Down Under – Documentation

11776835 Florian Obermayr
11777706 Mario Stoff

Elevator pitch

The aim of the game is to navigate the kangaroo across multiple floors down the tower to reach the bottom. You win by reaching the marked area on the final floor. You lose if you hit one of the traps or flames laid out on your way down.

Controls

Use the "A" and "D" Keys to navigate left and right. Use the "W" and "S" Keys to move the kangaroo closer or farther away from the screen. End the game using the "ESC" key or try again instantly with the "R" key. When you finished a level, advance to the next one using the "ENTER" key.

Camera movement

The camera is always centered in on the kangaroo, following it around the tower, wherever it goes.

How it works

First things first we have to create a window. This is done by creating an object of class Engine and calling the initialize method. This method creates a window and initializes GLEW. The necessary parameter like window width and height are read from the settings.ini file. Reading this file is done with the help of the ini and **INIReader** class. The code for this reader is taken from an online tutorial[1] and slightly modified.

After establishing a working window all the necessary models are loaded from a file path using the **ASSIMP** library. The code that loads the assets is located in the model and mesh classes and is influenced by a tutorial on learnopengl.com[2]. For the player model, which also contains the skinning information, the mesh and model classes were extended to read all the necessary data. We implemented **Vertex Skinning** loosely based on a tutorial[3] we found online.

Entering the main game loop, events are polled and the keyboard input is processed. Uniform matrices for the central tower are calculated and the tower with all its elements is drawn.
Finally the kangaroo is drawn. Before the kangaroo is put on screen, all the bones in the model are traversed to get the correct data that is needed at the time. Using the bones and information about them, we can interpolate the values to create the animation effect. NOTE: as we faced a lot of issues developing, we are still working hard on this effect and are confident that we can finish it properly before the game event presentations.

Collision detection along with gravity is properly implemented using the **PhysX** Library. Therefore, collision with walls and holes in the floor that access the next floor is all managed by the physics engine. Upon reaching a hole in the floor the kangaroo will automatically fall down to the next floor. Additionally, an invisible barrier around the level keeps the kangaroo from falling off the tower into the abyss.

---

[1] https://cplusplus-development.de/c/cplusplus-erweitert/ini-reader

[2] https://learnopengl.com/Model-Loading/Model, https://learnopengl.com/Model-Loading/Mesh

[3] http://ogldev.atspace.co.uk/www/tutorial38/tutorial38.html

When the kangaroo reaches a specifically marked area on the final floor of the current level, you can press the "ENTER" key to advance to the next one. When the kangaroo reaches the finish area in the final level, the game is won and has to be terminated using the "ESC" key or restarted using the "R" key. Upon hitting a hazard or finishing the current level or game, a text is rendered on screen to communicate the game state.

The flame hazards in the game are implemented using a **Particle System**. One such Particle System consists of a large number of little particles. All of them have a certain direction in which to move, a color and a lifespan. The lifespan decreases over time and when it reaches zero the particle is replaced by a new one. The color of the particles changes depending on the remaining lifespan in order to create a fiery effect. Each frame all “alive” particles are drawn to the screen using an instanced draw call.

As a little extra, we added a favicon image to our final build using the software **Resource Hacker**[4].

---

[4] http://www.angusj.com/resourcehacker/