Lab 2 - 2D game or animation

Laura Mazzuca

22/09/2021

0 Assignments

1. Create 2D demo of a game or animation

1 2D demo: Lunar Lander

The idea was to recreate the Lunar Lander 2D game by implementing:

1. the environment, composed by
   (a) a starry sky background with stars changing position every once in a while
   (b) the moon
   (c) the flags specifying the arrival
2. the ship
   (a) looking somewhat similar to an actual Lander
   (b) a (mostly) physically accurate movement
   (c) a particle system as feedback for the thrusting action
3. the win condition: land in between the flags at a low velocity and small rotation

1.1 Environment

The **starry sky** was implemented by creating a star VAO as a white circle and instantiating it 50 times in random positions. A countdown activates their random displacement.

The **moon** was implemented as a circle with a shade from light grey - rgb(85,85,85) - to white, so as to give a depth feel to it.

The **flags** are made of a line and a triangle. The line is shaded from black, at the bottom, to lincoln green - rgb(36,85,1) - while the triangle has the two vertices attached to the line colored with lincoln green and the one on the extended tip is colored with maximum green - rgb(83,141,34). Again, this was done to give a feeling of depth.

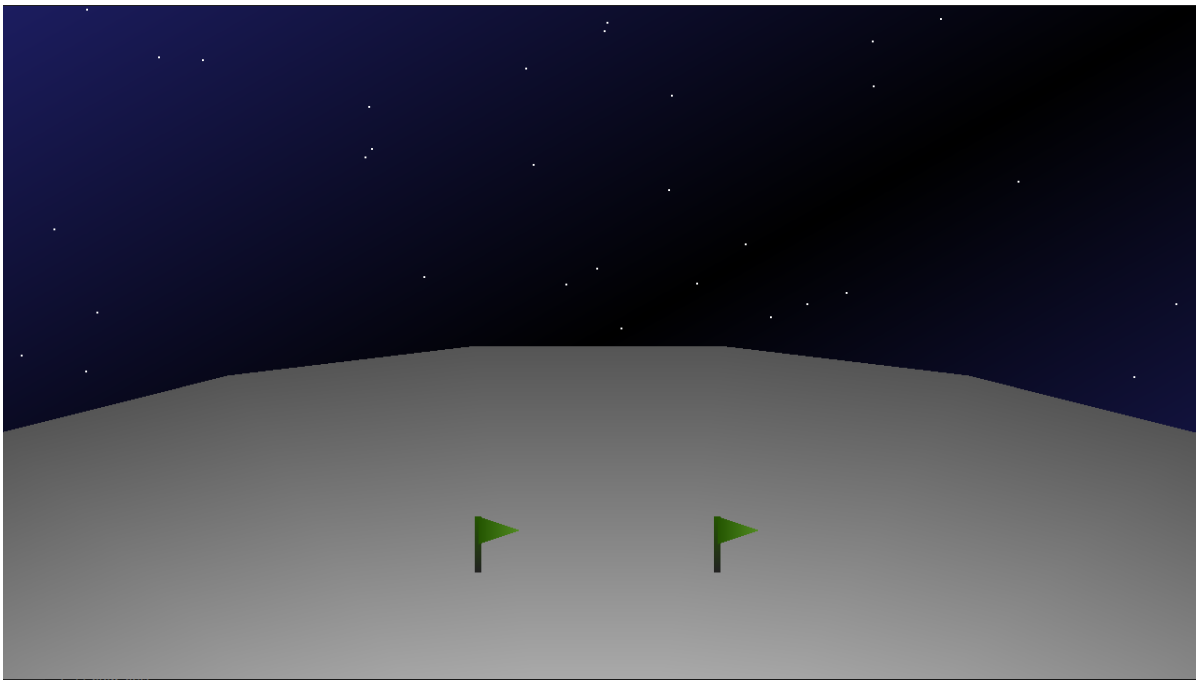


Figure 1: The environment.

1.2 The ship

The **ship** was modeled to vaguely remind of an actual lunar lander so the copper - rgb(184,115,51) - and dark gray color - rgb(51,51,51) - were used for the body and the legs, while a white and dark gray polygon was placed on top of the body.

1.2.1 Physics

The **physics** for the spaceship movement were implemented by approximating Newton's laws of motion. In the *Ship* class (*utils.h*), the following were defined:

- a *gravity* constant variable
- the ship's *position*, *velocity* and *angle*
- a *thrust* constant variable representing the force applied by the ship's engine when active
- a $fuel$ variable, decreased when using thrust and filled back only when empty
- an *update*() function, to update the ship's motion

In the *update*() function, the ship is updated in position depending on the current velocity's values. Then, the new angle is computed by:

1. adding or removing 0.33° depending on the last rotation performed; this was done to have a more challenging gameplay

2. adding or removing 1° depending on player's input (see 1.2.3)

Then, if there is a thrust input, the new velocities are computed as:
$velx- = thrust * sin(angle)$;
$vely+ = thrust * cos(angle)$;
Finally, the y velocity is increased by the gravity.

