

# CGUE18 Rest in Pepperoni - Documentation

---

Philip Bugnar (1402617), Johannes Eschner (01633402)

## Description

"Rest in Pepperoni" is a 3D-Platformer. The goal is to navigate a Pepperoni through a gauntlet of dangerous kitchen tools.

The game is about Al Peppino the Pepperoni who wants to be the superb finish for a freshly made Pizza Diavolo. He wants to be the centerpiece of the pizza.

The other kitchen devices however have decided that he should be chopped up and scattered around the whole pizza.

But Al Peppino wants to get there in one piece. He wants to be the center of attention, the marvellous finish for this delicious Pizza Diavolo. Your goal as a player is to help Al Peppino reach the pizza without getting chopped up by dangerous knives, drowned in stormy sinks or burnt by angry stoves.

In order to do this, Al Peppino has to utilize his considerable set of abilities, including running, jumping, strafing and standing.

## Controls

Key	Action
W	Move Forward
A	Strafe Left
D	Strafe Right
S	Move Backwards
Space	Jump, Double Jump (in mid-air or after touching a wall)
Escape	Quit Game
F2	Debug HUD
F3	Wireframe
F4	Toggle Outlines
F5	Toggle Toon-Shading
F6	Toggle Backface-Culling
F8	Toggle View Frustum Culling
F11	Restart (After Game Over)
V	Infinite Jump (For Testing/Debugging)
Tab	Switch between Free/POV-Camera

You control Al Peppino with the keys listed in the table above. W accelerates Al Peppino, while letting go of W stops him. Pressing S makes Peppino go backwards. You can use D and A to avoid obstacles by strafing left or right. Another way of avoiding harm would be using Space to jump. Since Al Peppino is no ordinary Pepperoni, he can also perform a double jump by pressing Space again in mid-air and a wall jump after hitting a wall. Escape allows you to quit the game.

## Features & Walkthrough

The Features of the game will be explained alongside a walkthrough.

When the game starts, Peppino spawns at the start of the gauntlet. The timer on the upper left left corner of the screen indicates how much time one has left to reach the pizza. One can also use the F-Keys mentioned in the table above to toggle various effects. Every static object in the scene was modeled with Blender and imported using Assimp.

The goal is to reach the pizza at the end of the gauntlet before the timer runs out and without getting hit by any of the obstacles.

The first obstacle the player has to face are kitchen knives, which are thrown using PhysX. Touching a knife will result in death - remember, the goal was to reach the pizza as a whole!

After one has successfully avoided the knives, the next goal is to get to the table. this can either be done by utilizing Peppinos double-jump ability with the proper timing or by landing on the broomstick and then jumping directly onto the table. Falling onto the floor will result in death - no one wants a dirty Pepperoni on his pizza!

Note: if one only misses a jump by a little bit, wall jumps can be used to climb back up. However this is a considerable time-loss.

After passing the books one can either use a properly-timed double jump or the chair to get back to the kitchen counter. The next obstacle is the kitchen sink. The water inside the sink is created with a procedural texture. Falling into the water will result in death - pepperonis dont make good swimmers!

After passing the water and jumping across to the second counter, the player has to get past the stove. The fire on the stove is created using a GPU-Particle System. The fire switches randomly between the hotplates every few seconds. Touching a flame will result in death - burnt pepperonis are bad pepperonis!

The final part of the gauntlet consists of four chairs and can be tackled in one of two ways. The slower but easier way is to jump across the seats and then walljump your way up the books. From there you can easily reach the pizza.

The faster but harder option is to directly jump onto the chair lean of the second chair. From there one can jump from chair lean to chair lean and then from the books onto the pizza.

Congratulations for making it this far!

Your best run will be saved as a highscore - your score is equal to the time remaining on the clock. The faster you are the higher your score will be. If you want to reset the highscore, just open the file

`assets/score.txt` and replace the current score with a 0.

## Illumination

Since we use cel-shading we chose to only illuminate the scene using a directional light coming from in front of the character. All regular objects in the scene are textured and use the `assets/shader/texture.*` shaders and light-mapped objects use the `assets/shader/texture_lm.frag` fragment shader.

## Effects

### Light-Mapping (Separate Textures), 0.5 Points

The light-mapping was done by baking the lightmaps in Blender and then loading them as separate textures for all the models using light-mapping. If a model uses light-mapping a different material is applied to it. If an object is light-mapped it will use the `assets/shader/texture_lm.frag` shader. In this shader the values of the lightmap are added to the brightness values of the objects color texture, resulting in shadows at the desired places.

