IN1007 Java Programming Project **(50% of module mark)**

The project involves building a functional 2D Game using the City Engine Physics library. You will deliver the game in three deliverables by the deadlines listed below. Each deliverable with need to meet a set of concrete requirements (detailed below), will be submitted on Moodle and marked during a brief face-to-face chat. This should be a piece of individual work; collaborative work is regarded as plagiarism.

Task	Moodle Submission Deadline	Weight
Milestone 1	Sun, 7th February, 17:00	30%
Milestone 2	Sun, 7th March, 17:00	35%
Milestone 3	Sun, 28nd March, 17:00	35%

Submission and Marking

Each deliverable will be submitted on Moodle by the deadlines listed above (Sunday). They will then be marked in your presence by designated markers during the scheduled tutorial times following those deadlines (Tuesday). Please note that to receive a mark you will need to both submit the work on Moodle by the deadline and meet your assigned marker during the set time; we cannot accommodate custom meetings and it is not allowed to approach a different marker than the one assigned to mark your work. Submission and marking details are provided below.

Submission: Submit each deliverable on Moodle as a Zip archive of your IntelliJ project/game directory. 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 IntelliJ project folder. The Zip archive must be in a format which is recognised and can be unpacked by standard archiving software (e.g. 7Zip). If you do not know how to do this, you should ask for help in the labs as soon as you can. Please also note that the maximum file size you can submit is 200MB.

You are required to use the city.cs.engine library that is provided on Moodle and are not allowed to use other libraries (unless you obtain written permission from the lecturer in advance). To receive marks your unzipped project needs to run on our own machines without the need for additional configuration.

Please note that you don't need to include a copy of the engine library with your submission. Instead the engine library should be linked to your project as a global library as demonstrated during the first lecture. 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.

It is your responsibility that the project is received on time; we are not allowed to mark projects received past the deadline, even if they are only a few minutes late! Please account for the fact that uploading a project to Moodle can take a long time depending on the size of your project and the number of students trying to submit. Also account for unexpected circumstances (e.g., you file size is too large; your WIFI crashes). Submissions via email or other mediums after the deadline has passed are not acceptable.

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).

Marking: Submissions will be marked in your presence on the Tuesday following your submission. Concretely, the lecturer or a member of the TA team will be assigned to mark your work and will set up an online meeting to do so. During that meeting they will download and unzip the file that you have submitted on Moodle; they will ask questions about your game and its implementation; assign you a mark on the spot; and communicate it to you.

Markers will be assigned to you prior to assessment and may be different from your regular tutors. Also, different markers may mark different milestones to ensure that your work is assessed from different perspectives.

Assessment Criteria: Marks will be allocated in line with the criteria below, corresponding to degree grades. Specific features are required for each milestone as detailed in the following section. Once a feature has been assessed you can remove it from your game, if you so wish. You are even allowed to change the game fully between milestones though we would advise against that.

Mark range	Criteria
70-100 [1st class]	Work that meets all the requirements in full, constructed and presented to a professional standard. To achieve a full mark (100%) your work should show evidence of independent reading, thinking and analysis (i.e., go beyond the requirements).
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.

Milestone 1 (Requirements)

You must submit a preliminary game that compiles, and contains the features below. You may use any of the demos made available in the first three weeks as a starting point for your project and modify it as you see fit.

1. **[30%]** 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. 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. To gain higher marks consider introducing more types of bodies; adding them to the world using loops or in response to events; and making interesting use of images (e.g., animated GIFs; left/front/right images for your characters).

2. **[15%]** 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.

3. **[20%]** The game should be controllable with either the keyboard or mouse (or both). More sophisticated controls will gain higher marks.

4. **[25%]** 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.

5. **[10%]** Your project should evidence good coding practices such as use of encapsulation principles; following Java naming conventions; comments.

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 (Requirements)

You must submit a modified version of the game which compiles, and contains the features below. You may use any of the demos made available as a starting point for your project and modify it as you see fit.

1. **[25%]** 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 some differences between the levels (e.g., different backgrounds, different bodies, different behaviors). 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; more than three levels.
2. **[15%]** At least four different kinds of dynamic body that interact with other bodies in varying ways (bodies may include those presented for the first Milestone.) You can achieve higher marks if you: use some sort of sophisticated code or special behavior (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 behavior (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)

3. **[15%]** Use of sound. At the very least your game should have a background track, play a sound in response to an event (e.g., a collision or a user interaction), and load sounds efficiently. You can get higher marks if your background tracks change between levels; you play sounds in response to many in-game events; and/or can control your sounds via your GUI (see requirement 5).
4. **[15%]** A graphical interface that displays information (such as health, score, etc.). Display at least two pieces of information. You can achieve higher marks if you: use some sort of sophisticated code or special behavior (e.g., transitioning player characteristics from level to level rather than resetting them to 0); use special graphics (e.g., progress bars, icons or small images, or components that were not covered in lecture); display different properties in different levels.
5. **[20%]** 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 behavior (e.g., custom made buttons, lay out components manually rather than using the IntelliJ GUI editor; 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, sound control).
6. **[10%]** Your project evidences good coding practices such as following Java naming conventions; comprehensive commenting; encapsulation; exception handling.

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 innovations.

Milestone 3 (Requirements)

You must submit a fully functional game. The final assessment will focus on four aspects:

1. **[20%]** Implementation of game-state saving and loading. At the very least your implementation should have a GUI option for saving and loading, and should save/restore level and player scores and attributes. For full marks players should be able to save or load to and from different files selectable from the GUI and should be able to restart mid-level (i.e., store not just the level number but also the state within the level).
2. **[30%]** Overall game complexity**.** Your game should expand on requirements from previous milestones (e.g., more sounds, more interesting characters) and introduce new features. The number and/or level of sophistication of features you introduce will determined the mark. Appendix 1 exemplifies additional features you can add to your game to fulfill this requirement (but you are free to explore others too).

3. **[20%]** Overall game quality and game play. To get a satisfactory mark your game-play should make sense and the game should feel polished (e.g., game features should work well together, imagery and sounds should be polished and complement the functionality of the game, etc.).
4. **[15%]** Quality of code and code-documentation. We will look at how professional your code looks (e.g., appropriate encapsulation and division into packaged and classes; avoidance of duplication; proper indentation; appropriate naming, use of inheritance, use of collections, etc.). We will also look at how well your code is documented. Your code should contain inline comments throughout. For at least 3 of the more complex classes you should generate full Javadoc documentation (i.e., classes/methods/fields should be fully annotated with Javadoc tags and Javadoc should be generated and included in the submission archive).
5. **[15%]** Video and portfolio update. You should create a video no longer than 2 minutes. It should demonstrate the game play and the game's main features, and it should briefly mention the main challenges you encountered and lessons you've learn doing the project. Finally, you should add a link to the video along with a brief description to the portfolio page you started during Bootcamp.

Annex 1: Possible features for Milestone 3

Sound: You can expand on your implementation from Milestone 2 to: implement background sound that changes between levels or changes in response to game events and progression; add sounds that play on events (collision and interaction) or that play on timers (e.g., music changes as time runs out).

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.).

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).

Menus and other advanced GUIs: Consider using 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.

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).