CS 110Z - Introduction to Computing

Fall 2019

Lunar Lander - 175 Points

Help Policy

AUTHORIZED RESOURCES: **NONE**, with the exception of help received by your assigned partner, any CS110Z instructor, and any Fall 2019 CS110Z course materials and course textbooks.

NOTE:

- Any/all help received or referenced must be clearly and fully documented.
- All submissions will be compared for electronic copying. If found, you will earn zero points for the assignment.
- Undocumented copying will be considered a possible honor violation.
- While you are authorized to receive some help from the limited authorized resources on this assignment, the purpose is for *you* to learn and become a proficient problem solver. You should attempt to work solely with your partner to the maximum extent possible.

Documentation Policy

- The documentation statement must be included as a comment in the top block of your submission.
- You must document all help received or referenced from any source, to include any CS110 instructor or course material.
- Each documentation statement must be specific enough that it explicitly describes **what** assistance was provided, **how** it was used in completing the assignment, and **who** provided the assistance.
- If no help was received or referenced for this assignment, the documentation statement must state "NONE."
- Vague documentation statements must be corrected before the assignment will be graded, and will result in a 10% deduction on the assignment.

Instructions

- You and your partner (if assigned) will create a game from scratch using pythonGraph.
- This assignment is broken into six (6) submissions:
  - 2 x gate checks on your planning and design
  - 1 x online quiz checking your understanding of the requirements on the following pages
  - 2 x gate checks on your implementation progress
  - 1 x final submission
  - Additional information for the first two submissions is provided in this document. Information for the remaining four will be released in future updates to this document.
- **Submit your solutions** via the labs associated with the project turn-in in zyBooks.

Introduction

Lunar Lander was an arcade game released by Atari, Inc. in 1979. This game simulates a moon landing in mountainous terrain, although there were many variants and look-alikes. You will be recreating a simplified version of this game for your CS110 project. You can play a browser version of the original at http://my.ign.com/atari/lunar-lander.

Requirements

Requirements using the word, **Shall**, must be satisfied to receive full points for this project. Those with the word, **May**, indicate additional features and might earn the developer **Bonus Points**, but are not required for full credit.

0. DOCUMENTATION

- a. **Shall** be located in the file header comment for your team's submission.
- b. **Shall** include the names of all team members.
- c. **Shall** contain the documentation statement.
  - **i.** **Shall** be cumulative (i.e., also contain doc statements for all prior gate checks).
- d. **Shall** contain the list of any features for which extra credit is sought.
- e. **Shall** list any shall requirements that are not fully functional and describe the deficiency

1. THE GAME GRAPH WINDOW

- a. **Shall** initially be set to 800 x 600 pixels (width x height). **Note:** Your instructor will change these values and re-run your submission to see if it correctly handles changes to the width and height of the graphics window
- b. **Shall** store and reference the dimension values in appropriately named variables.
- c. **Shall** have a window title consisting of three elements separated by hyphens:
  - i. The course/game information: "CS110Z (F19) Lunar Lander"
  - ii. The assignment identifier: e.g., "GC#3" or "Final Project"
  - iii. The names of the partners: e.g., "C4C Tanya Smith and C4C Fred Jones"

  Example: "CS110Z (F19) Lunar Lander – GC#3 – C4C Tanya Smith and C4C Fred Jones"

2. CONFIGURATION PARAMETERS

- a. **Shall** have a function named "config"
- b. **Shall** define (and use) the following variables to set information about the assignment:
  - **i.** GATE_CHECK: The gate check number with 6 being used for the final project submission.
  - **ii.** PARTNER1: A string in the form <grade firstName lastName> (e.g., "C4C Tanya Smith").
  - **iii.** PARTNER2: Same format as partner1 – empty string if no second team member.
- c. **Shall** define (and use) the following variables in the "config" function
  - **i.** SCREEN_WIDTH = 800 (width of screen, in pixels)
  - **ii.** SCREEN_HEIGHT = 600 (height of screen, in pixels)
- d. **May** define additional configuration parameters AFTER these as needed/desired.

3. DISPLAY ELEMENTS

- a. **Shall** animate the lander's flight.
- b. **Shall** show the player's lander.
  - i. **Shall** use some kind of "lander-like" representation (i.e., not just a box).
- c. **Shall** graphically show and animate thrusters (whether thrust is on or off at a minimum).
- d. **Shall** generate a random landscape (mountains) for each game run.
  - i. **Shall** store information about the height of the mountains in a list.
- e. **Shall** generate a random location for the landing pad for each game run.
- f. **Shall** show a game conclusion message.
  - i. **Shall** indicate if the lander crashed or successfully landed.
  - ii. **Shall** inform the user that a left mouse click will exit the game.

