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 at two milestones by the deadlines listed below. Each milestone will need to meet a set of concrete requirements (detailed below), will be submitted on Moodle and marked during a brief viva scheduled after the submission. This should be a piece of individual work; any form of collaborative work is regarded as academic misconduct. The use of generative AI is accepted but needs to be declared and documented.

Milestone	Moodle Submission by	Viva on	Module Weight
1	Sun, March 9th, 17:00	Wed-Thu, March 12-13th	25%
2	Sun, Apr 27th, 17:00	Mon-Tue, April 28-29th	25%

Submission and Marking

Each deliverable will be submitted on Moodle by the deadlines listed above. 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 in person. Please plan accordingly.

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 (make sure that your project folder is under that limit well in advance – crunching it down an hour before the submission is due is difficult and stressful).

You are required to build a game that builds on the city.cs.engine library that is provided on Moodle; implementations that don't use it will receive a mark of 0. You 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 in the setting up materials. 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 will not be accepted.

Marking: Submissions will be marked in your presence – a viva – in the days following your submission. A schedule will be made available in the days before the viva. This will include the time and room of your assessment, and your allocated marker (the module leader or a member of the TA team). 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 at the designated date/time will result in a mark of 0.

During the viva markers will:

- download and unzip the file that you have submitted on Moodle on their computer and attempt to run it;
- they will then ask questions about your game to judge your implementation against the requirements *and* to assess your understanding and mastery of your code;
- assign you a mark on the spot and communicate it to you.

It is your responsibility that your project runs as submitted on Moodle and code that cannot be run might ultimately result in a mark of 0. If markers can quickly address an obvious fix they will do so, but they cannot delay the marking process. Should your project not run, you will be given a little time to figure out the cause and the marker may attempt to reassess you later, if time permits. However, it is ultimately your responsibility to ensure that your code runs!

Submitting your work on Moodle but failing to attend the viva 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]	Code that meets the requirements in full and demonstrates excellent use and understanding of most or all concepts covered by the module. It will be well-constructed, fully functional, and demonstrate a professional approach to academic practice. The student will demonstrate full understanding and exceptional mastery of their submitted code. Where relevant, there will be

	evidence of independent reading, thinking and analysis.
60-69 [2(i)]	Code 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. The student will demonstrate solid understanding of their submitted code.
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 but structure and presentation may not always be clear. Attempts to demonstrate academic practice will be evident. The student will demonstrate a general understanding of their submitted code (but may need to be helped along occasionally).
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. Student will demonstrate some understanding of their submitted code – this may be limited and in some cases reflect incorrect mental models.
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. Students will show limited or no understanding of their code.

Extenuating Circumstances: 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). If your application is approved, you will be able to submit your assignment and sit the vivas as shown below:

Milestone	EC Moodle Submission by	EC Viva on
1	Sun, March 16th, 17:00	Mon-Tue, April 28-29th
2	Sun, May 4th, 17:00	Wed, May 7th

EC submission schedule. Only applicable for approved extensions!

Please submit your work on Moodle, in relevant "EC Late Submission" areas. These will only become available once the regular deadline has passed.

Please note the following:

- Please do not email the module leader to ask about your particular circumstance or to ask for assessment outside the timeframes listed above. Contact **ug.cs@city.ac.uk** if you have any questions. Only contact the module leader if:

    - You were not able to submit your work in the EC submission areas on Moodle despite being on time
    - You were not included in the viva schedule despite having been granted an extension
- You are encouraged to deal with your extenuating circumstances (sickness, family emergency) as your priority. However, if you are capable, consider submitting something (even incomplete work) by the regular deadline as a contingency plan (should your EC application not be approved, you would still earn marks).
- Submissions can be at most one week late and can be assessed at the latest about a month later. Sometimes EC cases can take longer than this to resolve (if for example, required evidence was not provided or was insufficient):
    - Should your case take longer than the allowed submission timeframe, submit your work within the timeframe even if you haven't received the EC decision yet.
    - Should your case take longer than the assessment allowance, then you will unfortunately need to resit the project in August.

Use of AI (e.g., ChatGPT, Bard, Gemini): Using these aids without acknowledgment is academic misconduct. However, you may use such aids if you think it useful if and only if you include with your submission a declaration of use. This should include:

- a description of exactly how you used AI tools (what tasks you sought helped for; samples of the prompts you used; how you integrated the outputs into your project)
- a reflection of how their use helped your learning and programming in general (rather than the completion of your assessment)

Title your declaration "Use of AI" and include it along with your game code in the main root of your zip submission.

Additionally, please note that you are responsible for understanding any AI generated outputs you choose to use in your project. We actively probe your understanding of code during the viva assessments - including in your project code and structures that you don't master may significantly lower your mark.

Milestone 1 (Requirements)

You must submit 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. **[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; create visual layering by playing with transparency and background vs. foreground rendering.

2. **[20%]** The game should be controllable with either the keyboard or mouse (or both). At least some changes from demo code provided in class are necessary. More sophisticated controls will gain higher marks (ex: changing orientation of your character, run/walk, changing abilities, etc.).
3. **[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.
4. **[20%]** 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.).
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).

Prerequisite for assessment: early demonstration of a working development environment: During tutorials of weeks 3 and 4, tutors will assess (on an individual basis) whether you have a working development environment, i.e., IntlliJ, physics engine and Github installed; a game structure with known code location that compiles and runs. To receive a Milestone 1 mark, you need to pass this perquisite assessment – make sure to attend at least one of tutorials 3 and 4.

Milestone 2 (Requirements)

You must submit a fully functional game. The game should have been built by extending the GitHub repository made available in the first week of term. The final assessment will focus on the following:

1. **[20%]** 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. **[30%]** Overall game complexity**.** Your game should contain functionality that evidences significant expended effort and deep knowledge of programming and design concepts explored in the module. 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. **[15%]** 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.).
4. **[15%]** Quality of code and 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.) and how extensive your documentation is (both inline comments and Javadoc documentation is necessary).
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. **[10%]** Video: 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.

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 60 games during each viva-session and will want to do so quickly. They will not find it fun to have to play a game multiple times because they can't reach a platform or dodge an enemy.
- 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: 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.

Graphical User Interface: You may consider implementing some of the following functionality: buttons or controls that trigger game events; laying out controls manually, using Layout Managers rather than using the IntelliJ GUI editor, and perhaps nesting Layout Managers; use components that were not covered in lecture (text boxes, drop downs); link GUI elements to complex functionality (ex:, jumping between levels, sound control); 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.

Saving and loading game-state: 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).

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.).

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).