Mars Lander Report

# Functionality

The game runs @ 30 fps. All the numbers/durations are defined by constants declared in config.py file. The game can be paused at any time by pressing “P” and resumed by pressing Enter. If the game is paused, the current session can be ended, in which case a menu is displayed. From that menu, the player can begin a new game or exit the program completely. This menu is also invoked once the player has run out of lives.

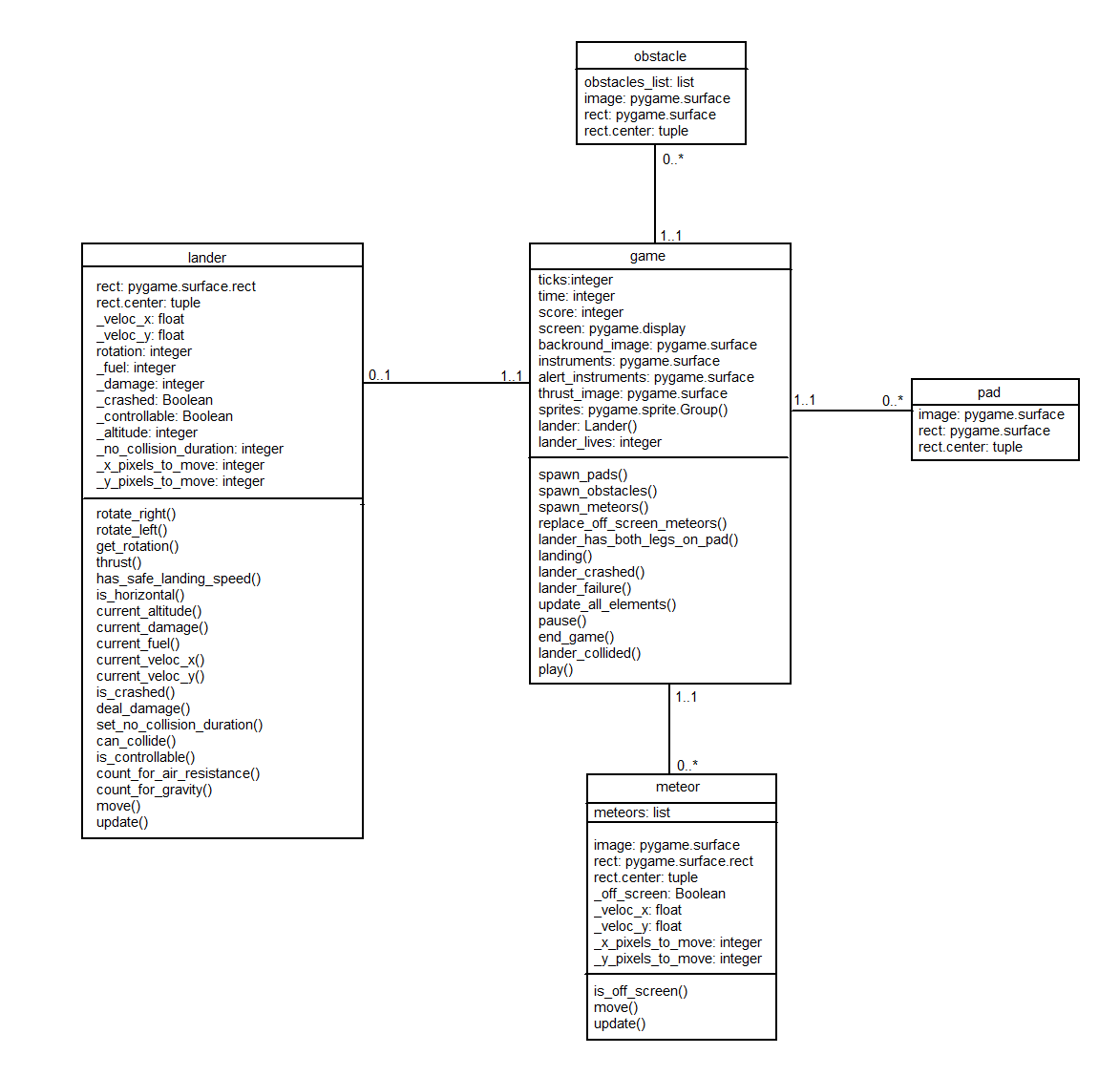
CGS D: The program is executable. The lander can be rotated by 1° on each tick clockwise/counter clockwise, depending on player input. The lander image is rotated without quality degradation. When thrust is on, a flame image is shown beneath the rocket, it rotates simultaneously with the lander. The instrument panel displays all the information necessary and turns red if the lander is damaged. Lives and meteors counters are on top right. If the vehicle hits top of the screen, it sustains some damage and bounces off it. If the lander tries to fly off the screen by going to the sides it instantly warps to the other side. The lander crashes upon reaching bottom of the screen, a message appears. The next mission is started, if the player has at least one remaining life.

CGS C: 3 landing zones randomly spawn on the screen, these can be tall or simply flat. These do not overlap, although it is possible one is above another, rendering the bottom one useless. Upon a successful landing the player is awarded 150 points – lander damage. Afterwards a new mission begins. There is 0.1% chance of failure on every tick – 3% per second. Only one component can be faulty at the time. Failure duration is 2 seconds. Instead of displaying ‘alert’ on the panel, it turns red for the failure duration.

CGS B: There are 5 – 15 randomly spawned objects in every mission. I scaled them, so they are now 25% of the original size. If the player hits an object, the object disappears, the player gets 10% damage and becomes immune to other objects collision for two seconds. ‘NOCOL’ is displayed on the panel for the duration. Once the lander reaches 100% of damage, it becomes uncontrollable and gravity is significantly boosted, so the player does not have to wait long before starting a new mission.

CGS A: At the beginning of each mission a random number of meteors (5-10) is selected. Sizes of their images were scaled down for the sake of playability. The meteors have random horizontal and vertical speeds and can disappear by flying of the screen or upon hitting another object (landing pads/obstacles/player). These then get replaced by a new random meteor spawning on the top of the screen unless the lander was hit. In that case, it suffers damage, becomes briefly immune to collisions and the total number of meteors decreases. In my opinion this balances the game, especially when a bigger number of meteors is selected by the game.

UML



# Testing activities

No functional testing was done on this game due to its relatively small size. All the debugging was done by using the console or print statements. As for the game balance, all the durations, speeds, numbers etc. were finally decided on after extensive playing of the game, because I don’t know of any more formal/better ways to measure game’s playability apart from trying it myself.

The only significant hiccup I experienced was with the instruments’ meters updates. As the number displayed on screen is internally represented by two variables (value, location), I chose dictionary as the appropriate data type. This would get created on every tick, with the callback being a key, and location the value (e.g. {lander.get\_current\_fue() => (100,200)}). For a few hours I could not figure why if there are multiple meters with the same value, only one of them is displayed. When I think about it, the only thing necessary was to swap keys and values, as there are no meters located on a single spot, which would help with debugging if I accidentally selected the same location for two meters. Anyway, after a painful realization I quickly remade it using arrays.