4. **PLAYER CONTROLS**
    a. **Shall** increase the lander's horizontal velocity (positive is to the right) while the left arrow key is pressed and held down (the right thruster is firing).
    b. **Shall** decrease the lander's horizontal velocity (positive is to the right) while the right arrow key is pressed and held down (the left thruster is firing).
    c. **Shall** increase the lander's vertical velocity (positive is up) while the up arrow key is pressed and held down (the main/upward thruster is firing).
    d. **Shall** turn off the associated thrust when the up, left, or right arrow key is released.
    e. **Shall** immediately exit the program (during game play) when the Escape key is pressed.
    f. **Shall** exit the program (after game is over) when left mouse button is pressed.
5. **LANDER MOTION**
    a. **Shall** place lander initially at top of screen at a random horizontal location with zero velocity.
    b. **Shall** implement gravity (downward speed should steadily increase when main thrust not in use).
    c. **Shall** detect when the lander hits a mountain.
    d. **Shall** detect when the lander successfully lands on the landing pad.
        i. **Shall** crash if going too fast vertically while attempting to land.
        ii. **Shall** crash if going too fast horizontally while attempting to land.
        iii. **Shall** crash if the lander is not <u>completely</u> on the landing pad.
    e. **Shall** be allowed to go vertically off the screen (and come back on).
    f. **Shall** return to the left edge of the screen when going off the right (appears to go off the screen to the right and wrap around and reappear on the left) and vice versa.
6. **FUNCTIONS**
    a. **Shall** use a function with parameters to populate the mountains list.
    b. **Shall** use a function with parameters to draw the lander at a specific location to include thrust.
7. **EXTRA CREDIT FEATURES** (up to 10 pts Bonus Points – 1% of your final CS110 grade!):
    a. **May** implement a scoring system (this system might consider fuel consumption). [6 pts]
    b. **May** implement a "HUD" showing relevant information (such as fuel, velocity, thrust, etc.). [10 pts]
    c. **May** implement a leaderboard that displays high scores and allows the user to enter their name and save their score to high scores file. [6 pts]
    d. **May** expand the game to include an enemy UFO that attempts to crash into the lander. [10 pts]
    e. **May** display an introduction screen with game description and instructions. [3 pts]
    f. **May** allow multiple plays in succession without re-running the application. [3 pts]
    g. **May** allow levels of play that increase in difficulty (landing pad gets smaller, gravity stronger, etc). [7 pts]
    h. **May** use images, special graphic effects, and/or audio (like animated explosions when you crash).[3-6 pts]

    The features for which you are seeking extra credit MUST be listed in the comments attached to the Start symbol on the main chart. There are endless options for extra features. Talk to your instructor if you have an idea not listed.

Storyboard

The following is a Storyboard of one version of the game (notice that the landscape and the location of the landing pad are different in each screen capture).

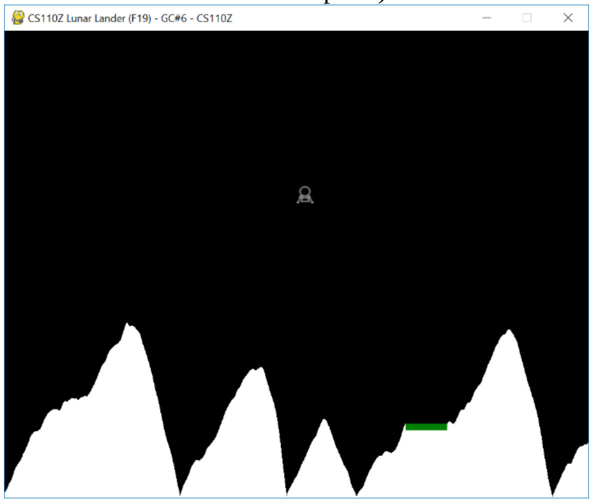


Figure 1 shows the game with a randomly generated landscape, the randomly generated landing pad and the lander. Note: The landing pad is the cyan-colored rectangle.

