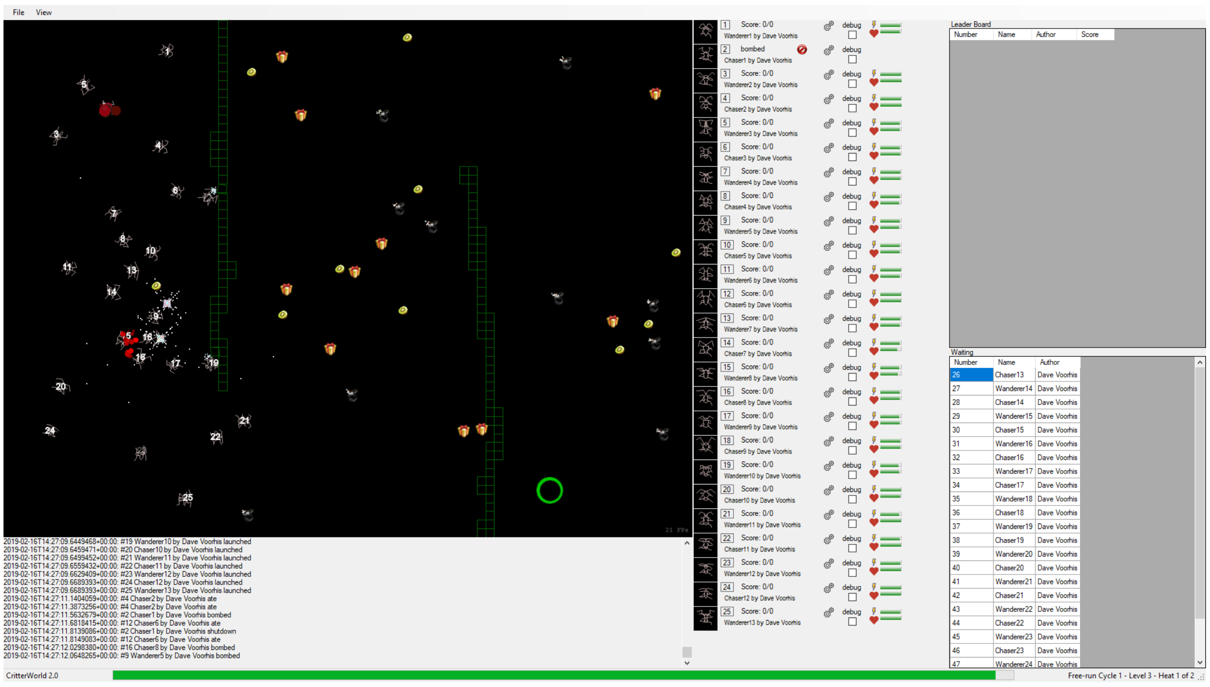
Learning Outcomes covered in this Assignment:

1. Design, develop, and test computer applications of moderate complexity, using a professional approach.

2. Demonstrate the ability to implement software that meets the requirements described in a specifications document.

It's a Bug's Life…

Critterworld is a simulation of a habitat of small cybernetic creatures – loosely based on insects – called Critters. They run around in mazes, eat food to gain energy and lose energy when they move, earn points by picking up prizes, and sometimes fight with each other. Their main goal is to escape from the maze before a timer runs out, because if the timer runs out before escaping, all points earned on the current level are lost.



```
2019-02-16T14:27:09.6449468+00:00: #19 Wanderer10 by Dave Voorhis launched
2019-02-16T14:27:09.6459471+00:00: #20 Chaser10 by Dave Voorhis launched
2019-02-16T14:27:09.6499452+00:00: #21 Wanderer11 by Dave Voorhis launched
2019-02-16T14:27:09.6559432+00:00: #22 Chaser11 by Dave Voorhis launched
2019-02-16T14:27:09.6629409+00:00: #23 Wanderer12 by Dave Voorhis launched
2019-02-16T14:27:09.6689393+00:00: #24 Chaser12 by Dave Voorhis launched
2019-02-16T14:27:09.6689393+00:00: #25 Wanderer13 by Dave Voorhis launched
2019-02-16T14:27:11.1404059+00:00: #4 Chaser2 by Dave Voorhis ate
2019-02-16T14:27:11.3873256+00:00: #4 Chaser2 by Dave Voorhis ate
2019-02-16T14:27:11.5632679+00:00: #2 Chaser1 by Dave Voorhis bombed
2019-02-16T14:27:11.6818415+00:00: #12 Chaser6 by Dave Voorhis ate
2019-02-16T14:27:11.8139086+00:00: #2 Chaser1 by Dave Voorhis shutdown
2019-02-16T14:27:11.8149083+00:00: #12 Chaser6 by Dave Voorhis ate
2019-02-16T14:27:12.0298380+00:00: #16 Chaser8 by Dave Voorhis bombed
2019-02-16T14:27:12.0648265+00:00: #9 Wanderer5 by Dave Voorhis bombed
```

