The University of Melbourne
School of Computing and Information Systems
SWEN20003 Object Oriented Software Development

Project 2, 2018

Released: Friday April 13th
Project 2A Due: Friday May 4th, 11:59pm
Project 2B Due: Friday May 18th, 11:59pm

Overview

In this project, you will create a graphical space shooter game in the Java programming language, continuing from your work in Project 1. [1]

This is an **individual** project. You may discuss ideas and approaches with other student, but all submitted code must be your own work. **You may not share any part of your source code with other students, even if this code is in the public domain.** You may use any platform and tools you like to develop the game, but we recommend using the Eclipse IDE.

You are not required to design any aspect of the game itself – this document should provide all necessary information about how the game should work. You are however required to design the classes for your software implementation of the game.

There are two parts to this project, with different submission dates.

The first task, **Project 2A**, asks you to produce a class design demonstrating how you intend to implement the game. This should be submitted in the form of a UML diagram (to be discussed in lectures and tutorials). This diagram should show clearly all classes you intend to implement, their relationships (e.g. inheritance and association), and their attributes, as well as the main primary public methods. (Methods such as getters and setters need not be explicitly included). You **must** use correct UML notation for this diagram. It should be submitted in PDF format.

The second task, **Project 2B**, asks you to complete the implementation of the game as detailed in the rest of this specification. You **do not** have to follow your UML diagram; it is there to demonstrate knowledge of UML, as well as to encourage you to think about your design before you start programming.

[1] We will be releasing a full solution for Project 1 when all submissions have been received. You are welcome to use all or part of this solution in your submissions for Project 2.

Shadow Wars

Game overview

Shadow Wars is an action game where the player must defeat wave after wave of robotic *enemies*, culminating in an epic fight against the robot mother ship. The enemies appear in a set pattern, and increase the player's *score* when they are defeated. They also have a chance to drop *powerups*, which increase the player's abilities temporarily.

Gameplay

This is a real-time game; objects may move even without the player pressing any keys. The game takes place in *frames*, with many frames occurring each second. On each frame:

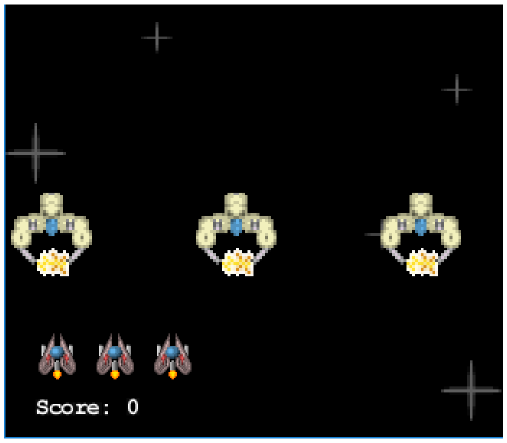
1. All sprites, including enemies, the player, and powerups may move on the screen.
2. Collisions between sprites are detected, and the sprites respond accordingly.

The player may also speed up the game by five times by holding the **S** key; this is in the interests of testing the game. All objects should move five times faster in this state.

The player

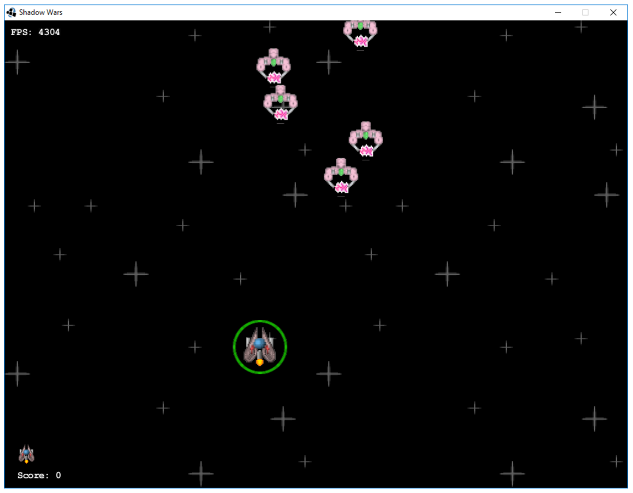
The player behaves similarly to Project 1. A new feature is that the player should continue to fire laser shots for as long as the spacebar key is pressed; this should be limited to a rate of one shot every 350 milliseconds.

Furthermore, the player now has *lives*. The player begins with three lives. When the player makes contact with an enemy, instead of the game ending, the player should lose a life. This should be displayed in the bottom-left of the screen as shown below, using the image `res/lives.png`:



Note that the player has three lives, so three copies of the image are drawn. As a **suggestion**, you can draw the first image at $(20, 696)$, with 40 pixels between the centres of each image.

