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; any form of collaborative work is regarded as plagiarism.

Task	Moodle Submission Deadline	Weight
Milestone 1	Sun, March 13th, 17:00	30%
Milestone 2	Sun, April 10th, 17:00	30%
Milestone 3	Sun, May 8th, 17:00	40%

Submission and Marking

Each deliverable will be submitted on Moodle by the deadlines listed above (Sunday). They will then be marked in your presence – a viva - by designated markers in the week that follows. You will be notified of a date/timeslot for your viva a few days in advance; this will not clash with your other University commitments, and you are required to attend it.

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 single Zip archive of your entire IntelliJ project/game directory. Please do not submit separate/multiple source files. 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., 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 – a viva – in the week following your submission. Concretely, the lecturer or a member of the TA team will be assigned to mark your work and will communicate a date/timeslot during which to meet. 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. Submitting your work on Moodle but failing to attend the viva will result in a mark of 0.

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. You cannot choose yourself which marker to assess you and failure to meet your designated marker will result in a mark of 0.

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 the requirements in full and demonstrates excellent use and understanding of most` of the concepts covered by the module. Where relevant, it will show evidence of independent reading, thinking and analysis. It will be well-constructed, fully functional, and demonstrate a professional approach to academic practice.
60-69 [2(i)]	Work that meets all requirements to some extent and most well, and demonstrates a sound use and understanding of many concepts covered by the module. It will be well-structured, fully functional, and demonstrate good academic practice.
50-59 [2(ii)]	Work that meets all requirements to some extent and some well but perhaps also including irrelevant or underdeveloped material. Most work will be functional. Structure and presentation may not always be clear. Attempts to demonstrate academic practice will be evident.
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. Some work may not be fully functional. Structure and presentation

	may lack clarity and evidence of academic practice will be limited.
0-39 [fail]	Unsatisfactory work that does not adequately address many of the requirements. Much of the work may not be functional. Most structure and presentation may be confused or incoherent.

Milestone 1 (Requirements)

You must submit a brief document outlining your game concept as well as a preliminary game built by extending the GitHub repository made available in the first week of term. The game should compile, and contain the features listed below:

1. **[10%]** Describe your game concept in a pdf document of up to three pages. You may use rough sketches (ex. hand-drawn mockups), prospective artwork, and explanatory text to illustrate envisioned game play, features, and general game feel. Do so by using the requirements outlined in the *Project Brief* and class discussions (ex: from weeks two and three) to imagine a game that you can accomplish and is sufficiently rich to receive a high mark. View the game-concept document as an opportunity for you to think ahead towards an end-goal rather than a commitment: your final game may differ considerably from what you imagine at this stage and that would be perfectly fine. Name the document *GameConcept.pdf* and place it in the root of the IntelliJ project directory. 3.5/4
2. **[30%]** Your game should contain a visually styled *World* populated by a rich array of *Bodies*. Specifically, your *World* should display a background image and contain multiple *StaticBodies* that fit well together and match your game concept (ex: platforms, walls, portals, etc.). It should also be populated by at least three *DynamicBodies* or *Walkers* (ex: player, enemies, etc.). Most if not all of your *Bodies* should be rendered with images to give the game an organic feel. 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. They should also be extensions of the Physics Engine's default classes – use inheritance and encapsulation to create your own classes. For high marks do one or more of the following: use inheritance to create extensible groups of game assets (ex: platforms, enemies, collectibles); consider introducing many, diverse types of bodies; add bodies to the world using loops or in response to events; make interesting use of images (e.g., animated GIFs; left/front/right images for your characters); use bodies composed of multiple fixtures or having different physical properties (ex: trampoline); use bodies that change shape or appearance during game play; use ghostly fixtures and sensors; create enemies that move on their own or collectibles that appear throughout game play.