Figure 1: Lunar Lander – Game Play

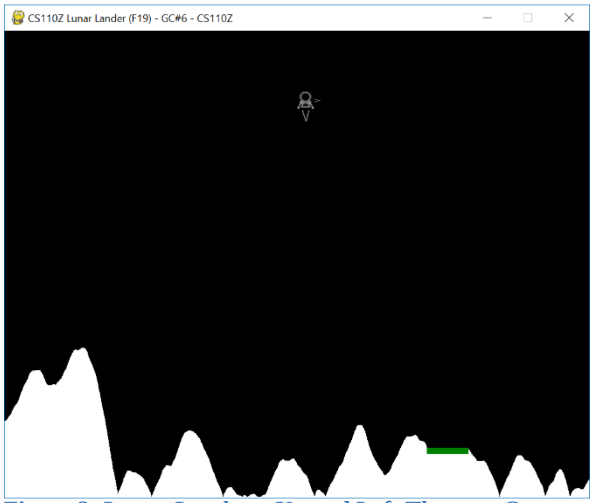
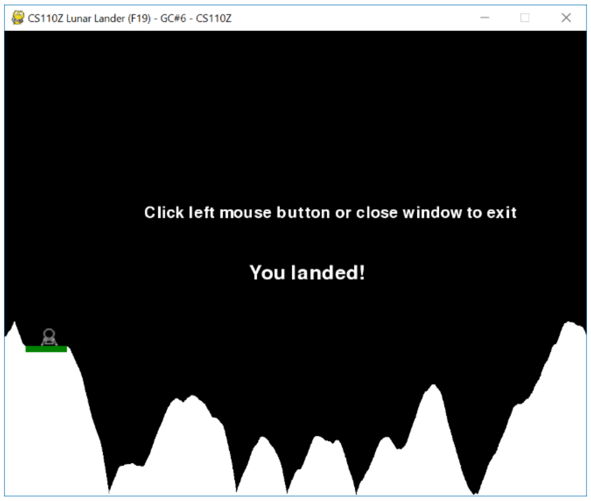


Figure 2 shows the lander in flight **while the user is pressing and holding** the up arrow key and left arrow keys at the same time.

Figure 2: Lunar Lander – Up and Left Thruster On



Click left mouse button or close window to exit

You landed!

Figure 3 shows a successful landing.

Figure 3: Lunar Lander – Successful Landing

Generating Landscape and Landing Pad

You may use any technique you feel appropriate for generating the random landscape and location of the landing pad as long as it meets the requirements in this document. The following is one approach.

First for the mountains, recall that the pythonGraph call for drawing a line is of the following form: `draw_line(x1,y1,x2,y2, color, width)`. To begin generating the mountain landscape you start on the left edge of the graphics window and begin drawing vertical white lines one after the other until you reach the right edge of the graphics window. More specifically you will repeatedly draw a line from the bottom of the screen (where the x1 parameter to the draw_line call is 0, and y1 is WINDOW_HEIGHT) to a randomly generated height (where the x2 parameter is 0 and the y2 parameter is WINDOW_HEIGHT – RANDOMLY_GENERATED_HEIGHT). As you move across the screen your x1 and x2 values increase by 1 for each step across the screen and you successively continue to modify height for the y2 parameter (your y1 parameter remains fixed at 1). You will need to store each height in an list which will be used to detect whether or not the lander has crashed or successfully landed. For example, if the width of graphics window is 800 pixels, the length of the mountains list will be 800, and each element within that list contains a number representing the height, in pixels, of the landscape at that particular x-coordinate location along the horizontal width of the game area. Therefore, `mountains[99]` would refer to the height of the line that is drawn 100 pixels from the left edge of the graphics window.

You will need to experiment with the how you modify height from drawing one line to the next in order to get a nice looking landscape. One method is to use slope in combination with other parameters when changing your height. In the equations below `random` is the Python random function. The `slope` is the amount you will change your `height` from one line to the next. `step_change` is a constant value that when very small results in small changes in slope (smooth rolling hills) and when set to a larger value results in larger changes in slope (jagged mountains). The example solution uses a `step_change` of 0.9. Additionally, you will have to account for height values that get too big (fill up the screen) or too small (off the screen). You will also have to limit your slope in order to get realistic mountains. The example solution doesn't allow a slope outside the range -2.6 to 2.6.

```
height = height + slope
slope = slope + (random.random() * step_change * 2 – step_change)
```

The randomly generated landing pad is a section of your landscape where the height is fixed for the width of the landing pad. Variables to store the location of the landing pad and its width will be useful.

Lander Movement

You may use any technique you feel appropriate for simulating the movement of the lander as long as it meets the requirements in this document. The following is one approach.

