Computer Science E214 (2024)
Project: Cosmic Conquistadors

Lecturer: Steve Kroon
Moderator: Gavin Rens

The deadline for project submission on SunLearn is 11h59 on 29 April. It is strongly recommended that you hold to the suggested timetable (see the development plan section) in order to ensure you complete the project in time.

We aim to hold project demonstrations during the tutorial period of 29 April. Note that the project demonstration is compulsory, and you can get a mark of zero for the project if this is missed—please see the study guide for more information.

This is a project for groups of (typically) three, and a group mark will be awarded. Peer rating will be used to modify the group mark to obtain each group member's mark—this adjustment is based on the rest of the group's perceived level of contribution of the group member to the group effort. Details of the peer rating system used for calculating the adjustments will be included with this project description on SunLearn.

1 Introduction

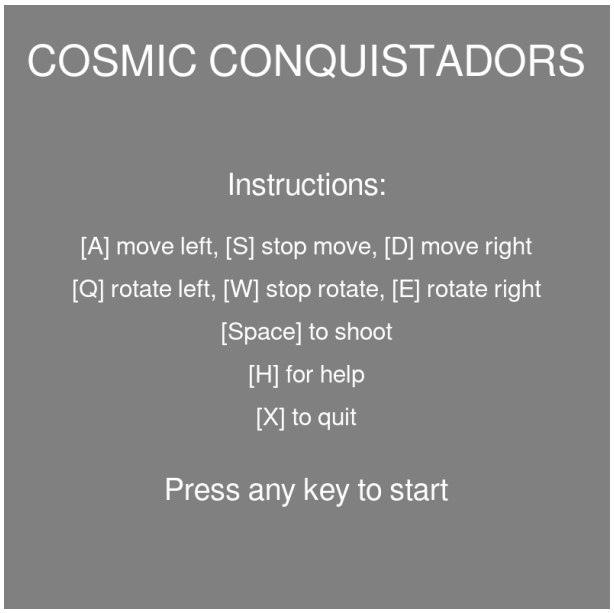
Space Invaders was an early video game that, according to its Wikipedia page, "helped expand the video game industry from a novelty to a global industry". It was also a game I loved playing as a child. According to the same article, the developer of the original game had to design custom hardware and development tools to complete his task. Fortunately, creating the game with modern technology is a substantially easier task, as you will hopefully discover: for this project, you will implement a game similar to the core of the Space Invaders game. Beyond this core, you will be able to expand the game in ways that excite you — some may choose to add more elements present in the original game, while others may add totally different gameplay elements.

To get a feel for the game, you can play it online at http://www.pacxon4u.com/space-invaders/, and read more about the game at its Wikipedia page (http://en.wikipedia.org/wiki/Space_Invaders).

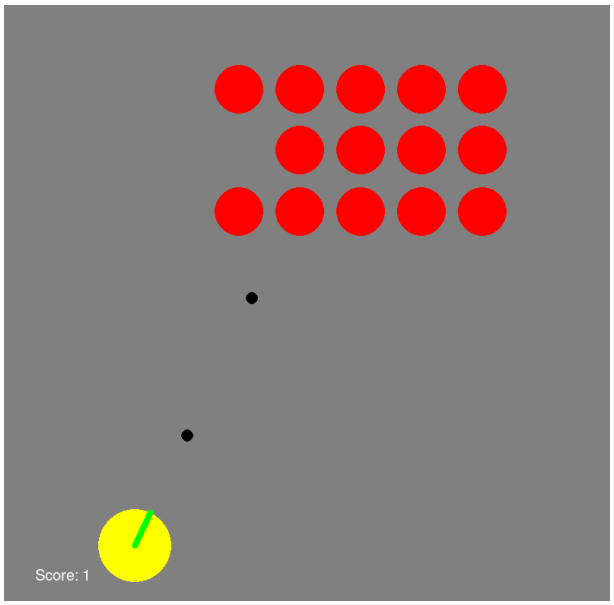
2 Minimum Deliverables

A group mark of 60% 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 the space bar. You can name your program anything you'd like.