3. **[20%]** The game should be controllable with either the keyboard or mouse (or both). At least some changes from demo code provided in class is necessary. More sophisticated controls will gain higher marks (ex: changing orientation of your character, run/walk, etc.).
4. **[20%]** Your game should respond to collisions between bodies (ex: player vs. collectible, player vs. enemy) and change the states of bodies involved (ex: decrement player's health). Wider use of collision handling will gain higher marks.
5. **[10%]** Use the World's *paintForeground* method to display a game or player statistic (ex: lives left; collected items) that changes during the game play you have implemented so far. To gain higher marks display multiple statistics or use graphics instead of plain text (ex: draw a progress bar, use small icons, etc.).
6. **[10%]** Your GitHub account should evidence ample use of GitHub as part of project development (you don't need to submit anything, markers can check this on their own).

Milestone 2 (Requirements)

You must submit a modified version of the game which compiles, and contains the features below. The game should have been built by extending the GitHub repository made available in the first week of term.

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 significant differences between the levels (e.g., different backgrounds, different *Bodies*, different behaviors). For high marks: 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. **[10%]** Rework your *Bodies* from Milestone 1 and introduce new ones if necessary, so that each level has at least two Bodies that are level-specific (ex: platforms, collectibles, or enemies that are specific to each level). Exploit inheritance as much as possible to create groups or types of *Bodies* that allow you to create new levels that feel different with minimal effort. For high marks, your *Bodies* should differ not just in appearance but also in behaviour (ex: lava platforms vs. regenerative platforms, collectibles that cause different outcomes).
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 4).
4. **[20%]** Graphical User Interface controls (for example Pause and Restart buttons). You should have at least three different controls and they should differ from those demonstrated in class. 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 complex functionality (ex:, jumping between levels, sound control).

5. **[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 high marks players should be able to save or load to and from different files selectable from the GUI using *JFileChooser* and should be able to restart mid-level (i.e., store not just the level number but also the state within the level).
6. **[10%]** Your GitHub account should evidence ample use of GitHub as part of project development (you don't need to submit anything, markers can check this on their own).

Milestone 3 (Requirements)

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

1. **[30%]** Overall game complexity**.** Your game should contain complex functionality that evidences significant expended effort and deep knowledge of programming and design concepts explored in the module. Particularly complex functionalities implemented in previous milestones do count but it is likely you will want to introduce new ones. 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).
2. **[20%]** Overall game quality and game play. To get a satisfactory mark your game-play should make sense and the game should feel polished (ex:, game features should work well together, imagery and sounds should be polished and complement the functionality of the game, etc.).
3. **[10%]** Quality of code: 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.).
4. **[10%]** Quality of code documentation: 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. **[10%]** Your GitHub account should evidence ample use of GitHub as part of project development (you don't need to submit anything, markers can check this on their own).
6. **[20%]** Video and portfolio: 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 have learned 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.

Help us mark your projects easily:

- Make your games interesting and engaging but not too difficult to play. Your markers will each need to play and assess around 40 games during each assessment and will want to do so quickly. They rarely think that having to play a game multiple times because they can't reach a platform or dodge an enemy is fun.
- Make sure your game window resolution is reasonable (ex: 800 x 600). Your computers might have high resolution displays. Without even realizing, you might be developing games that are 2000 pixels wide or heigh. If that's the case, your markers might not be able to see most of your game, let alone play it.
- Make sure the City Engine is included as a **global** library (as shown in the first lecture). Otherwise your markers will have to fiddle around to make your project work on their computer.

Annex 1: Possible extensions and new 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).

Enemies with a mind of their own: Create enemies that patrol a region, follow the main player around, or change state under certain circumstances (ex: they get attacked or time runs out).

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; weaponry that can be collected; use abstraction or inheritance (e.g., base class for projectiles).

Advanced collectibles that significantly alter game play (ex: jet pack allows player to fly; armor allows them to take more hits; different weaponry; ghostly cloak allows them to walk through walls, etc.).

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).