CS1527 Object-Oriented Programming
2017-2018

Mini-Project 2: Mars Lander Arcade Game

Important
This assessment is worth **40%** of the overall marks for course CS1527.

Your submission must be submitted via MyAberdeen by **Friday April 27th 2018 at 12:00 noon**.
Late penalties will apply if you miss the deadline.

Remember that as this is an individual assessment, ***work submitted must be your own***.
If you haven't already done so, you should familiarise yourself with the University guidance on plagiarism, available at https://www.abdn.ac.uk/sls/online-resources/avoiding-plagiarism/

Note*: If you do not submit this assignment, you will be given a C6.*

Read through this entire document before starting to program.

Introduction:

The Python `pygame` package is an Open Source library for making multimedia applications and games. It is built on top of the excellent SDL library, and like that library it is highly portable and runs on nearly every platform and operating system.

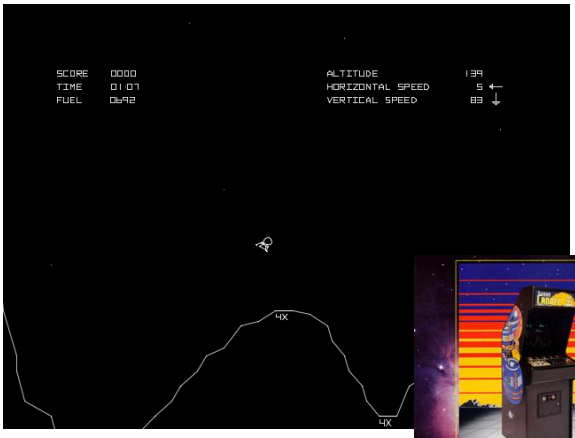
> **`pygame`** home page:
> https://www.pygame.org
>
> Documentation, including tutorials and sample code:
> https://www.pygame.org/docs/

`pygame` is excellent for reproducing classic arcade games from the 1970s and 80s, and it has been used to recreate Pong, Pacman, Snake, Defender, and Asteroids amongst many others.

We introduced `pygame` in CS1527 Practical 8 and you are urged to study the sample programs and exercise solutions before starting work on this mini-project assessment.

Background:

One of the first arcade games to be based on a real space mission and using pseudo-realistic physics was Lunar Lander (Atari, 1979).



In the game, players control a space vehicle as it makes its approach to the surface of the moon. By pressing various keyboard keys, the vehicle can be rotated right or left, or the main rocket engine fired (to decelerate the vehicle). Fuel is limited, so the player has to carefully manage use of the main engine. Points are scored by landing on various landing zones, but the vehicle must have horizontal and vertical velocity below certain acceptable limits or it is destroyed. Similarly, attempting to land outside a landing zone results in a crash. A successful landing triggers an award of points, and a new landing mission (with landing zones in new, random locations).

Your Assignment:

You are required to bring all the knowledge and skills you have acquired during CS1527 to design and build an updated version of the Lunar Lander game in `pygame`. A basic version (called "**Mars Lander**") should have the following characteristics:

- The player begins with 3 'lives' – i.e. has 3 lander vehicles to use (one after another) before the game ends.

- The lander starts at the top of the screen (assumed to be 1000m above the surface of the planet) with vertical velocity (`veloc_y`) set to a random value between 0.0 and 1.0 m/s and horizontal velocity (`veloc_x`) set to a random value between -1.0 and +1.0 m/s.

- The lander controls are as follows:
  - rotate right [right-arrow key]: rotate the vehicle 1 degree clockwise;
  - rotate left [left-arrow key]: rotate the vehicle 1 degree counter-clockwise;
  - fire main engine [spacebar]: fire the rocket (burning fuel) to counteract gravity.

- Each lander starts with 500 kg of rocket fuel, and this is reduced every time the main engine thruster is fired. 5 kg of thrust is used for each press of the spacebar.

- When the thrust key is pressed, a small image is positioned below the lander to illustrate the rocket thrust is on.

- If the lander flies off the right of the screen it should wrap onto the left side (and vice-versa); the lander should not be permitted to fly off the top of the screen.

- Three landing pads appear at different locations on the screen; pad locations are not restricted to the bottom of the screen (0m altitude) and you could, for example, place a landing pad on top of a mountain (with an altitude > 0 m). Landing outside of a landing pad location causes an immediate crash (lander destroyed).