1	Score: 0/0 Wanderer1 by Dave Voorhis	debug ☐	
2	bombed Chaser1 by Dave Voorhis	debug ☐	
3	Score: 0/0 Wanderer2 by Dave Voorhis	debug ☐	
4	Score: 0/0 Chaser2 by Dave Voorhis	debug ☐	
5	Score: 0/0 Wanderer3 by Dave Voorhis	debug ☐	
6	Score: 0/0 Chaser3 by Dave Voorhis	debug ☐	
7	Score: 0/0 Wanderer4 by Dave Voorhis	debug ☐	
8	Score: 0/0 Chaser4 by Dave Voorhis	debug ☐	
9	Score: 0/0 Wanderer5 by Dave Voorhis	debug ☐	
10	Score: 0/0 Chaser5 by Dave Voorhis	debug ☐	
11	Score: 0/0 Wanderer6 by Dave Voorhis	debug ☐	
12	Score: 0/0 Chaser6 by Dave Voorhis	debug ☐	
13	Score: 0/0 Wanderer7 by Dave Voorhis	debug ☐	
14	Score: 0/0 Chaser7 by Dave Voorhis	debug ☐	
15	Score: 0/0 Wanderer8 by Dave Voorhis	debug ☐	
16	Score: 0/0 Chaser8 by Dave Voorhis	debug ☐	
17	Score: 0/0 Wanderer9 by Dave Voorhis	debug ☐	
18	Score: 0/0 Chaser9 by Dave Voorhis	debug ☐	
19	Score: 0/0 Wanderer10 by Dave Voorhis	debug ☐	
20	Score: 0/0 Chaser10 by Dave Voorhis	debug ☐	
21	Score: 0/0 Wanderer11 by Dave Voorhis	debug ☐	
22	Score: 0/0 Chaser11 by Dave Voorhis	debug ☐	
23	Score: 0/0 Wanderer12 by Dave Voorhis	debug ☐	
24	Score: 0/0 Chaser12 by Dave Voorhis	debug ☐	
25	Score: 0/0 Wanderer13 by Dave Voorhis	debug ☐	

Leader Board

Number	Name	Author	Score

Waiting

Number	Name	Author
26	Chaser13	Dave Voorhis
27	Wanderer14	Dave Voorhis
28	Chaser14	Dave Voorhis
29	Wanderer15	Dave Voorhis
30	Chaser15	Dave Voorhis
31	Wanderer16	Dave Voorhis
32	Chaser16	Dave Voorhis
33	Wanderer17	Dave Voorhis
34	Chaser17	Dave Voorhis
35	Wanderer18	Dave Voorhis
36	Chaser18	Dave Voorhis
37	Wanderer19	Dave Voorhis
38	Chaser19	Dave Voorhis
39	Wanderer20	Dave Voorhis
40	Chaser20	Dave Voorhis
41	Wanderer21	Dave Voorhis
42	Chaser21	Dave Voorhis
43	Wanderer22	Dave Voorhis
44	Chaser22	Dave Voorhis
45	Wanderer23	Dave Voorhis
46	Chaser23	Dave Voorhis
47	Wanderer24	Dave Voorhis

CritterWorld 2.0 Free-run Cycle 1 - Level 3 - Heat 1 of 2

Your job is to write the artificially intelligent "brains" or controllers for several Critters.

