Final Project Report

Daniel Trippa 1837561

September 25, 2022

Contents

1 Introduction

The Project is a game based on three.js. The game features a small cat that fights wizards and skeletons with magic balls in a small dungeon. The objective of the game is clearing all waves and save the mage. Three game modes and various settings are available.

2 Technologies

In this section the external libraries and technologies are listed

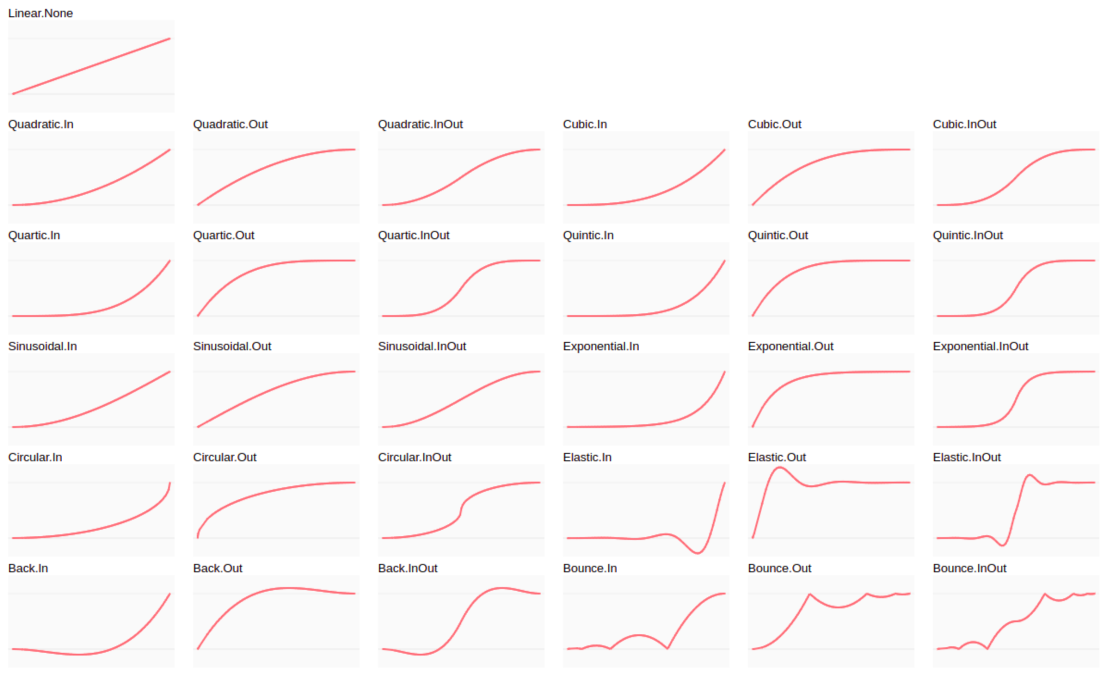
2.1 Three.js

Three.js is a lightweight, cross-browser, general purpose 3D library. Interally it is based on WEBGL. It features different **cameras**, **materials** and **geometries**.

2.2 Tween.js

Tween.js is a library for interpolating between propieties. It is used to create smooth animations. Different easing equations are avilable as shown in fig.1. For the most part, Cubic.inOut was used, as it gave the best results for interpolating between keyframes.

Figure 1: Avilable easing functions in tween.js



2.3 Three.js editor

Three.js editor is an online editor for gltf models. It is written using three.js. It helped with visualizing the model and finding keyframes for animations.

2.4 HTML and CSS

For menu and the interface, standard HTML and CSS was used.