### Heads-Up-Display (compulsory)

The HUD uses the text rendering library FreeType. The text is rendered above all other framebuffers in orthographic projection. For each character a quad is rendered and alpha blending is used to only show the character. Apart from debug options (Frametime, FPS, Number of objects), the HUD is also used to display the remaining time as well as the win and loose messages. With the F2 key the additional debug info display can be toggled. For the implementation the following tutorial was used as a basis: <https://learnopengl.com/In-Practice/Text-Rendering>

## View Frustum Culling

View frustum culling was implemented using axes-aligned bounding boxes. For each geometry object a bounding box is created upon initialization. All transformations of the object are also applied to the AABB. Before drawing each objects visibility in the view frustum is tested. The culling is only implemented in the pov player camera, allowing the debug camera to show the result of the culling (Objects will vanish depending on the pov cameras position). This feature was implemented following this tutorial: <http://www.lighthouse3d.com/tutorials/view-frustum-culling/>

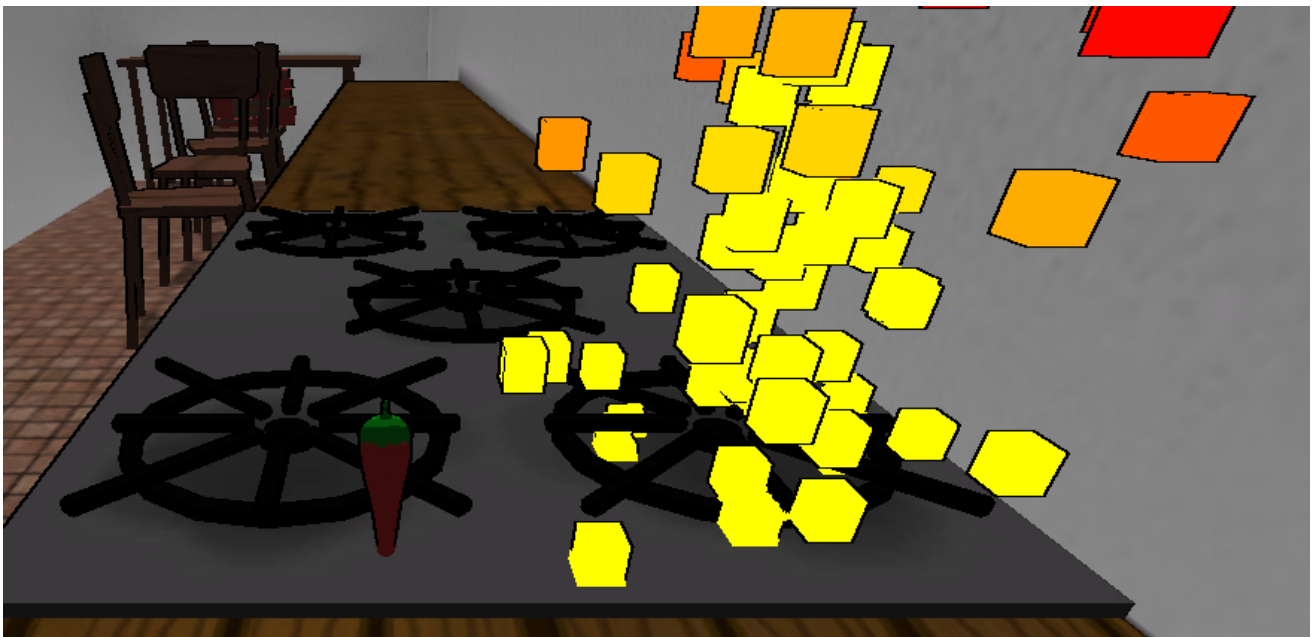
## Physics-Engine (compulsory)

The Physics Engine used for this Assignment was the Nvidia PhysX Engine.

## GPU-Particle System (+Compute Shader,Instancing), 1 Point

The GPU-Particle System was used to create the fire on the stove. The corresponding code can be found in under `rest_in_pepperoni/src/Particle_System/ParticleSystem.cpp` and `assets/shader/particle.*`

