DROIDrun

Libraries
The game is developed with Three.js as a 3D library, Tween.js for animations and OrbitControls to control the camera position.

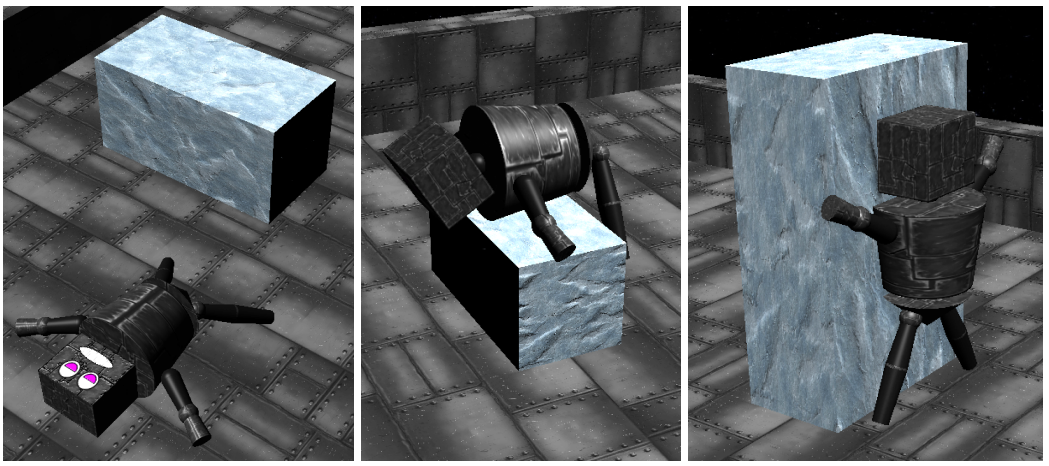
Project files structure
- **game.html** and **gameStyle.css** are the HTML and CSS files of the game page
- The folder **Map** contains all the classes for the environment generation
    - **BlockGenerator.js** to generate the blocks
    - **MapGenerator.js** to generate the floor and the walls of the street
- The folder **Objects** contains the classes that handle the objects (only the robot in this game)
- **CameraHandle.js** is used to control the camera position
- **CollisionManager.js** is used to detect collision between the robot and the blocks
- **KeyBoardHandler.js** handles the user interaction through the keyboard
- **index.html** and **style.css** are the files used for the menu page

Assets
All models in the game are generated by code using Three.js primitives.
The texture of the face is designed by me using Inkscape.
The textures of the floor, walls and robot surfaces are downloaded by awesometextures.
The texture of the blocks are downloaded from TextureCan.

CSS styles of the buttons are generated using cssbuttongenerator

Game
The character (a robot) is on a metal street in the space and it has to reach the maximum distance avoiding ice blocks, there are 3 type of obstacles:

- **Floating blocks**, they are in the air and if the character hits this obstacles he falls with the belly up (first picture)
- **Normal blocks**, they are placed on the floor and if the character hits one of them, he falls on the block (second picture)
- **Wall blocks**, they are walls placed on the floor, if the character hits one of them, he is squashes on it. (third picture)



The user can avoid obstacles going left or right; in case of floating blocks going down, or in case of normal blocks jumping over.

The speed of the run increases during the play in order to make the game more difficult when the traveled distance increases.

It increases in a logarithmic way with an upper limit of 10 units per frame.