3 Graphical aspects

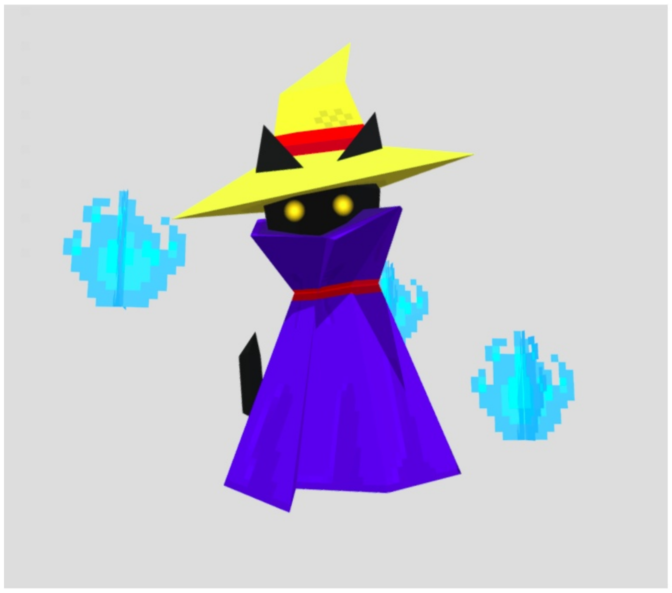
3.1 Models and Textures

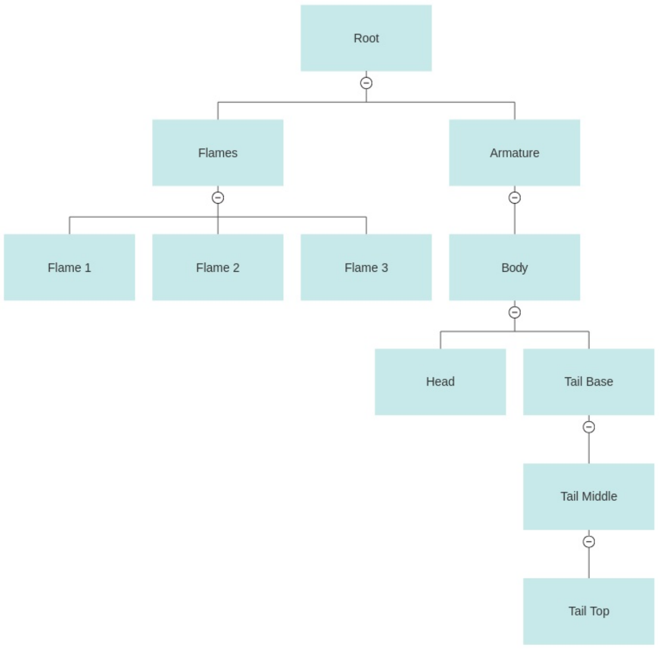
This section lists the external models and texture used in the project. For the most part, model didn't have a hierarchcal structure and thus it was added programmatically during the importing process.

3.1.1 Mage

The mage model. It's the protagonist of the game and it's possible to control it's movement.

