Computer Science E214 Project: Stellar Crush

Lecturer: Steve Kroon
Moderator: Cornelia Inggs

Hierdie instruksies is slegs beskikbaar in Engels. Indien u enige deel van die volgende teks nie verstaan nie, kontak asb. die dosent so gou moontlik.

This project is due by 7 May 2017, and must be submitted by then on SUNLearn. It is strongly recommended that you allocate time to work on this project early in order to ensure you complete the project in time—see the development plan section for a suggested order to tackle the project.

Project demonstrations are scheduled for 10 May 2017 during the normal practical period. Note that the project demonstration is compulsory, and you will get a mark of zero for the project if this is missed. See the study guide for more details.

1 Introduction

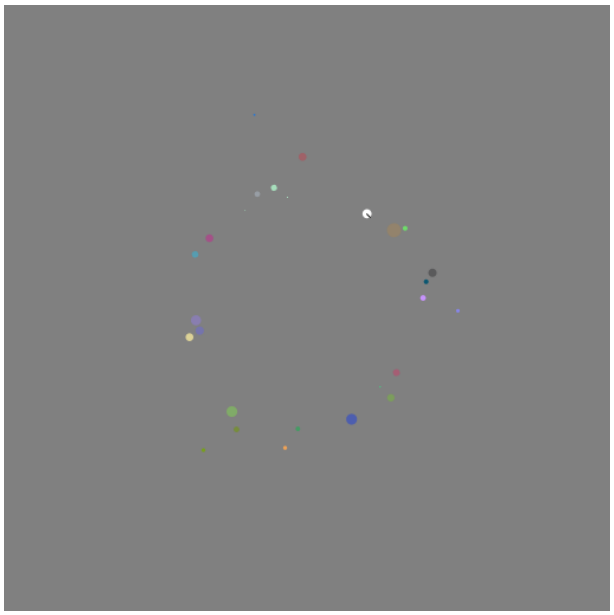
For this project, you will implement the core aspects of a video game, and hopefully expand it in new and challenging directions that motivate and excite you. The minimum requirement version of the game will be demonstrated in class: it involves simulating a 2-D universe of objects moving under the force of gravity, including one object under a player's additional control, and displaying the universe in a top-down fashion as well as from the player's perspective. To get you started and give you some structure, some code with potentially useful comments is provided. In general, you do not need to use the provided codei, but will need to motivate if you do not.

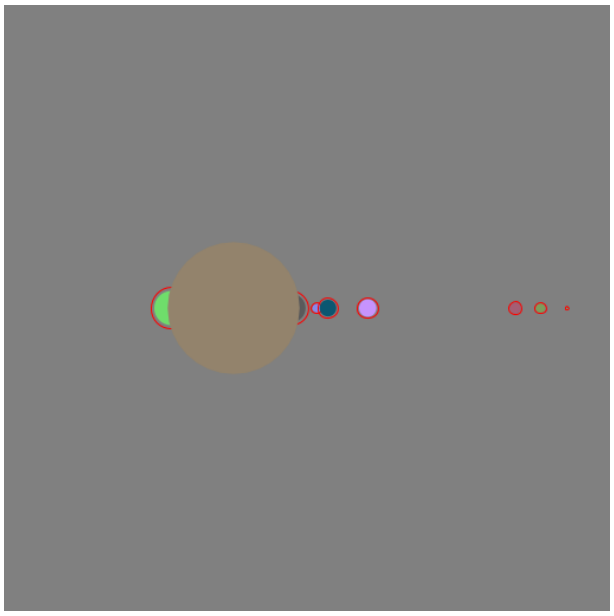
2 Minimum Deliverables

A mark of 75% for the project may be obtained for a perfect implementation at least as complex as the following:

- A title screen with instructions for the user must be presented at the start of the program, such as below. The title screen is left by pressing any key. You can name your program anything you'd like.

- The visual appearance of the game should be at least as complex as in the following diagrams. The first shows a top-down depiction of the objects in the game, such as that used in the N-body simulation in the course lectures. The second is a first person view of part of the universe from the perspective of the player, highlighting certain objects with an additional ring. For simplicity, we assume all objects, including the player, are spherical.





