Miracle Knight

By David Zúñiga Navarro and José Daniel Casado Aguilar

Turn-grid strategy game. Eliminate all the red units before they eliminate you!

# **Technical features**

### **Particle Emitter** (Animation)

A previously defined number of game objects are instantiated in an array, but are not drawn. Afterwards an “emitter” object generates trajectories for these objects that depend on certain parameters such as how many particles per second, and how much lifetime each will have.

This emitter then calculates the position in which every particle should be and then makes the respective draw calls.

Whenever a unit attacks another, a barrage of particles comes out of the target simulating sparks of a clash between sword and shield.

### **Fog** (Animation)

### We added an effect that makes things far away clouded by fog. We accomplished this via the fragment shader by combining the output color with grey proportional to the distance from the camera to the object that is being rendered. The fog is activated only when the camera is below a certain height and can be toggled by a uniform variable.

### **Toon outlining** (Graphics 2)

We added an outline to some of the objects in our scene. That way they get highlighted and it is easier to spot them. The color and width also change to differentiate between player and enemy units, also to highlight the currently selected unit.

To identify the pixels that are part of an outline, we stored the unique ID of the object that is drawn in each pixel on an additional FBO. In a deferred pass, we check the pixels around the current fragment. If there are any with different Object IDs, we paint that pixel as a border.

### **Negative colors** (Graphics 2)

The player has an ability to turn back time and redo their actions. When this power is activated, the colors of the scene are turned to their negative counterpart in a growing sphere. And the sphere shrinks when the power is deactivated.

To apply the effect to only the pixels in the circle, we draw a sphere on the stencil buffer. On the second pass we sample that buffer. If it’s not blank, the corresponding pixel has their colors inverted.

### **Toon lighting** (Algorithms and Gems)

We changed the way in which the objects in the scene are lit, to make a crisper border between shadow and light. The steps we took are the following:

1. In the fragment shader, we calculate how much the current fragment is being lit by the light sources in the scene. We get a number from 0 to 1, we call it the light power.
2. We sample a texture using this light power as the X coordinate. This texture (*show the texture in the video)* is a greyscale gradient.
3. We use the value sampled from this greyscale to obscure the output color of the current fragment to create the toon-like illumination.

The gradient texture can be designed to create the light to shadow transition as crisp or smooth as we want. We also made it so the objects don’t obscure completely when no light is hitting them so we had a clearer scene overall.

### **Instance Rendering** (Algorithms and Gems)

There are a lot of identical objects in our scene: terrain blocks, trees and rocks. To make a more efficient use of resources we store the position, rotation and scale of each of these objects and make a call that draws the same mesh multiple times, applying the transformations that we previously stored. That way we can render very big maps (4900 tiles) at a playable framerate.