This is done by writing plug-ins for the Critterworld software, following the specification of requirements linked from this document.

Our plan is to run a fun competition in the final week of classes to let your Critter controllers compete with other students' Critter controllers. Sorry, there will be no prizes or awards; only the unending admiration of your colleagues should you win!

Whilst this might seem like a somewhat trivial game, such simulations have a long and respected history. A similar idea first appeared in the late 1970's on the Apple II as "Robot Wars," coded by the famous game developer Silas Warner, who also wrote the original, 2D, view-from-above version of "Castle Wolfenstein."[1] Since the original, the idea has been resurrected numerous times and is now considered a classic – though perhaps somewhat under-recognised and under-appreciated – subgenre of computer gaming.

[1] The original Castle Wolfenstein was the inspiration for Castle Wolfenstein 3D, which led to Doom, which arguably led to the entire genre of "first-person shooter" style games.

It also has serious aspects: Similar simulations were the motivation for the first implementation of object-oriented programming in the late 1960s and are closely related to current research in artificial intelligence.[2]

Learning Outcomes

In addition to the explicit learning outcomes listed above, this assignment has other valuable goals that are part of helping you to become a better programmer: It's intended to make you read code – which is a fundamental part of being able to write code professionally, even if you're not a programmer *per se* – and it's designed to give you experience coding what are essentially software plug-ins, which are used in many modern software systems.

Also, whilst this is superficially a game, what you're *really* doing is writing software to meet specified requirements, that must meet minimum expectations of quality – both in terms of object-oriented design and run-time robustness – and that requires you to process a continuous stream of dynamic data.

In short, it requires you to demonstrate skills that apply to almost every kind of professional programming that you might eventually have to do.

…And it's meant to be fun! 😊

Task

The full Critterworld environment is available to you, as open source, at https://github.com/DaveVoorhis/Critterworld

There are three possible ways to get onto your own machine, in order from best to less-best. Pick one:

1. Fork it to your own account on GitHub. Use the "Clone or download" button on GitHub to copy the forked Git project URL, then in Visual Studio use the Clone option under the Team Explorer to clone the forked project.
2. Use the "Clone or download" button on GitHub to copy the Git project URL, then in Visual Studio use the Clone option under the Team Explorer to clone the project.
3. Use the "Clone or download" button on GitHub to download a .ZIP file of the project, then unzip the .zip file locally. *Note that you must unzip the .zip file. If you try to open the project without unzipping the .zip file, it will not load correctly.*

Open the project by double-clicking on the Critterworld2.sln file. Then right-click on the Critterworld subproject and select "Set as StartUp Project".

The Critterworld subproject contains all the logic required to handle the bodies of Critters, but to run, Critters also need controllers, which are the "brains" of Critters. Without a critter controller or "brain", a Critter can't even be loaded.

[2] Stylistically, Critterworld is a mash-up homage to classic vector-based arcade consoles of the 1980's, mixed with 1990's PC videogame bitmap graphics and early 2000's business software, in a joyous hodgepodge of aesthetic touches alluding to the author's favourite videogames from his youth without actually revealing what any of them are.

Critterworld dynamically loads critter controllers for Critters from dynamic link libraries (DLLs) at run-time and creates a Critter for each controller that it successfully loads.