Figure 2: Cat Mage model





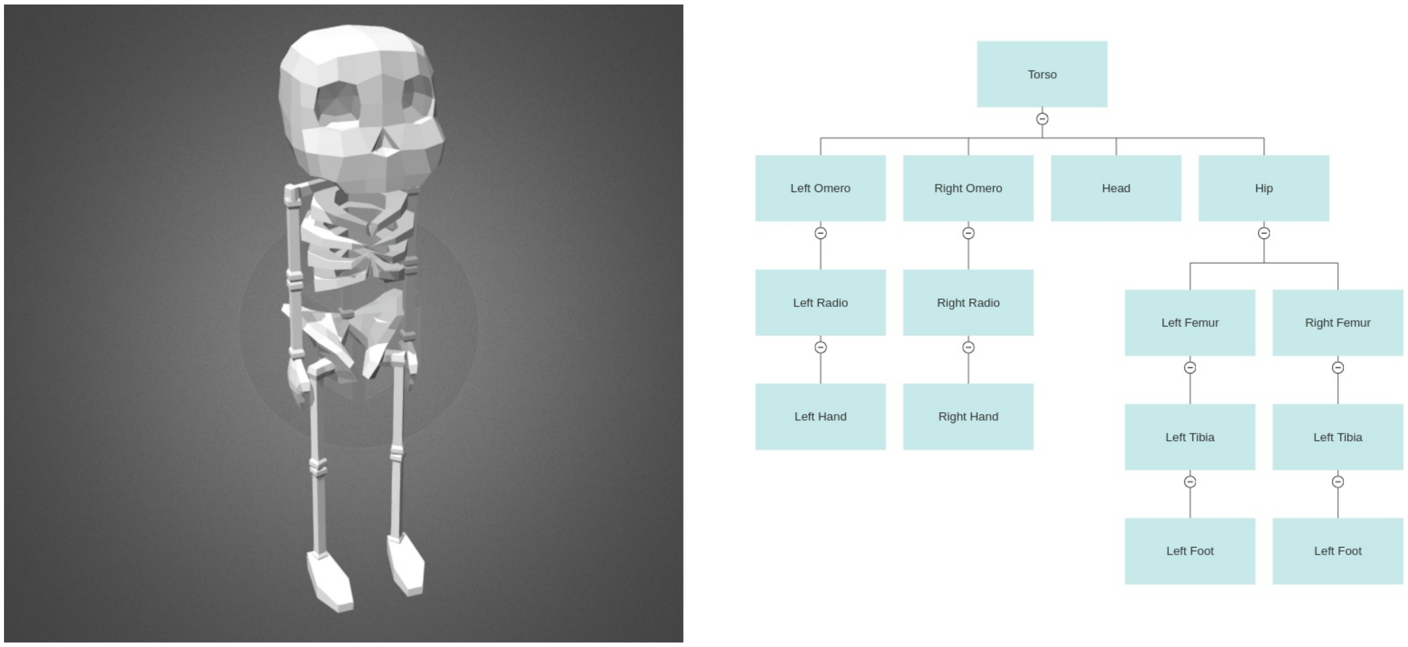
3.1.2 Skeleton

The Skeleton is one of the main enemies of the game. It follows the player and damages it when he comes close. As it doesn't come with a hierarichcal structure it was manually created(See section on Loading).

3.1.3 Wizard

Wizard is the second main enemy of the game. He damages the player by throwing fireballs at him. Again as it doesn't come with a hierarchical structure it was manually added.

Figure 3: Skeleton model and it's programatically created hierarchy



3.1.4 Walls and Props

Additionally, walls and some other props are used to decorate the game arena.

3.1.5 Textures

All models in game have its own texture. Additionally the plane that represents the floor uses a parametric scalable color texture, normal map and displacement map.

3.2 Animations

3.2.1 Camera

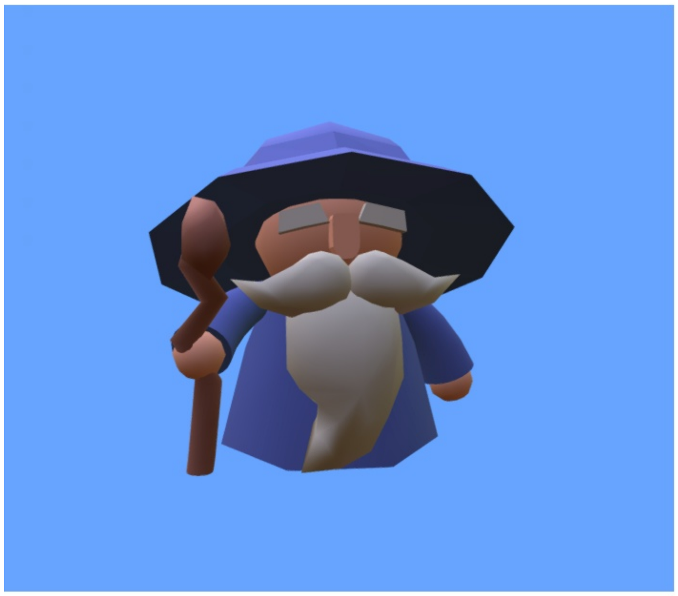
The menu opens with the camera pointed towards the player. Once the game is started the camera position and rotation are animated. The follow a carefully placed bezier curve that traces a path and ends up behind the player at a specific height. During the movement the rotation of the camera is kept towards the player.