- A successful landing is defined as one in which the lander is horizontal, has both landing legs on the landing pad (i.e. the lander sprite fully overlaps the pad) and is moving slowly enough. An acceptable landing is defined as one where the lander touches down with horizontal velocity (`veloc_x`) < 5.0 m/s and vertical velocity (`veloc_y`) < 5.0 m/s. A successful landing results in the award of 50 points to the player's score, and the game pauses until a key is pressed – after which a new landing mission starts. Successful landings do not cost one of the player's lives.

- Each lander starts with 0% damage but this immediately increases to 100% (a crash) if the vehicle hits a landing pad too hard. A hard landing is defined as one where the lander touches down on the landing pad with horizontal velocity (`veloc_x`) >= 5.0 m/s and vertical velocity (`veloc_y`) >= 5.0 m/s.

- Crashes cost a player a life and result in a "You Have Crashed!" message being displayed on the game screen. The game pauses until a player presses a key, after which a new landing mission begins.

- During a mission, various instruments display flight data to the player. These are: time (mins:secs) since start of the mission; fuel (kg); damage (%age); altitude (m); x-velocity (m/s); y-velocity (m/s). These are updated continuously in the top-left (instrument panel) region of the main game screen.

- The player's score is recorded (and updated as points are scored) throughout the game; the score is recorded in the region to the right of the "SCORE" label in the instrument panel.

- There is no requirement for sound in this game.

Physics
Although the original Lunar Lander game was often characterised as a 'simulation', in reality it was some way from a true simulation of spaceflight. For the purposes of this assignment you should also not attempt to create a fully realistic working simulation. Instead, it will be sufficient to create an approximate solution.

A few things to note:

- Each time the main rocket engine is activated (each press of spacebar) we need to increase the lander's velocity. To do this we need to know the angle of rotation of the lander, so that the *x* and *y* components of the velocity vector can be calculated. The following equations will be of use:

$$\texttt{veloc\_x}_{i+1} = \texttt{veloc\_x}_i + \mathbf{0.33} \times \sin(\texttt{-angle})$$
$$\texttt{veloc\_y}_{i+1} = \texttt{veloc\_y}_i - \mathbf{0.33} \times \cos(\texttt{angle})$$

  Note: The value of 0.33 shown above is a constant that seems to give a reasonable magnitude to the thrust vector and makes for a playable game. You may want to experiment with this.

- Gravity needs to act on the lander vehicle – pulling it down towards the Martian surface. When the lander is not at rest (landed or crashed) we need to increase the vertical (*y*) component of the velocity by a small amount every game cycle. Acceleration due to gravity on Mars is 3.711 m/s^2, or 0.38*g*, but increasing the *y-velocity* by this amount every game cycle will be too much; in the demo version of the game, the course lecturer found that increasing `veloc_y` by 0.1 m/s was sufficient.

Image Collection
CS1527 is not a graphical design course, and so it is important that you do not spend your time creating graphics and images – but rather work on the design, implementation and testing of your solution. To get you started, download the **`resources.zip`** file from *MyAberdeen*.
This contains the following:

`mars_background.png`	Mars image to use as the background for the game.
`instruments.png`	Basic instrument panel image.
`mars_background_instr.png`	Mars image to use as the background for the game. (includes embedded instrument panel graphic).
`lander.png`	Basic lander (spaceship) image.
`thrust.png`	Image of a rocket flame (used to indicate thrust is on).
`landingPads/`	Folder containing various images of landing pads. [Required for CGS C and above.]
`obstacles/`	Folder containing images of various items (rocks, satellite dishes, pipes, etc.) that can be used as obstacles in the Martian environment (for the player to avoid). [Required for CGS B and above.]
`meteors/`	Folder containing various images of meteors. [Required for CGS A.]

Note: You may need to scale some of the images in the various folders to make them the right size for the game screen.

Demo
Using the materials above and the `pygame` package, one of the course lecturers (Prof Edwards) produced the game demo shown in the screenshot below:

A video showing this solution in use has been uploaded on MyAberdeen (see assessment link).

