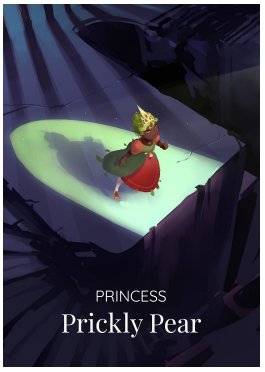
Princess Prickly Pear



Submission 2

- Eva Jobst: 51824341
- Klaus Galler: 01226155

A (well known) captured princess does not want to wait for her rescuer anymore and decides to break out of her prison. A Jump & Run Escape Game.

Implementation & Requirements

Compulsory & Optional Gameplay

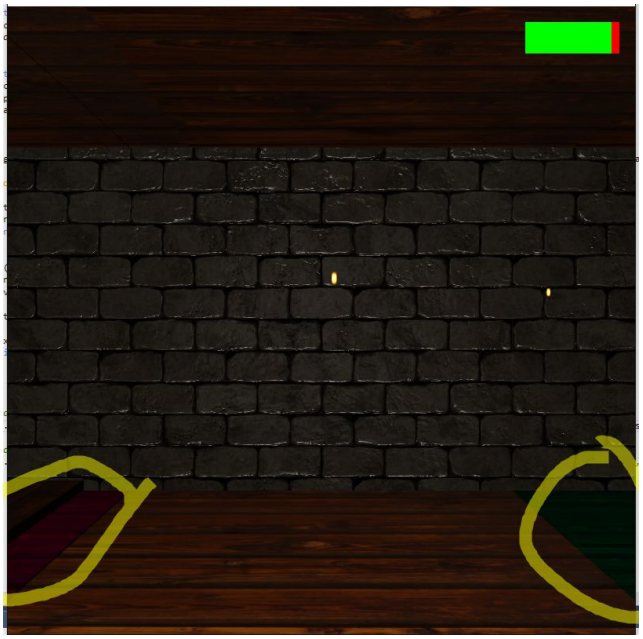
Requirement	How implemented	max. Points	Done by
Playable	One level, with labyrinth included from file through model loader. Basic controls and actions to move character and overcome obstacles to win or lose the game.	11	Klaus + Eva
3D Geometry	Model Loader can load all kind of objects. It is used to load the labyrinth made with blender (collection of simpler objects), but can also load “real” 3D. Tested with dose.obj - can be tested by changing file in modelloader or by changing name of dose.obj to level1.obj	6	Klaus
Win/Lose	Win-Condition through colliding with goal. Lose-Condition through colliding with trapdoor/cart/net. Additional lose-condition once timer runs out.	3	Eva
Intuitive Controls	Moving with Arrow Keys. Polling for continuous input. Framerate independence through calculation of time delta → additional factor in translation/rotation/jump of camera and translation of hidden door; also in position-calculation in compute shader for particle.	2	Eva

Intuitive Camera	First person camera, which can be controlled by the player with arrow keys to move their character.	2	Eva
Textures	Textures are currently applied on trapdoor, goal, net, walls, cart, candle, candle particle and hidden wall.	2	Eva
Moving Objects	In the game the following objects, provided by the model loader, can move: cart, net & hidden door. The trapdoor is created with rectangles and moveable as well.	2	Eva
Documentation	This document. Code documentation as comments in source code.	1	Klaus + Eva
Adjustable Parameters	Config file from ECG Framework used. Change of brightness through postprocessing stage with values ranging from -1 (very dark) to 1 (very bright). **Sources:** https://learnopengl.com/Advanced-OpenGL/Framebuffers (Vertex & Fragment Shader) http://wes-uoit-comp-graphics.blogspot.com/2013/04/post-processing-levels-brightness.html (Brightness calculation as a post-processing stage)	1	Eva
HUD	Used to show timer on the right top corner. Can be (de)activated with F2. **Sources:** https://learnopengl.com/In-Practice/2D-Game/Rendering-Sprites http://www.mbsoftworks.sk/tutorials/opengl4/009-orthographic-2D-projection/	4	Eva
		34	