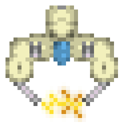
When the player loses a life, the player should display the image `res/shield.png` centred at its position for 3000 milliseconds. For this duration, the player should not lose a life when they make contact with an enemy. The player should begin the game with this temporary shield.

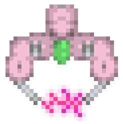


The player, finally, has a *score* keeping track of how many enemies have been destroyed. Different enemies are worth a different number of points. This should also be displayed in the bottom-left of the screen, as shown above. The screenshot renders the score at (20,738).

The enemies

There are four varieties of enemies in this game, detailed below. All enemies should begin the game off the screen at a y-coordinate of -64. They are associated with a *delay* value, which is the number of milliseconds they should wait before they begin moving.

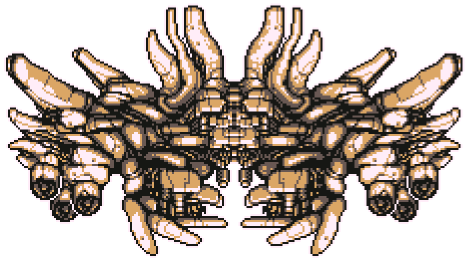
- **The basic enemy:** 
  The basic enemy does nothing except move down the screen at a rate of 0.2 pixels per millisecond. It should be destroyed when it makes contact with a player laser shot. It is worth 50 points.

- **The sine enemy:** 
  The sine enemy moves down the screen at a rate of 0.15 pixels per millisecond. Its x position should be centred around its x position when it is created, but with an offset added according to the following formula:
  $$\text{offset} = A \sin\left(\frac{2\pi}{T}(t - \text{delay})\right)$$
  where $A = 96$ is an amplitude constant, $T = 1500$ is a period constant, t is the number of milliseconds that have passed so far in the game, and delay is the enemy's delay value. In

this way, the enemy moves down the screen in a wave pattern. It should be destroyed when it makes contact with a player laser shot. It is worth 100 points.

- **The basic shooter:**
  The basic shooter chooses a random y-coordinate between 48 and 464, and moves down the screen at a rate of 0.2 pixels per millisecond until it reaches that y-coordinate. Once it reaches this position, it should fire one *enemy laser shot* every 3500 milliseconds. It should be destroyed when it makes contact with a player laser shot. It is worth 200 points.

  An *enemy laser shot* behaves like an enemy. It moves down the screen at a rate of 0.7 pixels per millisecond, and is not destroyed when it makes contact with a player laser shot.

- **The boss:** 
  The boss has slightly more complex behaviour than the other enemies. It should initially move down the screen at a rate of 0.05 pixels per millisecond until it reaches a y-coordinate of 72. The boss should then repeat the following behaviour:

  1. The boss should wait 5000 milliseconds.
  2. The boss should choose a random x-coordinate between 128 and 896.
  3. The boss should move towards this x-coordinate at a rate of 0.2 pixels per millisecond.
  4. The boss should wait 2000 milliseconds.
  5. The boss should choose another random x-coordinate with the same rules as above.
  6. The boss should move towards this x-coordinate at a rate of 0.1 pixels per millisecond. While doing this, it should fire 4 shots every 200 milliseconds, with x offsets of -97, -74, 74, and 97 relative to the boss's position.
  7. The boss should continue firing until 3000 further milliseconds have passed from when it started moving.

  The boss should be destroyed when it has made contact with sixty player laser shots. It is worth 5000 points.

Note that it should be impossible to destroy an enemy while it is not visible on the screen.

The enemy file format

The file containing data on which enemies to generate is called `waves.txt`. The file is made up of a series of lines, with a new line separating each line. It has two kinds of lines:

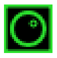
1. `# This is a comment`
   Any lines that begin with a # should be ignored when generating the enemies.

2. `EnemyName,x,delay`
   A line of this form is an instruction to generate an enemy of type `EnemyName`, at x-coordinate `x`, and with a delay value of `delay`.
   Recall that the *delay* is a number of milliseconds for which the enemy should take no action.

This file should be interpreted by your program when the game first begins. All enemies should be created at this time.

Powerups

There are two kinds of *powerup* in the game. When the player makes contact with a powerup, a certain effect should take place on the player for 5000 milliseconds. The player should then return to their standard behaviour.