Evaluation:
Your submission will be marked according to the following ***main*** and ***secondary*** criteria:

Main criterion (*programming and functionality*)

- **CGS D**: The program is executable. The lander can be rotated (left and right arrow keys) and the main engine fired (using the space bar); when thrust is on the rocket flame image is shown. The instrument panel displays the changing time, altitude, fuel reserves, and velocity (x, y). The vehicle behaves appropriately when it encounters the top, left and right sides of the game screen. The lander crashes at the bottom of the screen, displays the appropriate message and starts the next mission. After 3 lives are expended, the game ends.
- **CGS C**: The program fulfils the requirements for CGS D.
  It should also demonstrate: 3 fixed position landing zones and the player can control the lander to either land on one of these or crash – a successful landing awards 50 points to the overall score (displayed) and starts the next landing; random control failures occur (one of the input keys is disabled for 2 secs and an ALERT warning flashed on the instrument panel).
- **CGS B**: The program fulfils all the requirements for CGS C.
  It also demonstrates a minimum of 5 obstacles in the environment (rocks, satellite dishes, pipes, buildings) which act as fixed location sprites and cause 10% damage to the lander if it collides with them. If the lander sustains 100% damage, all the controls are disabled (resulting in a crash).
- **CGS A**: The program fulfils all the requirements for CGS B.
  It should also demonstrate random meteor storms (moving sprites) that cause 25% damage to the lander each time it collides with them; the number of meteors should be a random number between 5 and 10. Meteors disappear if they reach the Martian surface or if they pass off the left or right side of the game screen. If the lander sustains 100% damage (meteors and/or obstacles), all the controls are disabled (resulting in a crash).

Secondary criterion (*documentation and code style*)

Within the CGS band (A – D) determined by the main criterion, your actual CGS mark (A1, A5, B3, C1, D2, etc.) will be determined by the quality of your documentation report and overall coding style (comments, layout, use of object-oriented language features).

Submission Requirements:
By the deadline of **Friday April 27th 2018 at 12:00 noon**, you are required to submit a single compressed archive (ZIP) file (other types of compression files are not accepted) containing the following:

- **Mini-project report**
  A short report (Word document, max 5 sides of A4) containing the following:
  - Summary of system functionality
    (mapping your solution to the main evaluation criteria).
  - UML use case diagram.
  - UML class diagram(s) for any classes you implemented in your solution.
  - Discussion of testing activities.

- **Source code**
  Your complete Python code – suitably documented and organised into modules/packages.

You should name your ZIP file as follows:

```
<surname>_<studentID>.zip
```

If your surname was Einstein and your student ID was 51761234, then the filename would be:

```
einstein_51761234.zip
```

Upload your submission using the CS1527 Assessment/Mini-Project 2 link in *MyAberdeen*.

Appendix: *Resources (Image Collection)*

mars_background_instr.png (pixels: 1200 x 750)

lander.png (pixels: 77 x 60)

thrust.png (pixels: 13 x 15)

LANDING PADS

pad.png (pixels: 158 x 18)

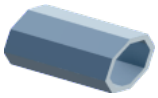
pad_tall.png (pixels: 158 x 82)

OBSTACLES

building_dome.png (pixels: 124 x 58)

building_station_NE.png (pixels: 84 x 107)

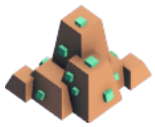
building_station_SW.png (pixels: 92 x 104)



pipe_ramp_NE.png (pixels: 112 x 67)

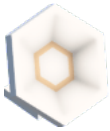
pipe_stand_SE.png (pixels: 113 x 106)

rocks_NW.png (pixels: 112 x 89)

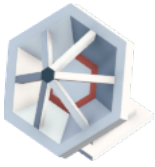


rocks_ore_SW.png (pixels: 111 x 88)

rocks_small_SE.png (pixels: 85 x 90)



satellite_SE.png (pixels: 78 x 92)



satellite_SW.png (pixels: 112 x 119)

METEORS

spaceMeteors_001.png (pixels: 80 x 80)

spaceMeteors_002.png (pixels: 50 x 49)

spaceMeteors_003.png (pixels: 25 x 26)

spaceMeteors_004.png (pixels:12x13)

Video notes:

- game has a single background
- top left corner has the following info:
    * Time: seconds.microseconds
    * Fuel: 500 – 0
    * Damage: 0 – X
    * Altitude: 1000 – 0
    * X Velocity: -X – X
    * Y Velocity: -Y – Y
- bottom has 3 landing pads
- ship moves based on x and y velocity i.e. can move in any direction and draw a curve given time
- Game over and Win cause cause to start over
- Game has a roof and hitting it at speed causes knock back