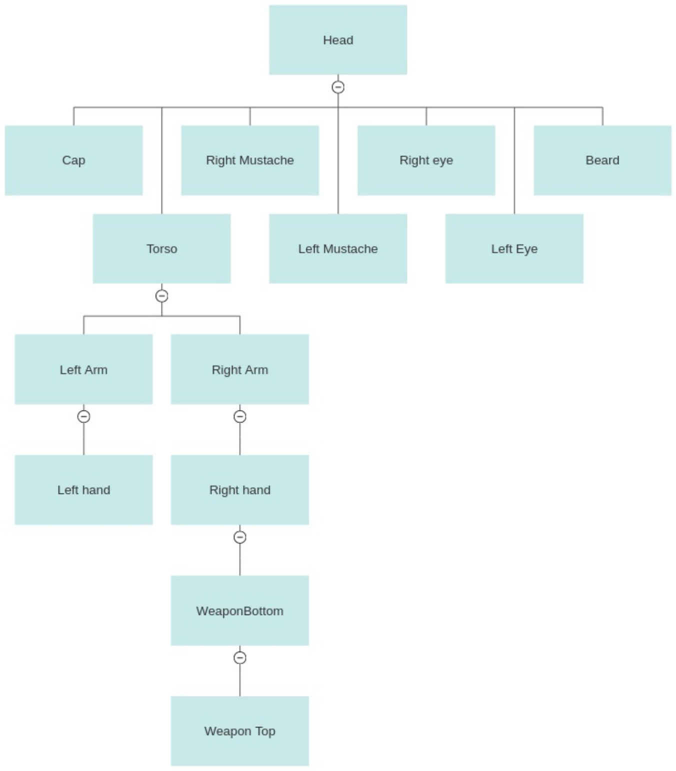
3.2.2 Mage

Different parts of the mage are animated separatly in an idle animation. The three flame are rotating around the body at different speeds. Head and tail are keyframed and chained in an endless tween with different durations. The tail is composed by three parts and each of them moves independently with respect to the other, to give the impression of a more natural movement. The model is also constantly floating and slowly moving up and down following a sine wave. Overall this becomes an idle animation that is observable from the start menu. Once the game starts the flames start moving faster around the player.

When the player is hit, another tweeen is started that changes the material opacity of the player. The final effect is that of an invulnerability state.

Figure 4: Wizard model and it's programatically created hierarchy





When the player dies, the camera is detached from the player and it slowly shrinks until it disappears.

3.2.3 Skeleton

The skeleton has three different animations: moving, attacking and dying. The Moving animation was obtained by moving the tibia with respect to the femours and the radio with respect to the homerus. Left and right Femours and Homerus are moving with respect to the torso in an alternating pattern. At the same time the body and head move up and down to simulate a walk. During the attack animation the body and the head still giggle around while the arm moves up and down to hit the player. Once the skeleton dies, it shrinks and shakes until it disappears.