- The visual appearance of the game should be at least as complex as in the diagram:



- There should be a *shooter* at the bottom of the screen, which can be moved left and right, and shoot missiles from its turret at intervals. The shooter can not move off the screen. The turret can be rotated *relative to the centre of the shooter* (i.e. the turret rotates, but the shooter does not); however, it can not aim below horizontal.
- Missiles shot from the turret should be visible as they travel across the screen.
- There should be a grid of *enemies* which begin near the top left of the screen, and approach the bottom of the screen by moving *in formation* back and forth across the screen, descending when they reach the edge of the screen. (No enemy may move off the screen in this process.)
- If a missile strikes an enemy, the missile and the enemy are both destroyed. If all the enemies are destroyed the game is over and the shooter wins.
- If an enemy touches the shooter or reaches ground level, the game is over and the enemies win.
- If the game is over, a game over message or screen must be displayed. After a short delay the game should automatically restart.
- A score indicator must be shown which starts at zero, and increases as enemies are destroyed.
- While no specific guidelines will be given on the number of enemies, the interval between shooting, or the speeds of the enemies or shooting, your choices must be reasonable to ensure challenging, though not impossible, gameplay.
- The game quits when the user presses 'q'.
- 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).

3 Possible extensions

Your group may score up to 60% for the minimum specified game elements. Up to 40% more can be obtained based on various improvements you introduce into your game.

This is an exercise in creativity, and each group'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 (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 improvements to consider:

- Improved graphics. The basic visual requirements are extremely bare-bones. Improve the appearance of the game and title screen in various ways. This could be done by making the game appearance resemble the original more. Alternatively, you could reimagine the look of the game in another aesthetically pleasing way (again, the difficulty of the implementation does matter though).

- Add sound. Add sounds for shooting, for destroying enemies, for the enemies descending, possibly adding music. Be aware that the `stdaudio` library is fairly limited in this regard, however.

- Add interesting gameplay elements. While the core game mechanics need to remain intact, adding interesting novel gameplay elements is encouraged. Do discuss large changes with the lecturer.

- Add a current score/high score screen which shows when the game is over.

- Add multiple lives, and levels which get progressively harder when the previous level is completed.

- Add an additional shooter to be controlled by a second player (Competitive and collaborative modes are both possible.)

- Allow the enemies to drop bombs, and add bunkers, as in the original game.

- Allow ways to control the speed of the shooter's movement and turret rotation.

- Give the enemies, shooter(s), and bunker(s) hitpoints which decrease when they are damaged, such that they are only destroyed when the hitpoints are reduced below zero.

- Give missiles a burst radius (possibly depending on how long the shooter waited before shooting each one).

- Add various powerups for both the shooter and the enemies. (These might be obtained by moving over them, or by shooting them).

Note: While your program must be structured in a sensible way, the actual structure is up to you. Be ready to motivate your choices (especially the quirky ones). 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.

4 Development plan

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.

After the first three weeks of class, we should have covered the textbook's standard library modules, as well as the use of functions. By then, you should have at least an implementation which can: (i) display a title screen; (ii) repeatedly start a new game and display a game over message; (iii) in each new game have a game loop which displays a shape on the screen and can move it based on keyboard input. It is recommended you implement horizontal movement before implementing turret rotation.

After the first week of the 2nd term—and perhaps before the test week—we should have covered creating new classes, so that you can create classes for the shooter, enemies, and missiles, as well as a game state class to manage these objects and their updates as a game progresses. You should now be able to get the core of your game going: having the various game objects appear and move (based on your keypresses in the shooter's case), and be able to shoot and detect missile strikes. Focus first on having one enemy appear and move, and being able to shoot and destroy it with a single missile. At this stage, we may not have discussed much about designing classes, so you may wish to reconsider your design and implementation of these classes once we do.