Effects

Requirement	How implemented	max. Points	Done by
Simple Normal Mapping	Applied on the walls and hidden wall with a brick-wall texture. Can be turned on/off with F1	4	Eva

	key. **Sources:** https://learnopengl.com/Advanced-Lighting/Normal-Mapping (For shader calculation; the texture.frag from the ECG framework was basis)		
GPU Particles using Compute Shader	GPU Particle System is used for the candle light. It uses the compute shader to calculate the time-to-live and position of the particle. This is used in combination with a geometry shader -> vertex shader **Sources:** ● http://www.planet-source-code.com/vb/scripts/ShowCode.asp?txtCodeId=5660&lngWId=3 (Candle light calculation) ● https://github.com/Crisspl/GPU-particle-system (Using compute/geometry shaders and their interaction) ● GPU Particle Systems Slides from **Repetitorium 2018** (Using compute/geometry shaders and their interaction)	12	Eva
Hierarchical Animation	The cart-object uses hierarchical animation. When the model is loaded a parent-child hierarchy is created. Once the animation of the cart is triggered the parent element starts to move forward and simultaneously the wheels rotate.	4	Klaus + Eva
Lightmaps using separate textures	This was a bit tricky because for debugging reasons we used *.obj which does not support multiple texture layers. So the level was once exported with all reusable textures and a second time with a single lightmap that is used by all meshes. Ingame the two *.obj are merged and both textures with both individual UVs are set for the shader. The shader then simply combines the correct texture and lightmap pixel. Because we colorized the blender materials of certain points (as the colors help us for collision detection) this feature can be checked by looking on the ground: using the same textures the colors of certain areas are lit different	8	Klaus

	because of the lightmap 		
		28	

Features

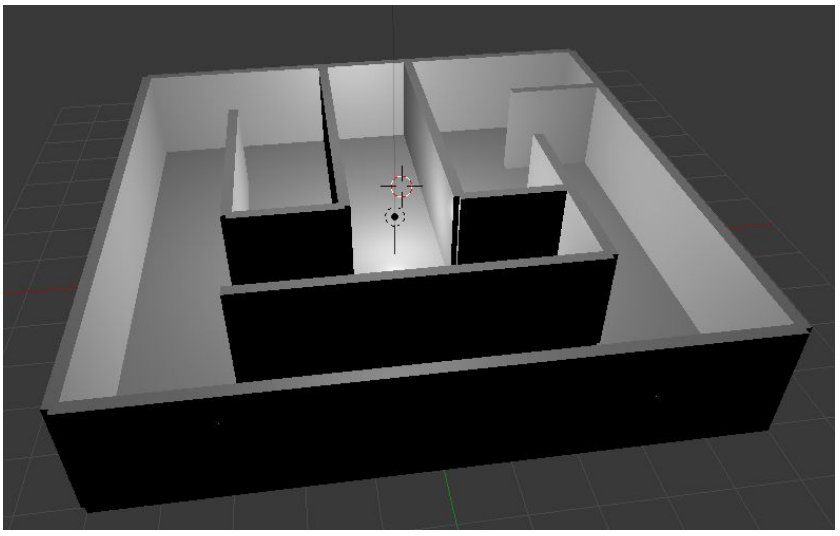
Controls

Key	Effect
Arrow Keys	Forward, back, turning around
Space	Jumping
S	Trigger
ESC	Quit Game
S + Arrow Key “Forward”	Fast running
F1	Turn on/off Simple Normal Mapping
F2	Turn on/off HUD

Labyrinth

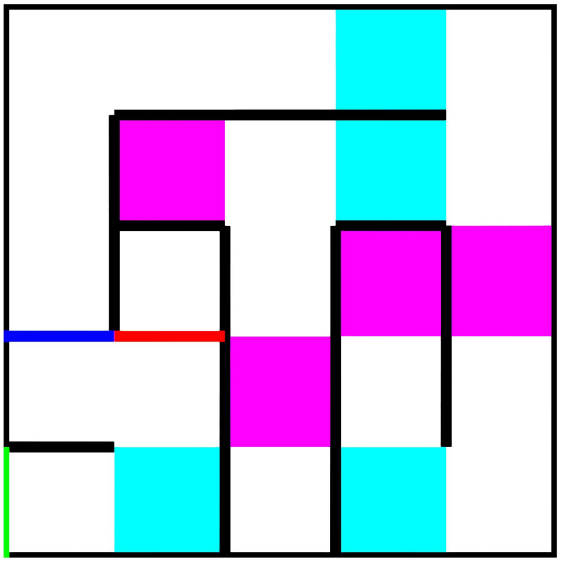
The Labyrinth is made with blender and essentially contains a 5x5 grid floor, 6x6 grid walls and 5x5 grid ceiling. Every single Mesh inside the Labyrinth can be removed or added

quickly to ensure a simple way to swap them with other objects (trap doors, doorways instead of walls, e.g.), making different levels more easy and for adding textures individually to each Mesh.
(loading Textures with ModelLoader is already working (and can be tested by loading wuerfel.obj into the game) and Textures are integrated into the level, which use Simple Normal Mapping.