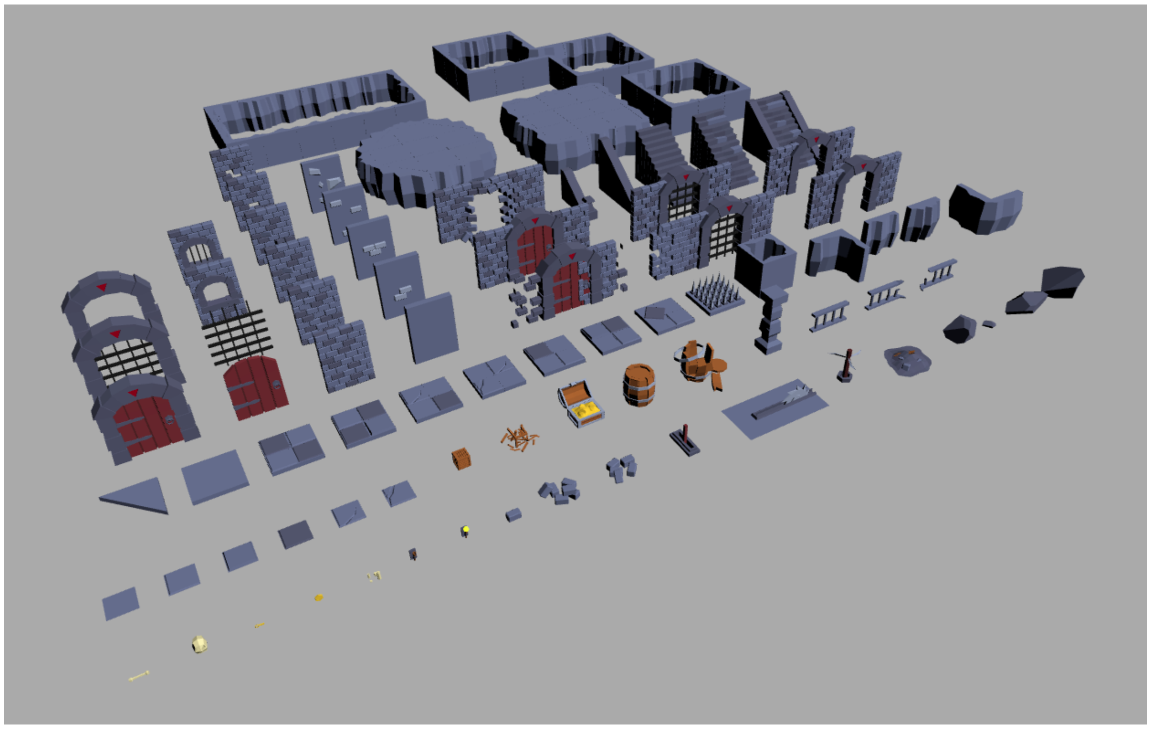
3.2.4 Wizard

The wizard is constantly floating following a sine wave. Additionally for the attack animation three keyframes were chained togheter. The keyframes involved moving the head with respect to the torso, the arms with respect to the torso, the hand with respect to the arm and the weapon with respect to the hand. On the end of the last keyframe, the attack function is called and a fireball is fired. When he dies, he stops attacking and shrinks while shaking until he disappears.

3.2.5 Particles

Particles are simple little sprites with texture. Sprites always appear flat and face the camera. Particles are emitted both from the players and the enemies fireballs. Each single particle moves independently. While the fireball is moving, the particles spawn randomly around the origin of

Figure 5: Dungeon waals and props



the fireball and travel upwards before disappearing. When the fireball hits something, particles originate from the center and choose a random direction they will travel. This gives the effect of a spherical explosion