Your job is to write at least three new Critter controllers. You can base your Critter controllers on the two examples provided in the DemonstrationCritters subproject, but to achieve a minimum pass grade you must:

- Create at least three distinct Critter controllers, the requirements for which are described in detail below and in a separate document linked below. You may provide more than three if you wish, but during the competition a maximum of five Critters will be loaded and run. Your controllers may (and almost certainly should) share functionality, but they must also be defined distinctly and make Critters controlled by them behave differently from each other.
- Your Critter controllers must be different from the provided examples. **Submitting a Critter controller that is merely a copy of one of the demo controllers – even if variable and method names are changed – will result in a failing grade for this assignment.**
- A Critter controlled by at least one of your Critter controllers must appear in the Critterworld arena (the area on the upper left with the dark background) and identify a name and correct author in the relevant lists. This means you must implement the required classes correctly and the project must emit a .DLL containing your Critter controllers.
- Ensure that every Critter controller you create is configurable via its own pop-up form with options specific to each type of Critter controller, and support saving and loading configuration settings to a file, as illustrated by one of the example Critter controllers. **If the pop-up form and/or save/load facilities are absent from one or more Critter controllers, your grade will be capped at 40%.**

Detailed specifications of the mechanisms you need to provide are provided in the "*Specification of Requirements for a Critter Controller*" document, which is available at https://github.com/DaveVoorhis/CritterWorld/tree/master/Documentation or can be found (in both .docx and .pdf format) in the Documentation folder of the Critterworld project. **You must read this document; you won't be able to successfully pass this assignment without reading it carefully. Note that warnings about "serious penalties" must be heeded or you will receive a failing grade!**

The aim is for your Critter to reach a specified destination (shown with a green circle) or "escape hatch" before the level timer (shown as a green progress bar at the bottom of the Critterworld window) runs out, whilst earning points by collecting prizes which are shown as gift packages in the Critterworld arena. If the timer runs out before your Critter passes through the escape hatch for a given level, you will lose all the points earned on that level, but you keep the points earned on earlier levels.

The top of the leader board will be occupied by the Critter controllers that earn the most points. On early levels, it's easy to gobble up points and head for the exit. On later levels, it's not so easy.

To receive *more* than just a pass grade, Critters controlled by Critter controllers written by you must:

- Interact in the arena in a manner defined by you. That doesn't mean you necessarily have to earn maximum points and head for the escape hatch. Full marks may be awarded for Critters that do interesting things but perhaps don't earn many (or any) points.
- Go significantly beyond what the provided demonstration code does, whilst reducing code duplication and improving readability and usability using sound object-oriented approaches.
- Be "good citizens" of their environment, and not unreasonably consume resources, either CPU or memory.

Note that the provided demonstration Critter controller code is, quite simply, awful. It's not object-oriented, it's full of duplication, and it doesn't work very well. You can use it as a starting point, but you must significantly improve on it.

Therefore, to achieve an excellent grade on this assignment, you should:

- Create an object-oriented framework for developing Critter controllers that fully hides the currently-exposed complexity (and fragility) of sending and receiving string-based messages to and from the Critter body.
- Make appropriate use of object-oriented facilities like inheritance and/or composition to reduce duplication; encapsulation to manage complexity; and perhaps polymorphism to define dynamic instance behaviour, but only if appropriate.
- Make a serious attempt to create reliable, robust and interesting (which could include, but isn't limited to, competitive) Critter controllers using the above framework and embodying the above principles.

The behaviour exhibited by your Critter controllers is entirely up to you. We are looking for imagination and thought applied to how your Critter behaves, and there are marks available for interesting attempts using clever techniques.

However, **all behaviour must be implemented via, and only via, the APIs provided in the CritterController subproject. "Hacking" the Critterworld environment – for example by overwriting files or using .NET reflection – is expressly forbidden no matter how cute or clever and any such attempts, either handed-in via the submission point or run in the final-week competition, will result in a failing grade for the submitter.**

Read this assignment specification very carefully! It contains all the information you need. Note that to get a Critter running does not take a lot of code – in fact, you can implement a simple Critter controller that simply stands still on the screen in about 20 lines of code. This assessment is testing your ability to read code, read and understand a specification, and implement code based on that specification.

Running Critterworld

To see Critterworld in action, load the solution – as described above – into Visual Studio and build it. If the Critterworld subproject is not highlighted in bold, right click on it and select "Set as StartUp Project."

Now you can run it.

Once you've launched Critterworld, select *File | Start running* from the main menu to launch the provided Critter controllers in Critter bodies in your local copy of the competition arena.