The current position of the lander needs to be tracked. This approach uses variables to represent the landers current position (`land_x` and `land_y`). In addition to the current positon, you will also need to track how much change there is in the landers position due to gravity and thrust (`delta_x` and `delta_x`). Your animation loop will need to update these values for each iteration of the loop and also appropriately handle the application of thrust. For example, if the right thruster is currently on and propelling the lander to the left your `delta_x` will go negative and will get increasingly negative during the application of thrust until the user releases the left arrow key. The left thruster will behave similarly to the right except `delta_x` will be positive. The main/up thruster works similarly to the left and right thrusters, but will impact the lander's `delta_y` variable. Gravity is modeled by setting `delta_y` to a negative value initially and slowly increasing `delta_y` due to gravity for each iteration of the animation loop.

```
land_x <- land_x + delta_x
land_y <- land_y + delta_y

delta_x <- delta_x + thrustX
delta_y <- delta_y + thrustY + gravity
```

Landing and Crashing

You may use any technique you feel appropriate for detecting when the lander has crashed or successfully landed as long as it meets the requirements in this document. The following is one approach.

Recall that the variable `land_y` is used to represent the y location of the lander. More specifically it represents the bottom of the landing craft. You have either crashed or landed when the landers y position is equal to or less than the height of the mountains underneath the lander (the landing pad is considered part of the mountains).

Recall that the height of each line used to draw the mountains is stored in an list. The index into this list represents the x coordinate of the line used to draw this portion of the mountain and the y coordinate represents the height of the line used to draw this portion of the mountain. Remember that `mountains[99]` refers to the height of the line that is drawn 100 pixels from the left edge of the graphics window. This list can be used to determine if the lander's position is lower than the mountains (a crash).

For example, if `land_y` were 57 and `land_x` were 120. The lower left corner of the lander would be (57,120). The width of the example solution's lander is 30 pixels. In other words, the area under the lander is defined by the landers x-position plus the width of the lander. Therefore the x values of pixels underneath the lander would range from 57 to 87. If any of the values in the mountain list from index 57 to 87 are equal to or larger than `land_y` the lander has crashed. You will have to compare the landers y position to the values in the mountains list from index 57 to 87, if any value is equal to or larger than `land_y` you have impacted the mountain. Once you have determined that you have impacted the mountain you will have to check to see if the lander is fully on the landing pad and not going too fast. The example solution uses delta_x and delta_y to determine how fast the lander is going. If the absolute value of `delta_x` or `delta_y` is larger than 0.5 or 1.5 respectively the lander crashes even if it was fully on the landing pad when it impacted the mountain.

Gate Check #1 – Structure Chart & Task Descriptions (25 pts)

This project is designed to follow the Four Step Problem Solving Approach we learned in the Algorithmic Reasoning block - Understand, Design, Implement, & Test. In Gate Check #1, you'll create a structure chart, similar to what you have done in class.

A Structure Chart (as described in class) should contain the major tasks you are going to accomplish**.** Each major task may have several more sub-tasks depending on how you design your game. You should plan on **EACH** identified task becoming a function in your finished game; therefore, your Structure Chart is a graphical representation of which functions call which other functions. <u>You should follow the Erase – Move – Draw – Update paradigm for animation as described in class.</u>

In addition, prepare a document that lists each item in your Structure Chart (i.e., for main and each function) and gives a brief (one or two sentence) description of what task that function is responsible for.

<u>Do not proceed beyond Gate Check #1</u> until you receive the approved Gate Check #1 solution. In order to keep teams on track for success (as well as to make grading more tractable), you will be required to base your subsequent project work on the approved solution unless you have your instructor's permission to do otherwise.

Gate Check #1 will be graded on:

a. **(15 pts)** Did you make an appropriate Structure Chart?
b. **(10 pts)** Did you provide reasonable descriptions of the tasks for each element in the Structure Chart?

Only **Shall** requirements need to be addressed.

Gate Check #2 – Requirements Alignment (25 pts)

For Gate Check #2, you will complete an online, 10-qustion randomized quiz hosted on Blackboard, where you and your partner will identify the appropriate task on the structure chart that should implement the specific task described in each **Shall** requirement. Essentially, you are aligning the a subset of the various requirements to their respective tasks, to ensure you and your partner fully **understand** the overall project objective, in addition to specific **Shall** requirements listed above.

The overall objective of Gate Check #2 is to ensure your team reviewed and fully understand this requirements document as well as the various **Shall** requirements that must be implemented in your project solution.