3.3 Lighting

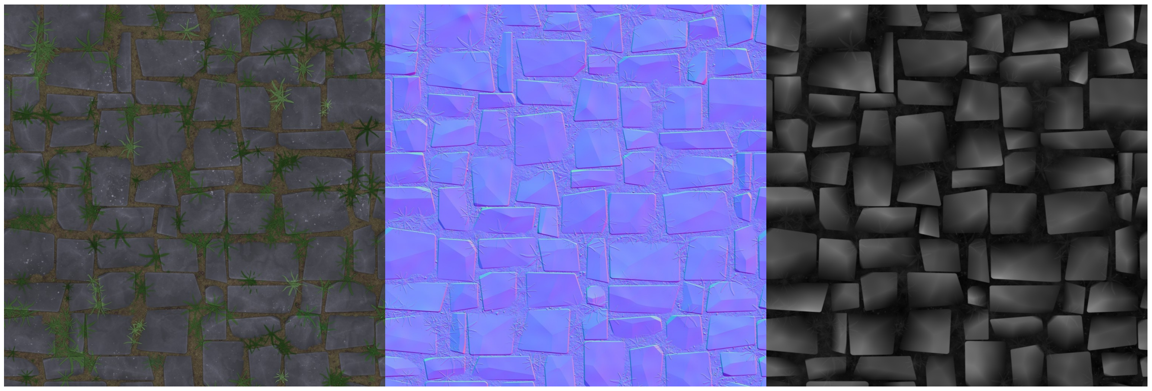
3.3.1 Lights

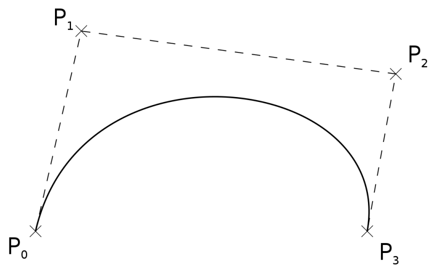
The scene features three directional lights, facing different directions to light the whole scene. The light are all white but have different intensities.

3.3.2 Shadows

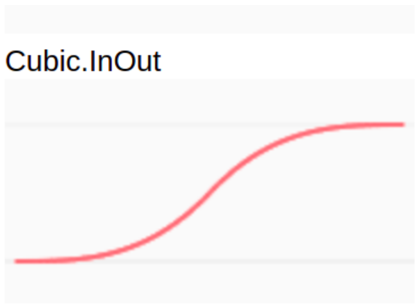
The third light is able to cast shadows and the floor is able to receive them. This light is the most intense and is almost vertical to the scene. Not all objects are able to cast light, in fact none of the visible one. To make a shadow appear below the players and the enemies, an invisible sphere that can cast shadow is added to their mesh and adjusted to give the right radius.

Figure 6: Floor textures. From left to right: color, normal map, displacment map.





(a) Bezier curve



(b) Interpolation function used between keyframes

Figure 7: Curves used for animations

4 Gameplay Aspects

4.1 Game Mechanics

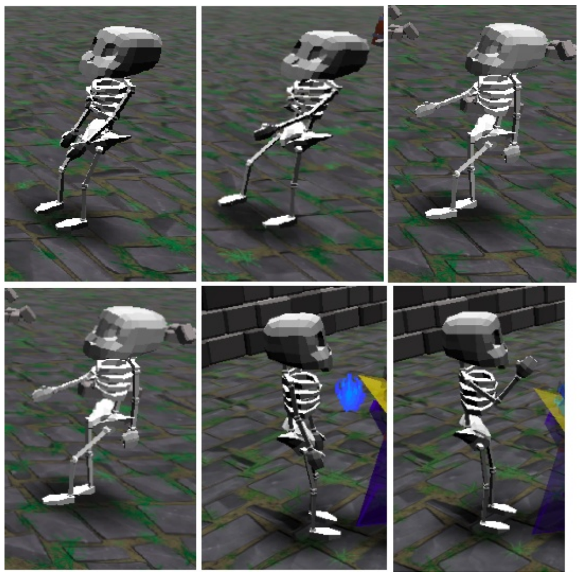
4.1.1 Controls

A	Move left
D	Move right
S	Move Backwards
W	Move Forward
←	Turn Camera Left
→	Turn Camera Right
↓	Move Camera Down
↑	Move Camera Up
SpaceBar	Shoot

4.1.2 Game Menu

The Game menu allows to select game mode and change some graphic options. Select the game mode to start the game. A tutorial is shown on screen when starting and disappears while playing.

Figure 8: Skeleton moving and attacking



4.1.3 Upgrades

At the end of each wave in game, it's possible to choose between some powerups before starting a new wave: health, damage and range. The selection changes the corresponding game parameter to upgrade the players capabilities.