- **Shield Powerup:** 
  The shield powerup simply grants the player a shield, similar to the shield that occurs when the player loses a life.

- **Shot Speed Powerup:**
  The shot speed powerup reduces the delay between laser shots fired by the player to 150 milliseconds while it is active.

When an enemy is destroyed, there is a 5% chance a powerup will be dropped at its former position. There is an equal chance of dropping either type of powerup. Once created, powerups should move down the screen at a rate of 0.1 pixels per millisecond.

Implementation checklist

This project may seem scarily big. Since there are a lot of things you need to implement, we have provided a checklist, roughly ordered according to how we think you should implement them. Next to each item is the number of marks that particular feature is worth, for a total of 8.

- All types of enemies, not including the boss, are in the game and behave according to the specification. (1 mark)
- The boss is in the game and behaves according to the specification. (1 mark)
- Enemies are correctly read from the provided file. (1 mark)

- Enemies are delayed correctly, so that they appear in waves. (1 mark)
- Lives and score system implemented. (1 mark)
- The player gains a shield when they lose a life. (1 mark)
- Powerups are in the game, and are dropped in the correct manner. (1 mark)
- Powerups have the correct effects when they make contact with the player. (1 mark)

Customisation

Optional: we want to encourage creativity with this project. We have tried to outline every aspect of the game design here, but if you wish, you may customise any part of the game, including the wave, graphics, types of enemies, etc. You can also add entirely new features. However, to be eligible for full marks, you **must** implement all of the features in the above implementation checklist.

For those of you with far too much time on your hands, we will hold a competition for the best game extension or modification, judged by the lecturer and tutor. The winning three will be demonstrated at the final lecture, and there will be a prize for our favourite. Past modifications have included drastically increasing the scope of the game, implementing jokes and creative game design, adding polish to the game, and even introducing networked gameplay.

If you would like to enter the competition, please email the head tutor, Eleanor McMurtry, at `mcmurtrye@unimelb.edu.au` with your username and a short description of the modifications you came up with. I can't wait to see what you've done!

The supplied package

You will be given a package, `project2-package.zip`, which contains all of the graphics and other files you need to build the game. You can use these in any way you want. Here is a brief summary of its contents:

- `res/`
  - `basic-enemy.png` – the basic enemy image
  - `basic-shooter.png` – the basic shooter image
  - `boss.png` – the boss image
  - `credit.txt` – a text file containing image credits
  - `enemy-shot.png` – the enemy laser shot image
  - `lives.png` – the lives counter image
  - `shield.png` – the shield effect image
  - `shield-powerup.png` – the shield powerup image

  - `shot.png` – the player laser shot image
  - `shotspeed-powerup.png` – the shot speed powerup image
  - `sine-enemy.png` – the sine enemy image
  - `space.png` – the starry background image
  - `spaceship.png` – the player's ship image
  - `waves.txt` – the data file for the enemy generation

Submission

Submission will be through the LMS. Please submit **your entire project folder**, including resources and all code.

Note that **all public methods, attributes, and classes** should have Javadoc comments, as explained in later lectures. You are not required to actually generate the Javadoc.

Extensions and late submissions

If you need an extension for the project, please email Eleanor at mcmurtrye@unimelb.edu.au explaining your situation with some supporting documentation (medical certificate, academic adjustment plan, wedding invitation). If an extension has been granted, you may submit via the LMS as usual; please do however email Eleanor once you have submitted your project.

The project is due at 11:59pm sharp. As soon as midnight falls, a project will be considered late unless an extension has been granted. There will be no exceptions. There is a penalty of 2 marks for the first day a project is submitted late, plus 1 mark per additional day. If you submit late, you must email Eleanor with your student ID and number so that we can ensure your late submission is marked correctly.

Marks

Project 2 is worth **22** marks out of the total 100 for the subject.

- Project 2a is worth **6** marks.
- Project 2b is worth **16** marks.
  - Features implemented correctly: **8 marks** (see *Implementation checklist* for details)
  - Coding style, documentation, and good object-oriented principles: **7 marks**
    - Delegation – breaking the code down into appropriate classes (2 marks)
    - Use of methods – avoiding repeated code and overly complex methods (1 mark)
    - Cohesion – classes are complete units that contain all their data (1 mark)

    * Coupling – interactions between classes are not overly complex (1 mark)
    * General code style – visibility modifiers, magic numbers, commenting, etc. (2 marks)
  - Documentation (javadoc): **1 mark**

Acknowledgement

This game was designed by Eleanor McMurtry.