- There should be a *player* shown on the top-down screen. The player has a viewport out of which it can view the other objects in the universe in front of it: the viewport is at the front of the player. The player can rotate with the left and right arrow keys, effectively changing the direction the viewport faces. The player can also be accelerated using forward or reverse thrusters, allowing acceleration either forward or backwards.
- There should be multiple other (non-identical) objects moving subject to gravity on the screen. The player is also subject to and generates gravitational forces like the other objects.
- If two objects collide, either the two objects merge into one, or one of the objects splits the other into two or more subparts. You can decide on the criteria for which happens when, but both should be evident in your game. Your code should make an effort to conserve momentum of the system when dealing with merges and collisions, but suitable approximation for simplicity/convenience are acceptable.
- Your game must have some criteria which are easily achieved under which the game ends. Once the game is over, a suitable game over message or screen must be displayed.
- While no specific guidelines will be given on the number of objects, their masses, and their spacing, your choices should try to ensure somewhat enjoyable gameplay.
- The game quits when the user presses "m".
- Your program must be structured in a modular fashion, using principles of object-oriented design. Chapter 2 and 3 of the textbook in particular discuss various relevant considerations when implementing larger programs. In particular:

  - To ease maintenance and debugging, code should be well-commented.
  - To aid encapsulation, all variables must be private — except for declaring constants using `static final` (where you may add `public` or `protected` as appropriate).
  - To encourage abstraction, your program must sensibly define and use at least one interface.
  - Your program must sensibly make use of polymorphism by class inheritance in at least one case.

- To ensure you develop experience using alternative collections to arrays and lists, *you may not use array- or linked-list based lists (e.g. `java.util.ArrayList`) at all, and the only place your code may use arrays is the `args` array provided to the `main` function of a class and when constructing double arrays to use when constructing `Vector` objects.*

3 Possible extensions

The final 25% of the project mark is for extending the minimum deliverables by adding significant additional functionality. *Your submission will only be considered for these marks if you obtain at least 60/75 for the core functionality.* This is an exercise in creativity, and each student's efforts will be judged according to the level of improvement to the game and the amount of effort involved in making these additions. If you use graphics/sound/music created by someone else (this must not be another student), you must describe (comment in your code where they are used) where/how you acquired them, and how you manipulated them into usable form. **This process must be your own work.** If you created your own graphics/sound/music, be sure to mention this!

Some ideas for your consideration are listed in the comments in the project skeleton. Note that these suggestions vary widely in terms of extent and difficulty.

4 Freedoms

- Any use of an external library must be discussed with the lecturer for approval. In particular, any library that you use must not trivialise the implementation of the core game elements.

- You are also free to provide a custom version of the textbook's standard libraries as part of your submission in case you need to change low-level aspects of their interaction with Java's underlying libraries, as long as these changes are your own work and you can demonstrate why they are necessary for what you are trying to do.

- While your program must be structured in a sensible way and some insight may be gleaned from the template code provided, the actual structure is up to you. Be ready

to motivate your choices and deviations from the provided skeleton structure. The suggested order and method for implementing each feature are entirely optional, and are merely meant as a guideline. Thinking through the problems and coming up with solid solutions is what projects are all about.

5 Design and implementation

Note that as we progress through the course, you will learn new concepts allowing you to address new aspects of the project, or to better implement aspects of the project you have already tackled.

5.1 Project structure

To give you an idea about how to approach structuring your project, some skeleton files and code is provided. This is from a solution implementation which is structured as follows:

- Main class `StellarCrush`, which handles the details outside the game loop and uses a `GameState` object to handle details in the game loop;
- Class `GameState` represents the state of the game, and is responsible for managing updates every time step in the game loop (process general input, update state of various objects, update the screen);
- Interface `IViewPort` which specifies methods needed to enable rendering from a first-person viewpoint;
- Class `GameObject` representing properties of all in-game objects;
- Class `PlayerObject` which extends `GameObject` and implements `IViewPort` in order to render the universe from the player's perspective;
- Class `Camera` which handles drawing the view from a specific `IViewPort` object;
- Class `GameObjectLibrary` which has static methods for constructing game objects and a player object;
- Class `Vector` from the textbook for representing locations, velocities, forces, etc.—*note that you may not change this class*;
- Class `VectorUtil`, a library with some useful functions for dealing with (especially 2-dimensional) vectors—use this for functionality you need for vectors not included in the `Vector` class.

While the final game should generate random objects in the game universe, the example universes used in the N-body case study can be useful for checking certain aspects of your code. To this end, some of these examples are also provided.

5.2 Development order

To tackle the project, you might find the following suggested path to completing the minimum deliverables helpful. First, complete the following steps.