4.2 Movements

Every moving entity, including the player, has a different direction it is facing at any time. In the case of the player, for example, this direction is given by its rotation around the y axis. To correctly move the player, it's position must be updated according to the direction it is facing. This involves some trigonometry as follows:
Forward Movement:

$$pos.z = pos.z + cos(rot.y)$$

$$pos.x = pos.x + sin(rot.y)$$

Right Movement:

$$pos.z = pos.z + sin(rot.y)$$

$$pos.x = pos.x - cos(rot.y)$$

In contrast, the direction that the enemies are facing is directly determined by **player.pos-enemy.pos** and their rotation is set accordingly.

$$dir = player.pos - enemy.pos$$

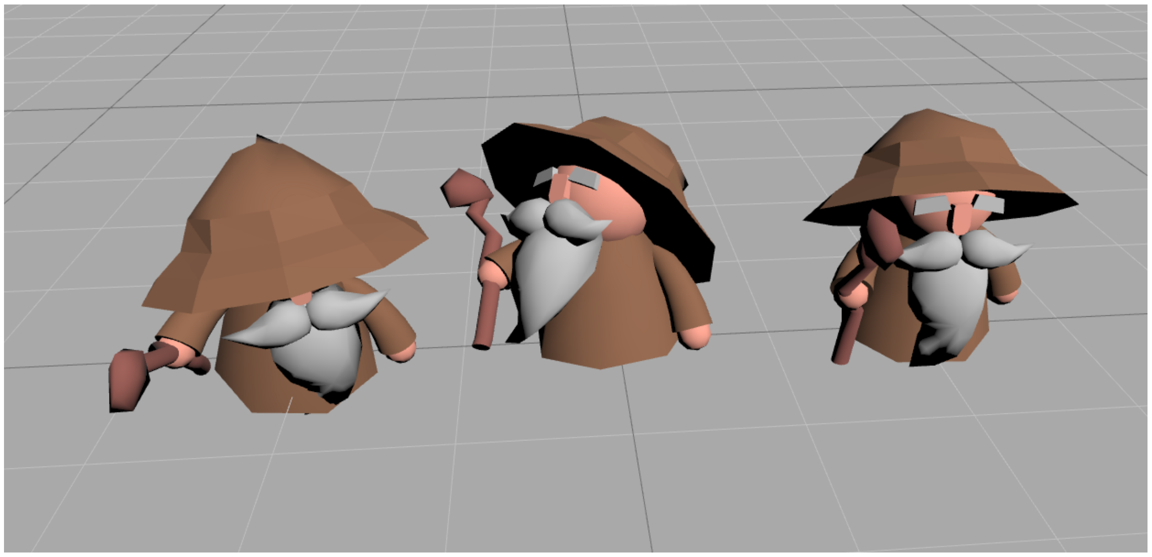
$$enemy.rot.y = atan2(dir.x, dir.z)$$

4.3 Game Entities

4.3.1 Loading

Almost each entity in game is modeled by a javscript class. This classes have a static function executed once between all instances. Here the model is loaded and adjusted accordingly by scal-

Figure 9: Three main keyframes for wizard.



ing, rotating and positioning. Skeleton and Warrior model are made hierarchical in this fase by connecting different parts togheter using three.js **attach()** function.

4.3.2 Spawning

Every time a new instance needs to be spawned, The geometry and material of the model loaded during the static function are used to create a new mesh. This is a way to optimize multiple instancing. A list keeps the number of Wizards and skeleton that need to spawn at each wave. At start different props are spawned in random **locations** and **rotations** to make the arena look different every time. Also enemies are spawned in random locations and will always face the player.

4.4 State Machine

The game logic is based on a state machine that controls the current state of the game and handles things accordingly. There is a phase for the menu, starting, playing, upgrading, loosing and winning. During each phase update functions are called to handle all object in the scene that need to move. This includes both enemies, the player and fireballs.