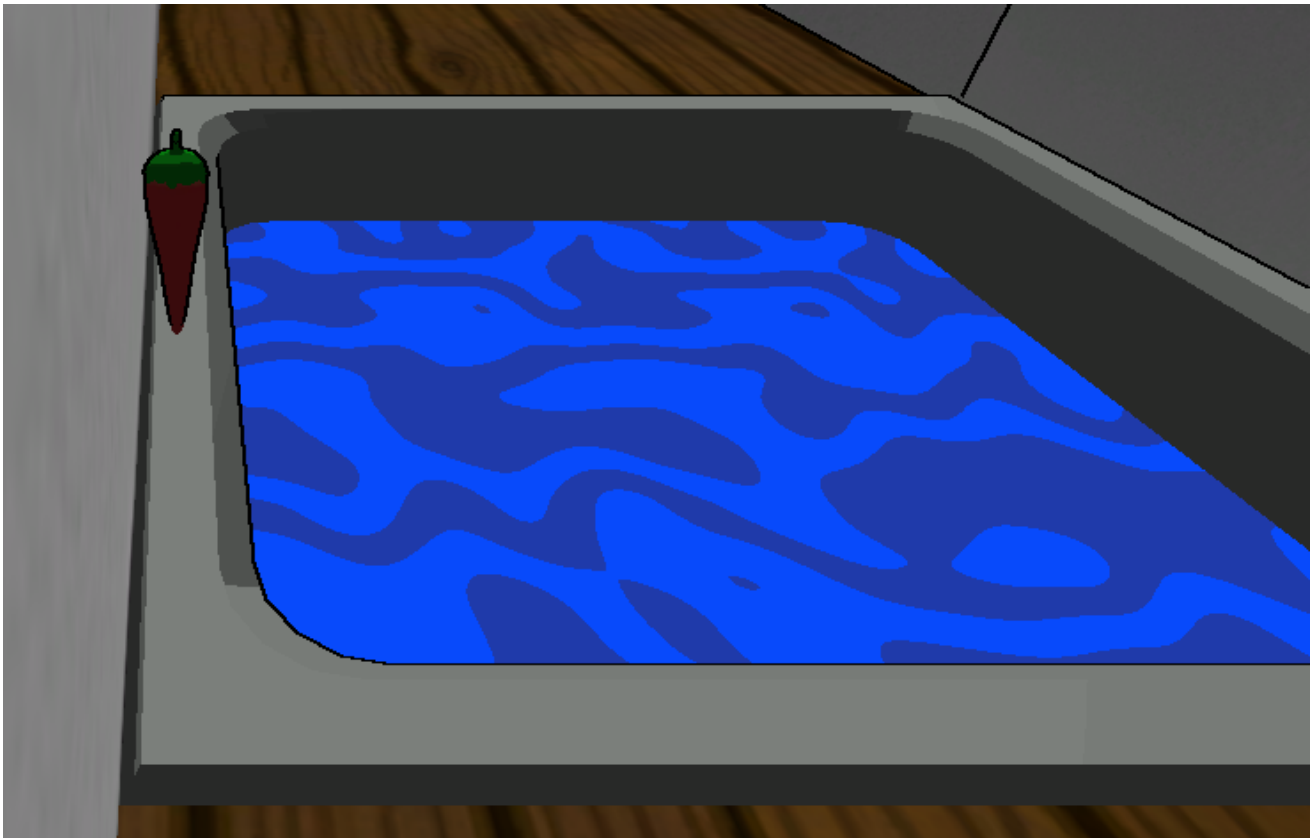
Sources: Slides from the CG-Tutorial regarding this topic.



## Procedural Textures, 1 Point

Procedural Textures were used to create the water in the sink. the corresponding code can be found under `rest_in_pepperoni/src/ProceduralTexture.cpp` and `assets/shader/procedural.*`

Sources: <http://www.upvector.com/?section=Tutorials&subsection=Intro%20to%20Procedural%20Textures> and <https://thebookofshaders.com/>



## Cel-Shading (+ Edge Detection) 1,5 Points

The cel-shading was implemented in the fragment shader `assets/shader/texture.frag`. Three different brightness levels are used to classify the colors in different classes. For implementing this feature the following tutorial was used as reference: <https://www.youtube.com/watch?v=dzItGHyteng>

In addition to the cel-shading we also added outlines to all our objects using an edge detection filter. For this feature the scene is initially drawn on a framebuffer object. On the depth buffer of the FBO we used a sobel edge detection filter to extract the edges. The image from the framebuffer is then drawn onto a quad filling the whole screen. In this drawing step all the pixels where an edge was detected are drawn in black, while all other pixels are drawn in the respective color of the FBO's color buffer.

For the implementation of the framebuffer the following tutorial was used as a guideline: <https://learnopengl.com/Advanced-OpenGL/Framebuffers>

## Tools & Libraries

All complex Models were created with Blender. <sup>1</sup>

Assimp <sup>2</sup> was used for importing the Models.

FreeType <sup>3</sup> was used for text rendering.

## Sources

---

1. Blender, [link](#)

2. Assimp, [link](#)

3. FreeType, [link](#)