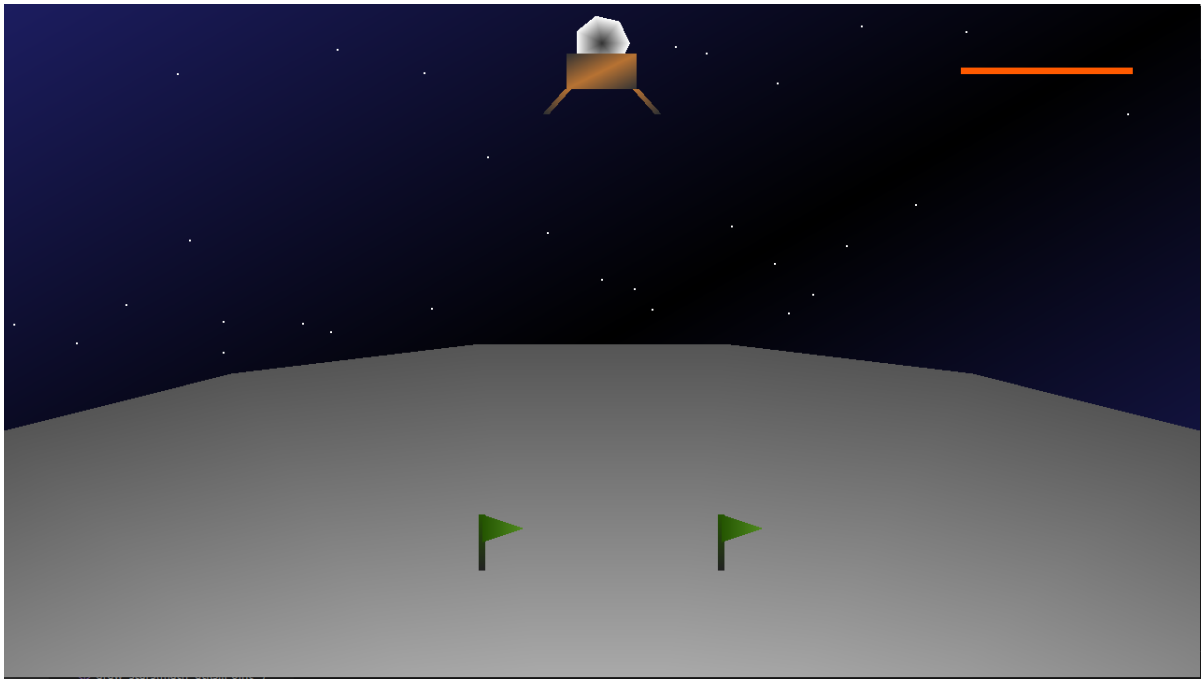
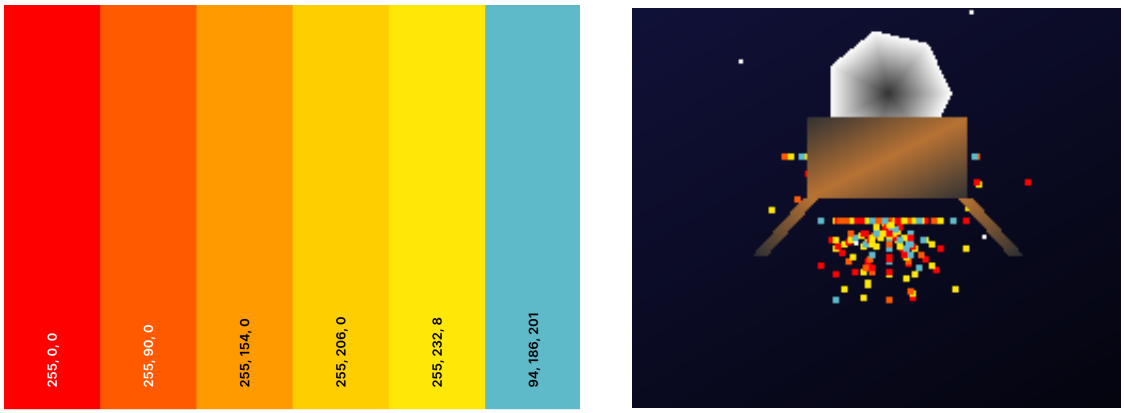


Figure 2: The complete scene at the beginning of the game.

1.2.2 Particle system

The particle system was implemented as a set of 6 points with colors ranging from red to lemon with the addition of fire blue - rgb(94,186,201) -, as in the palette in Figure 3a. These points are registered in the *init*() function and then randomly instantiated during the *drawScene*() depending on the current available positions in the *Sparks* class (*utils.h*).
The positions are computed depending on the position from which they are emitted, which is passed to the *emit*() function along with the amount of particles to generate. Also, the $xFactor$, $yFactor$ and *drag* variables are initialised for each particle, which will set the path each particle will take during the *update*(). Finally, to set the time to live, the *alpha* variable is initialised: when the particle's alpha reaches 0, it gets delted.
The emission is done of 1 particle each time the ship's *update*() registers a push from left or right

and of 10 particles from the bottom when the thrust engine is active.



(a) The palette used for the fire.

(b) All the emit positions active at once.

Figure 3: The visual effect of the particle system.

1.2.3 Player input

The input handler is just the keyboard and the controls mapping is:

A for pushing from the left, thus rotating right

D for pushing from the right, thus rotating left

SPACE for thrusting with the engine

1.3 Win condition

To win, the player has to land the ship so as to have:

- the ship's position between the two flags
- the *ship.vely* variable at a value greater than $-1.0f$
- the *ship.angle* variable in between $\pi/4$ and $-\pi/4$

If at least one of these conditions is not met, the player loses.In any case, the game starts again right after the game ends.
To have a visual feedback of the gameover condition, the shader saved in the second program ID, $programId_1$, is used changing the color to either green if the player won or red if the player lost. This shader flashes the color for a certain amount of time, defined by the *restart*() function, which is activated when the gameover condition is met.