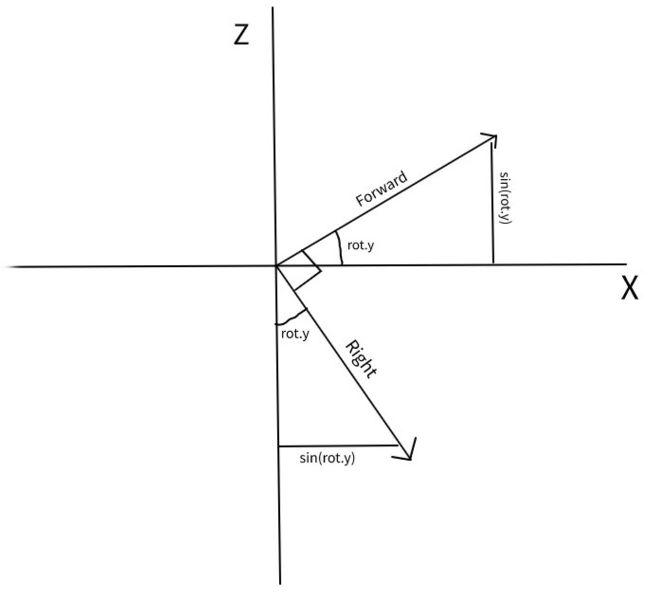
4.5 Walls and Raycasting

Walls act as a barrier to limit player's movement but at the same time could also limit it's view. Here follows the method used to solve this issue. Raycasting is the method three.js uses for **Picking**. In this case a ray is casted towards the player from the center of the camera. The ray only intercepts walls. If a wall is found between the camera and the player, wall's transparency is affected to make the player always visible on screen.

4.6 Collisions

During the main loop, collisions checking functions are called. The player can collide with the walls and the barrels. Fireballs can hit the player or the enemies. Collisions are checked based on the distance between objects that can collide and are logged in the console. When fireballs collide, their

Figure 10: Trigonometry involved in player and enemies movement



status is changed to **exploding** and the players or enemies health are updated accordingly. If a fireball collides with the player the **invulnerability** status is triggered and the player can't lose any more health in the next few seconds.

5 Interactions

5.1 Menu

5.1.1 Easy Mode

The menu allows to choose a difficulty. Different parameters affect the difficulty, such as wave numbers, enemies , health and fireball damage. Easy mode starts with just five waves left and fireballs that kill enemies with two shots.

5.1.2 Hard Mode

Hard mode changes the list that handles the waves , adding more enemies. Also fireball damage is reduced and upgrades are less effective.

5.1.3 Test Mode

Like easy mode but the player is invincible. He never loses health even when hit by enemies.

5.1.4 Disable Particles

A button to trigger fireball's particle effects. This can be disabled also for performance reasons.

5.1.5 Light Slider

A slider that allows to regulate the intensity of the main light, the same that also casts shadows.

5.1.6 Fog Slider

A slider that allows to regulate the fog that is added to the scene. The more fog in the arena, the more difficult the game is.

5.2 In Game

5.2.1 Movement

Player can be moved laterally, backwards and forward. Movement is blocked by barrels and walls.

5.2.2 Shooting

Player can shoot fireballs in the direction it is facing.

5.2.3 Upgrades

At the end of each wave, upgrades can be chosen:

- **Health**. Selecting this upgrade adds one hearth to the players health, updating the internal game options and the user interface.
- **Damage**. Selecting this upgrade increases the size of the **THREE.SphereGeometry** associated to the Fireball class and also increase the fireball damage parameter. Fireballs will appear bigger and will kill the enemies faster.
- **Range**. Selecting this upgrade increases the TimeToLive parameter of fireballs and updates their speed. This way fireballs will travel longer and faster.

5.2.4 Camera

- **Camera can be rotated laterally**. The effect is obtained by attaching the camera to the player and rotating the player itself. The camera will follow accordingly.
- **Camera can be moved vertically**. The **y** coordinate of the camera is updated with respect to it's local space (player coordinates). The camera y position has a lower and an upper limit.