Gate Check #3 – Logic Design (25 pts)

For Gate Check #3, we want you to complete a logical design for several "missing" functions in a given incomplete Logic Design. All functions are taken from the structure chart solution (located in zyBooks under chapter 21 – Project Gate Check 3). Your team's role is to complete the Logic Design by providing an easy to follow "List of Tasks" for each of the five (5) "missing" functions. These list of tasks for each function should include all the steps needed for your game to be easily translated to Python (i.e. order matters, loops must be indicated REPEAT, DO UNTIL etc).

Like the Structure Chart, this Logic Design is non-binding for your subsequent turn-ins. Again, it is OK to make changes and you are encouraged to improve your design as needed. We want you to think about how you are going to approach this problem **before** you implement your solution (i.e. write code). If your solution changes later, that's OK.

In order to standardize submissions for your instructor, you will use an instructor provided Structure Chart solution to Gate Check #1. If you would like to use your Gate Check #1 Structure Chart for Gate Check #3, you must get permission from your instructor first. Submissions that use a non-standard solution without permission may receive zero credit for the assignment.

Gate Check #3 will be graded on:

a. Did you make an appropriate logic design for each of the five (5) "missing" functions? (20 pts)
b. Did you identify the functions that have required input parameters and those functions requiring a loop? (5 pts)

Only **Shall** requirements will be graded. A more detailed grading rubric will be provided to you prior to the Gate Check #3 submission deadline.

Gate Check #4 – Basic Functionality (25 pts)

In Gate Check #4, we are only examining the basic functionality of your game for the following requirements:

- 1. Game Graph Window (2 pts)
- 2. Configuration Parameters (2 pts)
- 3.b. Display Elements - Lander (2 pts) – For GC4 only, a box may be used to represent the lander
- 3.d. Display Elements - Landscape (mountains) (6 pts)
- 3.d.i. Display Elements – Python List (3 pts)
- 3.e. Display Elements - Landing Pad (5 pts)
- 5.a. Lander Motion - Initial Location (2pts)
- 6.a. Functions with Parameters – Drawing Landscape (mountains) (3 pts)

You may implement additional requirements or features for Gate Check #4, but they will not be graded at this time.

Gate Check #4 will be graded by your instructor on these requirements. At your instructor's discretion, you may receive partial credit if a requirement is not fully implemented.

In order to standardize submissions for your instructor, you will use an instructor provided Logic Design solution to Gate Check #3 to start building your functionality unless you have explicit permission from your instructor to do otherwise. Submissions that use a non-standard solution without permission may receive no credit for the assignment.

Gate Check #5 – Advanced Functionality (25 pts)

In Gate Check #5, we are examining a more advanced functionality of your game for the following requirements:

- 3.a. Display Elements – Lander Animation (5 pts)
- 3.b. Display Elements – Lander (1 pt) – Something other than a box used to represent the lander
- 4.a. Player Controls - Right (3 pts)
- 4.b. Player Controls – Left (3 pts)
- 4.c. Player Controls – Up (3pts)
- 4.d. Player Controls – Thrust (2 pts)
- 4.e. Player Controls – <Esc> (1 pt)
- 5.b. Lander Motion – Gravity (7 pts)

Additionally, your project must retain your Gate Check #4 functionality (deductions of up to 5 pts will apply).

Gate Check #5 will be graded by your instructor on these requirements. At your instructor's discretion, you may receive partial credit if a requirement is not fully implemented.

As with Gate Check #4, you may implement additional requirements or features for Gate Check #5, but they will not be graded at this time

Final Submission (50 pts)

Your game should meet all stated **shall** requirements. **If any <u>requirement is not met</u>, it must be so stated in your documentation upon turn-in.**

The following requirements were not previously graded and need to be implemented for the Final Submission:

- 3.c. Display Elements – Thrusters
- 3.f. Display Elements – Conclusion Message
- 4.f. Player Controls – Left Mouse Button
- 5.c. Lander Motion – Landscape (mountains) Crash
- 5.d. Lander Motion – Landing
- 5.e. Lander Motion – Lander Off Top of Screen
- 5.f. Lander Motion – Lander Warp
- 6.b. Functions with Parameters – Drawing Lander

Additionally, your project must retain your previous functionality.

Extra features can be worth up to **10 bonus points – 1% of your entire course grade.** If your team attempted to add any extra features, you must follow the "0.d. Documentation" requirements described on Page 2. Without providing a list of your extra features, your instructor will not know what to test while grading.

A more detailed grading rubric will be provided prior to the Final Submission deadline, but you can expect it to be similar to the previous gate checks.