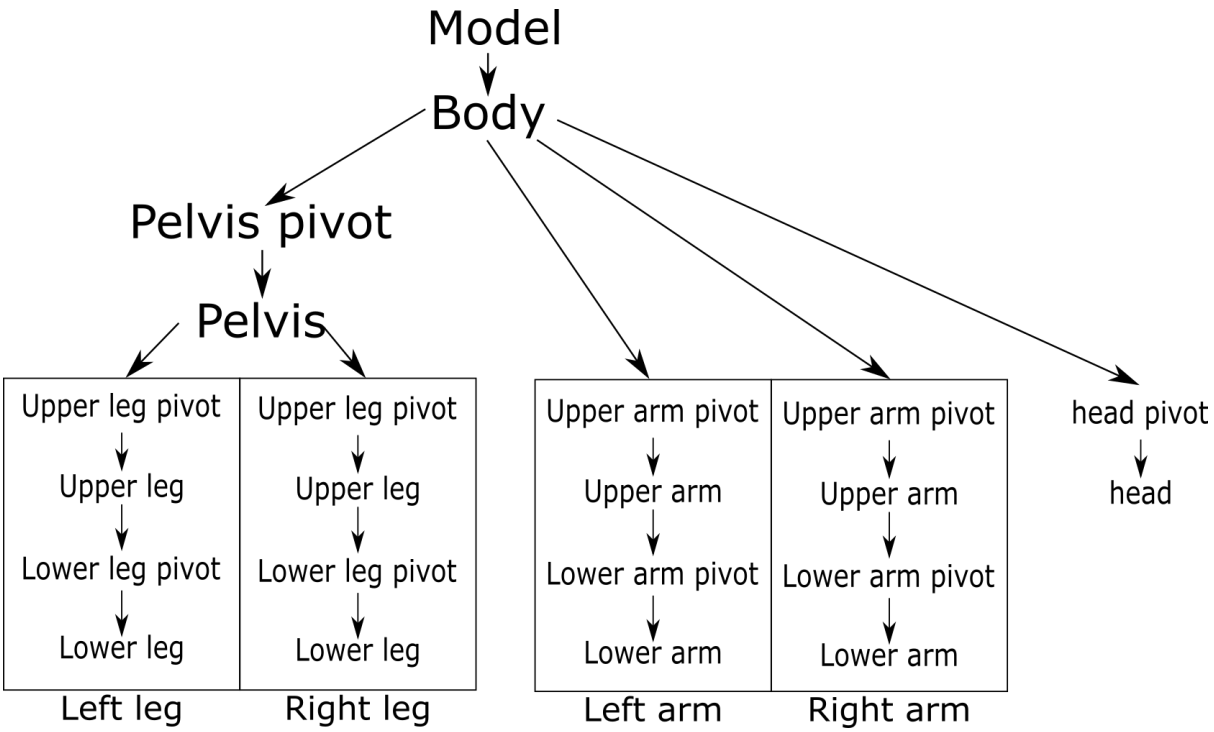
How to play:

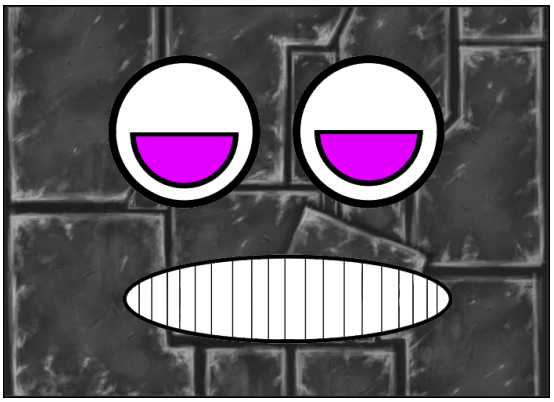
- Press SPACE to jump
- Press A to go left
- Press D to go right
- Press S to go down

Technical description

Character

The character is a robot composed by this hierarchical model.





The surface of the robot is made with materials that have the appearance of metal, some parts of the body has black metal and other parts gray metal.
The face has a specific texture.

The character is controlled by the class Character, in which the hierarchical model and the animations are defined.

Map expansions

The surface in which the robot runs is continuously expanded. It is expanded by steps of 1000 (ThreeJs units), when the character is on the second part of the current surface, another surface of 1000 is generated and attached at the end of the current. In this way the user has the appearance that the street is infinite.

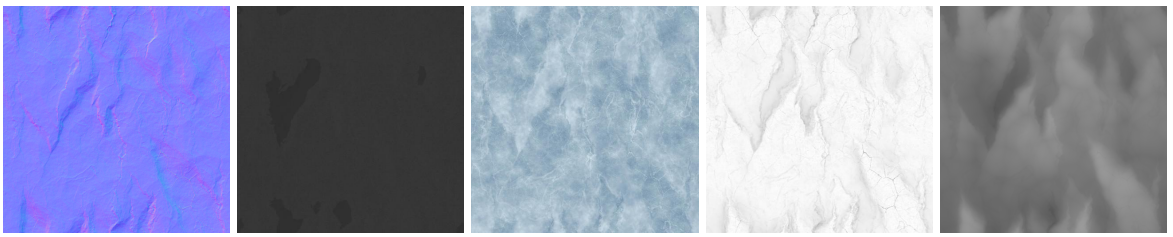
The blocks at the back of the robot are removed to increase performances (more details later)

The map generator and expansion is handled by the class MapGenerator

Blocks
Blocks are randomly generated by the class BlockGenerator, the 3 types of blocks are generated with the same probability.

The material of the blocks has the appearance of ice and it is composed with multiple textures to have a higher level of realism.

In these images the textures of the block material are represented (from the left to the right: normal map, roughness map, color map, ao map, bump map).



