Documentation	
Group Name/ Game Name	Clumzilla
GitHub Link	https://github.com/Anmoriso/cgue21-clumzilla
Student	Anna Zweckstetter, 01526822
Goal	Reach the other side of the city without destroying too much of it!
Implementation Game Play	
Win/Lose Condition	The character happened to become a huge hamster and wants to become smaller again without destroying too much of the city. To do this, he must reach the drink me bottle on the other side of the city. Once he has done that, the game is won. But if he is destroying at least 5 buildings on his way, he loses. (The finish is currently in the back left corner)
Gameplay	The character Clumsy moves through the city and desperately wants to return to a normal hamster size. But he crashes into the buildings (or other things like fountains) and unfortunately destroys them.
Cameras	There are two different cameras in the game: One third person camera and the debug camera. The view direction of both cameras can be manipulated with the mouse movement. The normal game camera moves together with the character. The other moves independently from the character in the viewing direction with WASD and Q, E for moving up and down the y-axe. You can switch between them with R.
Controls	• WASD: Moving Clumsy, the hamster, direction of the movement depends on the viewing direction of the camera • R: Toggle debug camera • new Movement with debug camera: WASD for front and sideway movement, Q for moving up und E for moving down • Mouse: Camera view direction • mouse wheel: Zoom the camera in and out a little bit • ESC: Quit • +/- (german keyboard): Increase/Decrease illumination multiplier • F1: Toggle wire-frame mode • F2: Toggle back-face culling • N: Toggle usage of normal map • O: for debugging, see the shadow map on the screen
3D Objects	The buildings in the scene are implemented as simple boxes and the only objects in the game that are not imported out of an obj-File. The other more complex objects (fountain, character, ground) were modelled in Blender and imported via a simple object file loader, which can only handle triangle data. These objects also have a representative in physx with a simplified geometry.

	The bottle that marks the target of the game is a model from the Internet (https://www.turbosquid.com/de/Search/3D-Models/free/bottle) and is also loaded into the scene with the object loader. To mark the borders of the city, only static actors in physX are used with no visual representation. So the character is running against an invisible wall.
Scene lighting	The main light source is a directional light (the sun light). Additionally there is a point light at the bottle (finish) to mark it special.
Moving Objects	Clumsy is moving around in the city and if he crashes into a building, it will disappear in the ground.
Adjustable Parameters	The following Parameters can be adjusted in the config file in assets/settings.ini ● Screen Resolution (Width/Height) ● Fullscreen-Mode (true/false) ● Refresh-Rate The brightness of the scene can be manipulated with a illumination multiplier in the game (using the keys +/- on a german keyboard (or } and / on a english keyboard)
Collision Detection	Clumsy is implemented in physX with a Character Controller. If he collides with any dynamic actor (building, fountain), the physX shape will be removed from the simulation. Now the building is no longer standing on the ground (static actor) and will fall through it. After that, the physX position of the object is synchronized with the graphical position.

Effekts	
Feature	*Description (Usage)*
Lighting: Shadow Map with PCF	To calculate the shadows in the game shadow mapping with PCF is used for the sunlight (directional light, main light source). An orthographic perspective for rendering the scene to get the depth values is used. Antialiasing of the shadow calculation: ● To avoid Shadow acne, a bias is added to the comparison of the pixel depth and the depth stored in the shadow texture. (fixed bias of 0.02 is used) ● to avoid peter panning, the back faces of the solid objects (not the ground) are used while rendering the scene from the lights point of view.

	• to get all the shadows of the scene the parameters are set to the size the scenery has (no over sampling) (https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping)
Shading: Simple normal mapping	All the buildings and the ground of the city each have a normal texture that is used in the fragment shader. With the N key, the normal Map can be switched on / off. (https://learnopengl.com/Advanced-Lighting/Normal-Mapping)
Post Processing: Contours via Backfaces	all objects that the character can destroy have a brownish contour.
Texturing: Specular Map	The buildings additionally have a specular Map, especially to have a difference between the wall parts and the windows on the diffuse texture. the effect can best be seen with the debug camera. (https://learnopengl.com/Lighting/Lighting-maps)