Near the final deadline, we should be close to the end of the most relevant course material. At this stage, you should be refining your code to improve the design, and adding missing features. Sensible use of object-oriented design could probably improve your implementation, and you should at this stage have some background to try to make use of them.

For some guidance w.r.t. design, ensure you use modular programming by implementing multiple modules/classes. For example, classes for the enemies, the shooter, and the bullets could each be in their own module, and optionally inherit from a common `GameObject` class capturing common aspects. Aim to keep the logic and data for the game loop separate from that for the individual game objects: having a data type for the game state that has methods for executing the game loop is one way to do this.

5 Additional constraints and notes

Your project will be evaluated on the reference virtual environment in FIRGA. You may thus not use any libraries not included in the reference environment. In addition, you may not use the `pygame` library in any way except by calling functions in the `stddraw` or `stdaudio` library. **Failure to adhere to these constraints on libraries will lead to a mark of zero for the project.**

We remind you again that project submissions are regularly tested for plagiarism, both against all the other programs in the class, as well as other implementations of the program. In addition, your group will have to give a demonstration of the program, during which

time various group members may have to explain how various parts work. You will be required to submit individual plagiarism declarations regarding your work on your group's project. Read the study guide section on academic misconduct and the ducuments referred to there in this regard.

To more clearly discern the contribution of each student to the project, each section of code (typically functions/methods) should be preceded by a comment block indicating the history of who worked on that block and what changes they made. This comment block should be updated each time a student works on that portion of code. Note that maintaining this effectively is for your own benefit if there are queries about plagiarism or the relative contributions of group members.

6 Marking Scheme

REMEMBER: Doing a few things that work well, are worth more than a lot of things that do not work at all. In particular, code that can not be succesfully run, simply crashes, or refuses to do anything sensible, are immediate grounds for assigning a mark of zero for the project.

Your game will be tested through play, and your code will be scrutinised. For the minimum deliverables: when elements of your program work meet a requirement, you are allocated the corresponding marks; if your program fails a requirement, the implementation of the requirement is lacking, or a group member can not explain the relevant functionality when asked, you are not awarded the corresponding marks. In addition, negative marking applies for poor style and general impression. For the possible extensions, partial marks for each extension can be awarded. Straightforward, functional implementation of many of the suggested extensions are roughly worth the same as a basic minimum deliverable, but the marks allocated for certain suggested extensions may depend on the quality and scope of the extension implemented.

A preliminary marking scheme is given on the following pages. Note that the marking scheme does not reflect a recommended implementation order—see the comments in Section 4.

Category	Mark Allocation
Minimum Deliverables	3 each
Title screen showing instructions	
Drawing shooter and turret	
Drawing enemies	
Turret shoots missiles, missiles drawn	
Missiles move correctly at the angle shot	
Multiple missiles possible simultaneously	
Turret has recharge time between missiles	
Enemies move and descend	
Shooter can move left / right and stop	
Shooter can not leave screen	
Turret can rotate left / right and stop	
Turret can not rotate below horizontal	
Missile strike destroys enemy and missile	
Score displayed and increases on missile strike	
Game over by killing all enemies	
Game over by enemy touching shooter	
Game over by enemy reaching ground	
Game over message	
Restarting game after game over	
Quit key	
Subtotal	60
Style and Design	
No comments / Unnecessary comments	-5
Poor variable naming convention	-5
Poor use of functions and modules	-5
Poor design and use of objects	-5
Sloppy style and inconsistent/lacking indentation	-5
Correct use of global variables	-5
Poor overall functionality / graphical appearance	-5
Subtotal	-35

Category	Mark Allocation
Further Marks	Approximate %
Improved graphics	Variable
Add sounds	4
Add music	4
Leaderboard/high score screen	4
Progressively harder levels	4
Extra lives	4
Additional shooter	4
Enemies counterattack	4
Bunkers	4
Power-ups	Variable
Hit-points	4
Different missile types	Variable
Different enemy types	Variable
Other	Variable
Subtotal	40
Maximum Total	100