Light
There are 3 lights in the scene:

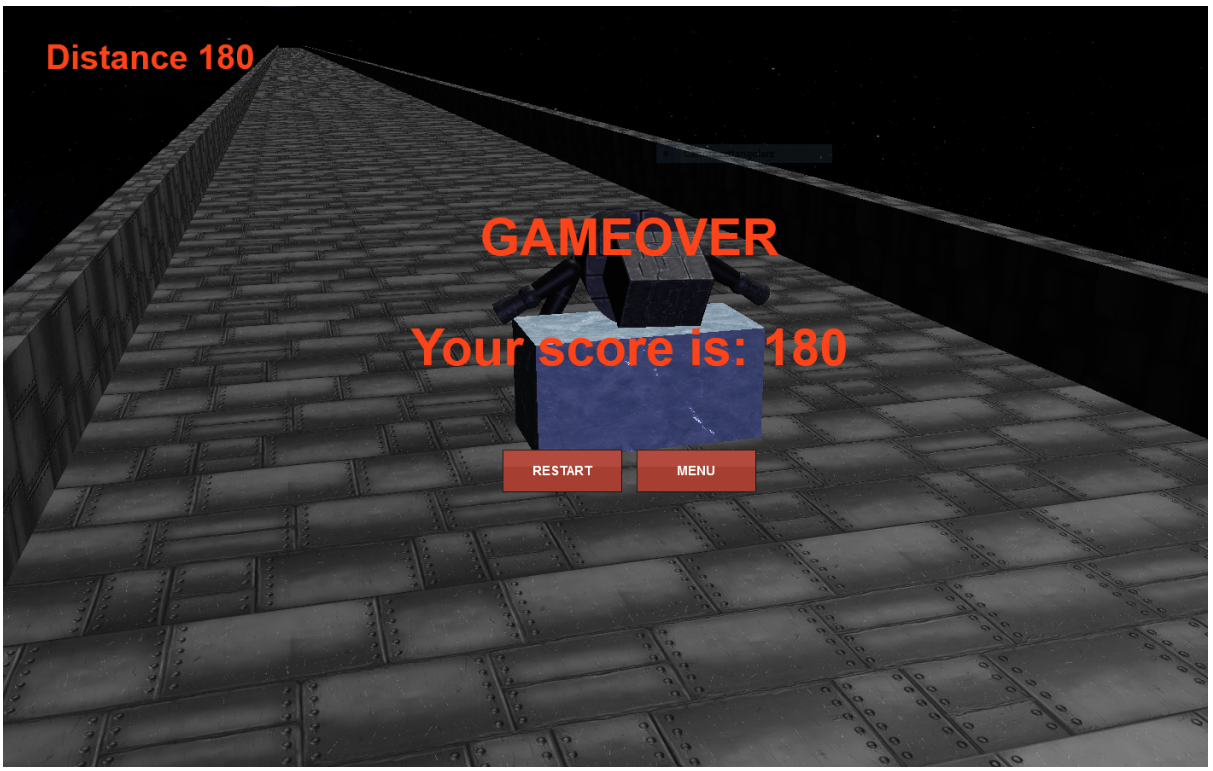
- The ambient light of white color with few intensity 0.05
- A blue directional light in position (0, -100, 5) to have the appearance that it is at the end of the street
- A white directional light on the back of the character to light the scene.

Animation
There are 6 main animations of the character

- **Walk** in which the character walks, the arms and the legs are moved alternately and the head is slightly rotated alternately.
- **Jump** composed by 3 keyframes
    - The character crunches and “loads” the legs to jump.
    - The character jumps.
    - The character lands from the jump
- **Go down**
    - Composed by 2 keyframes, one in which the character goes down, and another one in which he goes up.
- **Fall on the front**, trigger when the character hits a normal block, and the parts of the robot are moved in order to have him crouch on the block.
- **Fall on the wall**, triggered when the character hits a wall black, the robots are moved in order to have a final position in which he is squashed on the wall.
- **Fall over the block**, triggered when the character hits a floating block, the robots are moved in order to have a final position lying on his stomach.

Collision detection
Initially i had used the physics engine Cannon.js, but I had some behaviors of the characters that i don't like due to the gravity simulation, so i have removed the physics engine and implemented a simple collision detector.

All blocks on the scene are buffered in an array and foreach frame, each block is checked if it is hitting the robot.
To avoid a performance degradation when the block number increases, the blocks that are on the back of the robot are removed, in this way there are always few blocks to check.



In this image it is possible to see that all the blocks at the back of the character have been removed.

To compute the collisions the robot shape is approximated by a box that is rotated and translated among the robot movements.
The collision detector works fine, but sometimes some parts of the robot pass through the blocks without detecting the collision. But if the character hits the block well, the collision is always detected.

LOADING SCREEN
The game needs some time to download all assets, so when the game is opened there is a loading screen created using THREE.LoadingManager.