In this mode, Critter controllers will be reloaded after each level, and when the last level finishes it will start over at the first level.[3] This is good for testing your Critter controllers.

Alternatively, you can select *File | Start running a competition* to simulate running the competition. Critter controllers will be reloaded once at the start of the competition and running will stop after the last level. The leader-board will show the winners.

Normally, 25 or fewer Critters will run at a time. Multiple heats will run to ensure that all Critters have a chance to run on every level.

Two kinds of demonstration Critters have been supplied for you to use as a model, and for your own Critters to interact with. Food (kiwi fruit) will be displayed in the arena for your Critter to eat. Food gives your Critter energy and increases diminished health. Gifts will be displayed for your Critter to collect and earn points. In some levels, bombs will appear that can kill your Critter if your Critter runs into them. Green barriers are electrified; they stop your Critter and reduce your Critter's health. If you bump into another Critter, you're "fighting" with that critter, which will reduce your Critter's (and your opponent's) health. When your Critter moves, it loses energy. When a Critter eats a piece of food or picks up a gift, the food or gift will vanish, and a new piece of food or gift will be created somewhere else in the world.

If the energy or health of your Critter goes below zero, your Critter will die. If your Critter doesn't exit through the escape hatch (green circle) before the level timer runs out (green progress bar at the bottom of the window), it will lose all points earned on that level.

If you are having problems, look in the log window below the arena. It displays diagnostic messages that might be helpful. You can click on the "debug" checkbox for a specific Critter to display more diagnostic messages but use it sparingly and briefly – it can significantly slow down the environment.

Note that *all* the source code for the Critterworld project is supplied. Nothing is hidden.

We **do not recommend editing source code in the main Critterworld project**; you should run it to test your Critter controllers but not change it. This is because any changes you make to the main Critterworld project will only apply to your copy. During competition, we will run your Critter controllers with our copy of Critterworld, not yours.

However, if you do find bugs in Critterworld or if you add features, you're free to send us a Pull request but we reserve the right to ignore it. Working with buggy and incomplete systems are unavoidable aspects of developing and using real software!

Critterworld Competition

Although the assignment does not need to be submitted until the Friday deadline noted above, you are expected to provide the dynamic link library (DLL) containing your Critter controllers for competition on or shortly after Monday of the same week (teaching Week 12). This DLL will be loaded into a central Critterworld simulation using (possibly) different terrain from that provided on GitHub, possibly different configuration settings from the defaults, and it will be run against Critter controllers provided by other students. Information

[3] As of this writing, there are some memory leaks that may cause the environment to slow down and eventually crash after running through all the levels some number of times – normally after it's run for several hours. I'm working on a fix, hopefully to be available shortly.

about how the DLL is to be delivered and the naming convention to be used will be announced later via Course Resources.

Failure to submit a DLL for execution during the final week of this module will result in failure of this assignment. DLLs can be delivered at any point up to 3pm on the deadline date. Any deliveries of DLLs after this time will not be accepted.

If you have an official support plan that provides an automatic extension to deadlines and you want your Critter controllers to take part in the competition, then you must still deliver controllers by 3pm on the deadline date. However, if you are unable to do so, **you will not be penalised for not submitting your Critter controllers to the competition if you submit your .zip file – as described below – within the terms of your support plan extension.**

Submission

You are to submit your copy of the entire Visual Studio Critterworld project – contained in a .ZIP file named xxxxxxxxx.ZIP, where xxxxxxxxx is your student number – that hosts your Critter controllers in an appropriate subproject. It must be submitted on or before the deadline noted above, possibly considering support plans as appropriate. A submission point will be provided prior to the deadline on Course Resources.

In addition, you must submit your Critter controller .DLL to the competition, as described above, unless you have an automatic extension under the terms of your support plan. Instructions for doing this will be provided closer to the competition date.

Do not submit the demonstration Critter controllers along with your own!

Delete the demonstration Critter controllers before submission. This is to make sure the marker does not think you're trying to submit one or more demonstration Critter controllers as your own.

