# Lawn Mower

In our second submission we implemented following aspects:

## Controls:

**W/S:** move robot forwards/backwards

**A/D**: turn left/right

**Left Mouse:** Dragging lets the player look around the robot

**Scroll Wheel:** Zoom the camera in or out

**F1:** Toggle Wireframe

**F2:** Toggle Culling

**F3:** Toggle HUD

**F4:** Pause/Unpause the game, during pause, you may freely fly around with a free-move camera.

**ESC:** Exit the game

## Gameplay:

- 3D-Geometry: The lawn mower is loaded from an obj. file with assimp. It consists out of the roboter body and its wheels. The implementation can be found in the class ObjectLoader. We also implemented some basic 3D-objects. Like a ball, walls and a ground. The objects are implemented using vertices, indices, uv-coordinates and normal.
- Playability: We implemented a lawn mower, who can move around and mow grass. The lawn mower can move by pressing the keys "wasd". The grass, which is represented by flat cubes, are removed when the robot touches them. When all grass fields are removed the game ends.
- Win/Lose Condition: This is implemented by stopping the game when all grass fields are removed. It is checked if there are any grass fields left. If there are no grass fields in the scene then the game ends by not allowing the player to drive around anymore and zooming out so the whole playing field is visible. Furthermore, there is a time limit. When the time is up and the grass is not mowed than the player looses the game.
- Intuitive Controls: The lawn mower can be moved by pressing the keys "WASD". When a key is hold, the lawn mower speeds up.

- Intuitive Camera: We implemented a camera that follows the lawn mower around and always points in that direction in which the robot is driving. When pressing Left Mouse one can change the view around the mower. The camera snaps back in position when Left Mouse is released.
- Illumination Model: We implemented one point light and one directional light, to illuminate the scene. All objects in our game have materials. The one Point light is always above the robot.
- Textures: We implemented different textures for different objects. For example we use “grassTexture” for the grass-planes. We also made sure to make the textures tileable
- Moving Objects: We implemented a lawn mower as our moving object. It can be moved around. Also a ball is moving around, this is achieved with PhysX.
- Collision Detection: We implemented collision detection by using PhysX. Therefore we implemented “TriggerShapes”, these are triggered when the lawn mower touches them. When the lawn mower drives into the wall, a collision is detected and the robot stops. Moreover a collision is detected when the robot touches a grass-plane. The grass-plane is then removed. The lawn mower is implemented as a Character Controller, which has for example a density and a “contact offset”.
- Advanced Physics: A ball is implemented with PhysX. It falls to the ground because of gravity. Then a force is applied so the ball is moving from one side of the field to the other and back. When the lawn mower hits the ball, the ball is shot away.
- Heads-Up Display: A timer is shown as a 2D overlay. The speed down to the right and the FPS and Framecounter in the bottom left corner
- The cubes representing patches of grass are drawn instanced. (GrassField class)

## Effects:

- CPU Particle System: The smoke that comes out of the exhaust is made with (billboard) particles. We implemented the effect on the CPU side. Each particle has a certain age position and velocity, we draw on Quad instanced with different alpha settings to achieve a fadeout effect. On creation and on each update, the velocity of each particle is changed by a bit. The MVP-Matrix is modified in the Fragment Shader so that the quads always face the camera but keep their translation.

- Hierarchical Animation: The lawn mower consists out of a roboter body and four wheels. The wheels are rotating while moving with the roboter body. This is achieved with a parent child relationship. The roboter body is the parent and the wheels are the childs. This is implemented in the Main function under “Calc Player Position”. Furthermore, the body randomly changes its upwards position by a small amount each frame to simulate the motor running.

To understand how hierarchical modelling worked we read the article: http://morpheo.inrialpes.fr/~franco/3dgraphics/practical3.html. Based on that we build a parent child relationship between our roboter body and the wheels.

- Cel Shading: The Cel Shading is implemented in the shader texture.vert/texture.frag. There a function cel is implemented. The steps of brightness are set to 5.

  For Cel Shading we read this writing on Cel Shading: https://www.cs.rpi.edu/~cutler/classes/advancedgraphics/S12/final_projects/hutchins_kim.pdf. We integrated this approach in our existing shader.

- Contours via Edge Detection: To achieve this, a framebuffer is implemented for postprocessing affects. In the shader edge.vert/edge.frag an edge detection filter is implemented. We used this framebuffer/postprocessing tutorial to achieve this. https://learnopengl.com/Advanced-OpenGL/Framebuffers. Our extensions are that we implemented the shader to detect contours.

## Libraries:

We used PhysX for collision detection. https://github.com/NVIDIAGameWorks/PhysX
Assimp is used to load the objects. https://github.com/assimp/assimp/releases/tag/v4.0.1/
freetype to load textures from fonts. https://www.freetype.org/