Camera

First person camera, which can be controlled by the player with arrow keys to move their character.



Collision Detection

Collision Detection is realized with per-pixel collision. A compute shader loads the map (displayed on the right) and reads the color of the pixel that maps to the players current position. The shader is provided with the players future position and overwrites a buffer integer with the enum representation of the found pixel color. The image is generated using diffuse colors and orthogonal projection in Blender. The objects are color-coded as follows:

- **Wall:** Black
- **Trapdoor:** Magenta
- **Net:** Turquoise
- **Goal:** Green
- **Hidden door** (before animation)**:** Red
- **Hidden door** (after animation)**:** Blue

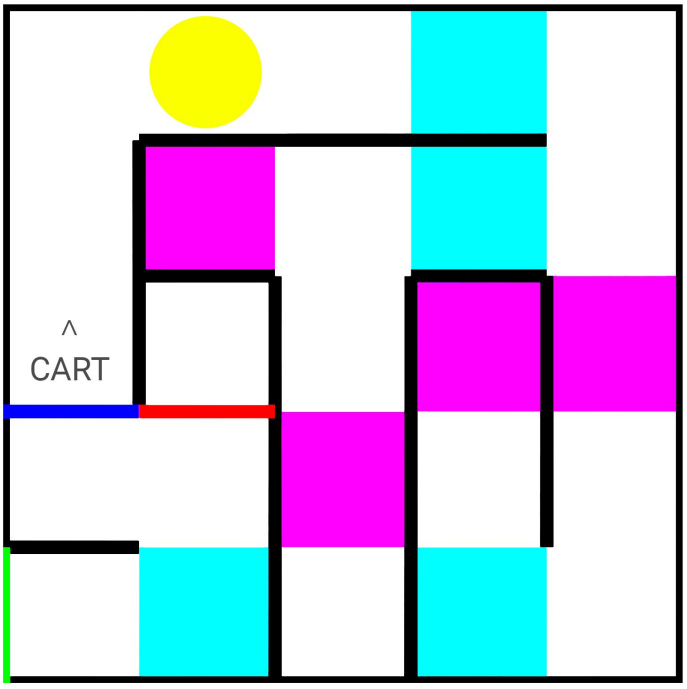
Once the player moves into the green, magenta or turquoise area an event is triggered. The player is supposed to collide with the walls. The player is supposed to collide with the blue area **after** the hidden doors animation has been triggered. The player is supposed to collide with the red area **before** the hidden doors animation has been triggered.
Events are triggered as well with an invisible radius. This radius is not displayed on the map-image and thus not dealt with with the compute shader. Once the camera (princess) is within the radius an action is triggered. This is implemented for the following objects:

- **Hidden door:** It is implemented for the area, where the hidden doors animation is triggered.
- **Cart:** The princess is not supposed to walk into the cart. If she does, the lose-condition is triggered. Since the cart moves, a trigger area, that follows the carts current position, is used.

Illumination

Textures

Gameplay & Instructions



This map shows the placement of all interactable items. It uses the color coding of the Collision Detection section:

- Trapdoor = magenta
- Net = turquoise
- Hidden door before animation = red
- Hidden door after animation = blue
- Goal = green

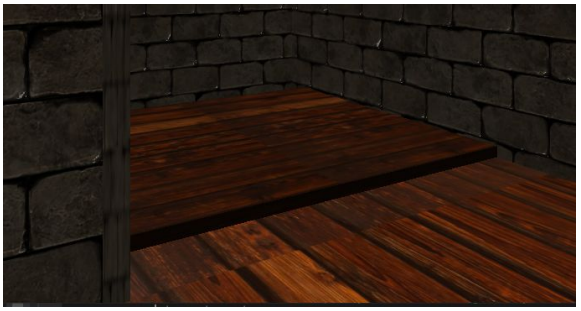
The yellow area is not part of the collision detection image. It shows the area where:

- The animation of the wall is triggered
- The cart starts to move in the displayed direction

Trapdoor

Details

- Animated when triggered
- Done by Eva



Instructions to fall into trap

- Walk into trapdoor
- Player falls down (unable to move)
- Game is lost

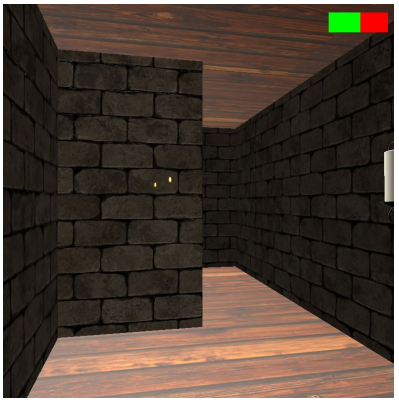
Instructions to avoid trap

- Stand right before the trapdoor
- Press Space-Key to jump
- Player jumps over trapdoor
- Trap is avoided

Hidden Wall

Details

- Animated when triggered
- Texture with Simple Normal Mapping
- Done by Eva



Instructions to be closed in

- Walk into invisible trigger-area
- Wall closes player in

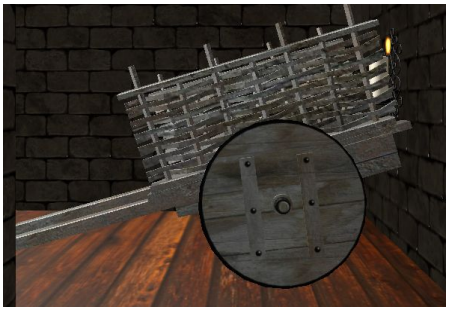
Instructions to be free again

- Player is closed in
- Press S-Key
- Wall opens up again
- Trap is avoided

Cart

Details

- Animated when triggered
- Hierarchical animation
- Done by Klaus & Eva



Instructions to fall into trap

- Walk into invisible trigger-area
- Cart starts to move
- Player comes into touch with the cart
- Game is lost

Instructions to avoid trap

- Walk into invisible trigger-area
- Cart starts to move
- Player is close to the cart

- Press S-Key
- Player jumps over cart
- Trap is avoided

Net

Details

- Animated when triggered
- Done by Eva

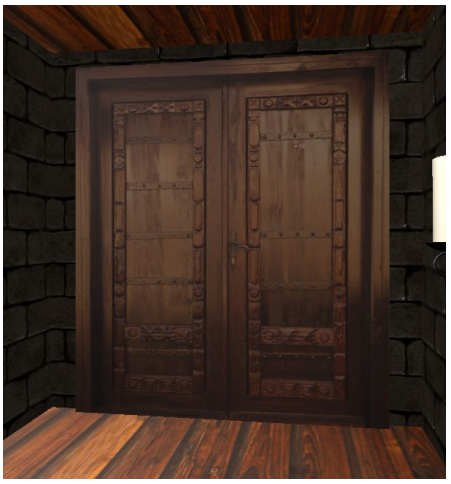
Instructions to fall into trap

- Walk under net (the 1x1 texture on the floor is trigger-area; net is centered on trigger-area)
- Net falls down
- Game is lost

Instructions to avoid trap

- Stand in front of net
- Press S-Key simultaneously to “Forward” Arrow-Key
- Player starts to run
- Stop once net is behind player
- Trap is avoided

Goal



Details

- Done by Eva

Instructions to win game

- Walk into door
- Game is won

Additional Libraries

GLFW & GLEW

For obvious reasons like managing the window, using vectors and matrices, ...

FreeImage

Loading Textures in Texture.cpp

Assimp

Loading Objects in ModelLoader.cpp - Therefore extracting positions, indices, normals, uvs and also textures from the file and creating Mesh-Objects that later can be accessed through the ModelLoader.

Tools

Blender

We used blender to create and export a level containing lots of meshes and simple textures (that are being reused).
We also used blender for collision detection. Therefore we exported a map showing the ground and walls colored differently so a position can be matched in 2D.
We then used blender to bake a single lightmap for all meshes. Therefore we had to export a second level object because the *.obj file format does not support multiple layers.