Plagiarism/Collaboration Warning

All submissions will be checked to see if there is any evidence of collusion with other students or contract cheating. If we suspect that you have submitted work that is not your own, we may ask you to attend a meeting to explain your work to us so that we can ensure that it is your own work.

Work on this assignment on your own. **Do not** ask to see the work of other students; do not ask for help on the Internet and do not share your work with other students. The penalties for plagiarism or collaboration are severe!

Grading Scale

The following grading scale is intended to give you a general understanding of how this assignment will be marked:

% mark	Mark descriptors	Class
70-100%	**Excellent** *90-100%* Exceptional work. An impressive attempt has been made to provide innovative approaches to at least three Critter controllers, perhaps using recognised or new AI techniques. All code is written, formatted, and commented to a professional object-oriented standard, and is efficient and highly readable. Language constructs are used appropriately and with clear justification. Comments and/or code indicates evidence of extensive testing and validation. Configuration forms are polished and professional. Flaws, if any, are insignificant. *80-89%* Outstanding work. A notable attempt has been made to provide innovative approaches to at least three Critter controllers, perhaps using recognised or new AI techniques. All code is written, formatted, and commented to a professional object-oriented standard, and is efficient and highly readable. Language constructs are used appropriately and with clear justification. Comments and/or code indicates evidence of extensive testing and validation. Configuration forms and file save/load facilities are polished and professional. Flaws, if any, are minor and do not obstruct functionality. *70-79%* High to very high standard of work. A strong attempt has been made to provide innovative approaches to at least three Critter controllers. All code is written, formatted, and commented to a generally professional object-oriented standard, and is efficient and readable. Language constructs are generally used appropriately. Comments and/or code indicates some evidence of testing and validation. Configuration forms and file/save load facilities are generally professional. Flaws, if any, are relatively minor and do not significantly obstruct functionality.	First
60-69%	**Very good** A very good standard. An attempt has been made to provide innovative approaches to at least three Critter controllers. All code is written, formatted, and commented to a generally very good but perhaps not quite professional object-oriented standard, but is reasonably efficient and generally readable. Language constructs are mostly used appropriately. Comments and/or code indicates some evidence of testing and validation. Configuration forms and file save/load facilities are generally very good. There may be some flaws and obvious bugs, but that do not prevent the Critters from working.	Second Division 1

50-59%	**Good** A good standard. An attempt has been made to provide different approaches to at least three Critter controllers. All code is written, formatted, and commented to a good scholastic standard, but needs significant work to be considered professional. There may be minimal effort to use appropriate object-oriented constructs, or they may be used poorly. Comments and/or code might not provide evidence of testing and validation. Configuration forms and file save/load facilities are functional but flawed. There may be obvious bugs, but the Critters are essentially functional.	Second Division 2
40-49%	**Satisfactory** An adequate standard of work. There are major or obvious bugs, and/or the code quality is below even a reasonable scholastic standard. There are significant flaws, but at least there's clear evidence of producing three Critter controllers that are obviously distinct from the demonstration Critter controllers provided. Forms and file save/load facilities may be significantly flawed or very limited. **If the configuration forms and/or file save/load facilities are absent from one or more of your Critter controllers, your grade will be capped at 40%.**	Third
35-39%	**Unsatisfactory** A submission has been made but is essentially just the demonstration Critter controllers with insignificant changes (e.g., just name and creator, or just some method and variable names) or no changes, or is fewer than three Critter controllers, or otherwise does not meet the stated requirements in this document. For example, no DLL is submitted for competition. **An automatic failing grade will be given for any Critter controller that reads or writes files anywhere other than the directory specified by the Filepath property of an ICritterController.**	Marginal Fail
21-34%	**Poor** The code can't compile without work or is largely dysfunctional.	Fail
1-20%	**Very poor** The code can't compile without significant work or is so dysfunctional as to be essentially irrelevant to the requirements of the exercise.	
NS	**Non-submission** No work has been submitted.	
Z	**Academic offence notation** Applies to proven instances of academic offence.	

For further information, see http://www.derby.ac.uk/academic-regulations