1. Ensure you can draw the title screen of the game—in `StellarCrush.java`
2. Accept a key to start the game, then display an N-body simulation for a while (no collision detection yet), and then display a game over message - all using `StdDraw`. Create the bodies in `GameObjectLibrary.java`, and put the animation loop in `GameState.java`. Each body created should be of type `GameObject`, and you might use the code for the `Body` type in the N-body case study as an inspiration—in particular, each `GameObject` should be responsible for moving (and later updating) itself at the request of `GameState.java`.
3. Implement `PlayerObject.java`, ensuring that it implements the `IViewPort` interface. It does not yet need to accept keyboard input. Add a player to your game in `GameObjectLibrary.java`.
4. Implementing `Camera.java` for rendering a first-person view from an `IViewPort` on a `Draw` canvas. Equip the `PlayerObject` type with such a camera so that you can see the universe from the player's point of view in a separate window. Note that keyboard input for the player comes from the camera's canvas, so that should be the focus window during the gameplay phase.
5. Add key detection to `PlayerObject.java` to allow the player to rotate the direction it is facing.
6. Add functionality allowing activating thrusters to affect the force applied to the player by itself, thus affecting its movement. (Note that usually calculating the resultant force on an object excludes calculating a force being applied to oneself!)
7. Now your player should be able to fly around the universe. Next detect when your player touches another game object and remove the object from the universe when this happens. For this, either the player or the other game objects will require some kind of function for updating the world to be called in (typically) each iteration of the game loop.
8. Finally, refine your code for dealing with what happens when two objects touch each other: define how objects should be merged, or how objects should be split up, paying attention to conservation of mass and momentum. (Note that being exact will lead to severe headaches, but making some simplifying assumptions will considerably ease things.)

Note that you can still process keys from the `Draw` window even if the first-person view is not correct, for example, so you do not have to follow the steps above slavishly—particularly if you are still struggling.

6 Marking Scheme

First, some general comments:

- **Note that all programs may be tested for plagiarism, both against all the other programs in the class, as well as other implementations of the program.**
- Finishing a few things that work is worth more than starting a lot of things that do not work at all. In particular, compilation errors, or code that refuses to do anything sensible, are immediate grounds for assigning a mark of zero for the project.
- You will have to give a demonstration of the program, during which time you may have to explain how certain parts work. If you cannot do that satisfactorily, you may be subjected to further testing and possibly fail the project.
- Stringent negative marking applies for poor style and general impression, as well as poor adherence to OOP principles as described above.

As an indication of how marks will be allocated, a preliminary marking sheet is given on the final page of this document. Note that the marking scheme does not reflect a recommended implementation order—see the comments in Section 5.2.

You will also need to complete a questionnaire on SunLearn providing some details about your code before the deadline, which will be consulted during the marking process. This will likely include at least the following:

- Document assumptions/approximations made in your code to simplify the game dynamics compared to real (Newtonian) physics.
- Where (if anywhere) did you use copy constructors or defensive copies?
- Document and motivate deviation from the provided skeleton.
- Document your use of polymorphism via class inheritance.
- Document your use of interface inheritance.
- List any extensions implemented in decreasing order of complexity
- Document any changes to the standard library provided by the textbook authors
- Document any additional libraries required

Student number and name:

	Preliminaries
	Above details do not match student card and SUNLearn Submission name incorrect (mark of zero) Submission structure incorrect (mark of zero) Code does not compile (mark of zero) Code does not do anything sensible (mark of zero)

	All Minimum Deliverables Complete (75 marks)
	Otherwise, mark completed features (10 marks each): Title screen showing instructions and game over message Random N-body simulation including a player Controllable player (rotation and thrusters) First-person view—fixed location (origin) First-person view—from player Intersection detection—at least removal Intersection detection—merging Intersection detection—splitting

	Style and Design Problems (-10 for each marked)
	Sloppy code (e.g. poor or missing comments, inconsistent or missing indentation, unhelpful variable names; poor use of functions) Disallowed non-private variables present No interface inheritance No polymorphism via class inheritance Disallowed array use Disallowed list/arraylist use

	Advanced Functionality
	60/75 required for core implementation after deductions. Student must provide list of extensions with marksheet. First 3 features 5 marks each, thereafter 4, 3, 2, and 1 mark. Guideline: an additional feature should correspond to around 10 marks in the core functionality.

Demi number and signature: