IN1007 Java Programming Project: Formal Details

Assessment and Module Weighting

This project is worth 30% of the total marks for the module. A minimum mark of 40% in the Programming Project is necessary for students progressing to Part 2 of any of the Computer Science or Software Engineering routes (does not apply to Business Computing Systems).

Individual Work

Pair/group working is NOT allowed for the Programming Project. The work you submit must be your own work only. Collaborative submissions will be given zero marks. Claiming copied or collaborative work as your own individual work is regarded as plagiarism and is subject to serious disciplinary action.

Submission and Assessed Deliverables

This project is assessed in stages. There are THREE assessed deliverables.

Task	Moodle Submission Deadline	Weight
Milestone 1	Sun, 18th February, 16:00	30%
Milestone 2	Sun, 18th March, 16:00	30%
Milestone 3	Sun, 8th April, 16:00	40%

The first two milestones need to be submitted by the deadlines listed above and will be marked in your presence during the labs following those deadlines (Thursdays). You will need to be there during those labs so you can demo your submission and have it assessed. Conversely, the last deliverable will be marked offline.

Note that it is a requirement that you use the latest version of the city.cs.engine library. Regardless of the files you submit, we will compile and run your source code against the latest stable version of the game engine libraries. If you write code that does not use the game engine libraries we provide, or code which relies on non-standard library hacks of your own, you will receive zero marks. Additional use of code libraries, other than the ones provided specifically for this project, is only permitted if you obtain written permission in advance from the lecturer.

Milestone Assessment

Assessment is broken down into three parts (Milestones 1, 2, 3). All three Milestones must be submitted via Moodle.

Submission

For each deliverable, you are required to submit a Zip archive of your project via Moodle by the deadline indicated above. The Zip archive must be in a format which is recognised and can be unpacked by the standard software installed in the university Windows labs. If in doubt try it in the labs before you submit.

If you fail to submit an archive in the correct format by the deadline, or submit a project that does not compile, your work cannot be assessed.

Applications for extensions to any deadline (based on valid extenuating circumstances) must be made using the usual Extenuating Circumstances procedure at the first available opportunity (forms available from Programmes Office and online).

What to submit (and what not to submit)

As your project advances, you will develop a relatively large number of classes. For this reason, it is not appropriate to submit separate source files. Instead, for each submission, you must create a Zip archive containing your entire NetBeans project folder. If you do not know how to do this, you should ask for help in the labs now.

Note that, regardless of the files you submit, we will compile and run your source code against the latest stable version of the game engine libraries. If you write code which does not use the game engine libraries we provide, or code which relies on non-standard library hacks of your own, you will receive zero marks.

Please note that you must not include a copy of the engine library with your submission. It is not permitted to include any jar or other code library with your submission unless you have obtained prior written permission for its use from the lecturer.

With each milestone, include a short but clear written description of the features implemented.

Comments and style

You must provide clear and informative header comments for your classes and methods. Assessment of Milestone 3 takes into account the quality and completeness of your header comments and your choice of names for classes, fields and methods. Your header comments must be Javadoc comments so that API documentation can be generated automatically from your source code. Your Javadoc will be assessed in Milestone 3.

Requirements and Assessment Criteria

Specific features are assessed for each milestone; once a feature has been assessed it is permitted to remove it from your game, if you so wish. Marks will be allocated in line with the following criteria, corresponding to degree grades:

Mark range	Criteria
70-100 [1st class]	Work that meets all the requirements in full, constructed and presented to a professional standard. Showing evidence of independent reading, thinking and analysis.
60-69 [2(i)]	Work that makes a good attempt to address the requirements, realising all to some extent and most well. Well-structured and well presented.
50-59 [2(ii)]	Work that attempts to address requirements realising all to some extent and some well but perhaps also including irrelevant or underdeveloped material. Structure and presentation may not always be clear.
40-49 [3rd class]	Work that attempts to address the requirements but only realises them to some extent and may not include important elements or be completely accurate. Structure and presentation may lack clarity.
0-39 [fail]	Unsatisfactory work that does not adequately address the requirements. Structure and presentation may be confused or incoherent.

Detailed requirements for individual milestones are included below. You must refer to the weekly lab tasks, as these are to be considered a supplement to this assessment brief, and contain more detail and (if necessary) corrections.

Milestone 1

You must submit a modified version of the game that compiles, and contains the features below. In the assessment, all five features carry equal weight.

1. The game should contain instances of at least two new body classes (subclasses of DynamicBody or Walker), significantly different from each other and from those distributed. More than two new body classes (up to a maximum of four) will gain higher marks.

2. Instances of at least one of your new body classes should be rendered with images. Body shapes do not have to align exactly with images but they should align sufficiently well that the look and feel of the game is not compromised. Wider and/or more effective use of images to render bodies will gain higher marks.

3. At least one of your new body classes should have at least one additional attribute with appropriate accessor and mutator methods. Wider use of additional body attributes will gain higher marks.

4. The game should be controllable with either the keyboard or mouse (or both). More sophisticated controls will gain higher marks.

5. Your game should respond to certain collisions between bodies by changing the value of the additional attribute mentioned above. Add print statements to your collision handlers so that the changes can be seen to be happening. Wider use of collision handling will gain higher marks.

Note: To achieve full marks, in accordance with the assessment criteria, you need to "show evidence of independent reading, thinking and analysis". This means you need to somehow go beyond the base requirements. Opportunities for extension are listed above ("… will gain higher marks").

Milestone 2

You must submit a modified version of the game which compiles, and contains the following features.

1. At least three game levels implemented as subclasses of a common class. On achieving certain goals within the game, the player progresses to the next level. There should be significant differences between the levels.
   You can achieve higher marks if you: use some sort of sophisticated code (e.g., use of collections/data structures to manipulate levels; automated generation of levels); introduce some innovative game behavior; create significant differences between levels (e.g., different backgrounds, different bodies, different behaviours - walking in first level, shooting in second, invisibility); more than three levels.

2. A graphical interface that displays information (such as health, score, etc).
   You can achieve higher marks if you: use some sort of sophisticated code or special behaviour (e.g., transitioning player characteristics from level to level rather than resetting them to 0); use special graphics (e.g., progress bars, icons, or components that were not covered in lecture); display different properties in different levels.

3. Graphical User Interface controls (for example Pause and Restart buttons).
   You can achieve higher marks if you: use some sort of sophisticated code or special behaviour (e.g., custom made buttons, using different Layout Managers hierarchically); use components that were not covered in lecture (text boxes, drop downs); implement functionality other than pause/start/quit (e.g., jumping between levels).

4. At least four different kinds of dynamic body that interact with other bodies in varying ways. (These may include bodies presented for the first Milestone.)
   You can achieve higher marks if you: use some sort of sophisticated code or special behaviour (e.g., ghostly bodies or collision filtering – i.e., collision physics don't apply to some bodies;

storing bodies in collections, lists, etc.); introduce many types of bodies with different behaviour (e.g., angular rotation, they change size or the way they move dynamically); code different interactions between bodies (e.g., non-player objects interact with other non-player objects)

Note: To achieve full marks, in accordance with the assessment criteria, you need to "show evidence of independent reading, thinking and analysis". This means you need to somehow go beyond the base requirements. Examples of possible extension are listed above ("You can achieve higher marks") but you can also come up with your own inovtations.

Milestone 3

The final assessment requires at least two additional game features of your own choice. Below are a few examples of things you could implement but you can choose to be creative and come up with your own features. The more complex and diverse your two features are the higher your mark will be.

Sound: implement background sound that is the same for all levels (low mark) or changes between levels (high mark) or changes in response to game events and progression (high mark); add sound control mechanisms (e.g., changing volume, switching on/off, choosing which sounds to play); add sounds that play on events (collision and interaction) or that play on timers (e.g., music changes as time runs out); make sure the manage your sounds efficiently.

FSMs: create an FSM with many states or multiple FSM characters that are somehow different; play with states that are triggered by things other than proximity (e.g., being shot, timer, etc.); use generic FSM classes (high mark);

Shooting: add projectiles that cause some action when hitting a target and get destroyed on contact with other things (so they don't clutter the scene); implement directional control (shooting in different directions); anti-gravity (projectile moves straight rather than falling towards the ground); implement different types of projectiles and weaponry; use abstraction or inheritance (e.g., base class for projectiles);

Main menu: use interesting controls not covered in class (e.g., text box to add player name, sound level, drop down lists); use sub-menus with navigation (e.g., instructions, settings accessible from the main menu); apply some nice styling to controls that preserve a particular look and feel; implement menus that can be hidden.

Timers: implement one or more timers that do something (e.g., spawn a character, times the game); get extra points for multiple timers that do different things; create timers that change some sort of persistent state (e.g., decrementing a remaining time indicator, increases a characters stamina); have a pause button that stops a timer.

Game save and load: save your game state (e.g., player lives, score, collectables; which level) and allow user to reload it; implement the ability to save mid-level (high mark: involves

recording which items were picked up so the level can be recreated accurately); add the ability to select different save-files or saving under different player names.

Loading levels from files or level editors: consider the ability to load the configuration of your levels from a text file specification; implement the ability to create and edit levels interactively and save them to files (high marks).

Scrolling: add zooming or other camera positioning effects; implement perspective effects (parallax scrolling); scroll the scene with the player when the player is close to the borders of the screen.

High-scoring: record scores and show the highest (low mark) or show all or some of them sorted (high mark); consider improving the way in which you are displaying scores (e.g., scrollable list; preserving the game look and feel).

There are a few other things that we will mark at this stage:

- How polished/professional your game looks. Does your game make sense and does it demonstrate attention to detail?
- Whether you have created Javadoc for your game and how complete your Javadoc is (i.e., all tags should be specified and described).
- How well organized and structured your source is (e.g., properly structured in classes and packages, without code duplication, uses inheritance, etc.)

Finally:

Include with your submission a concise written description of the features, explaining the key coding challenges involved. The presence and quality of this write-up will be marked.

Include with your submission a video no longer than 1 minute showcasing the game play and the features implemented as part of the Milestone 3. Use screen-recording to capture the video, then overlay commentary or subtitles on top describing what is going on. The video should be accessible via a URL and should not require any special codecs to be viewed (e.g., upload it to YouTube). The presence and quality of the video will be marked.