DOOMICRAFT

Minecraft + Doom = FUN

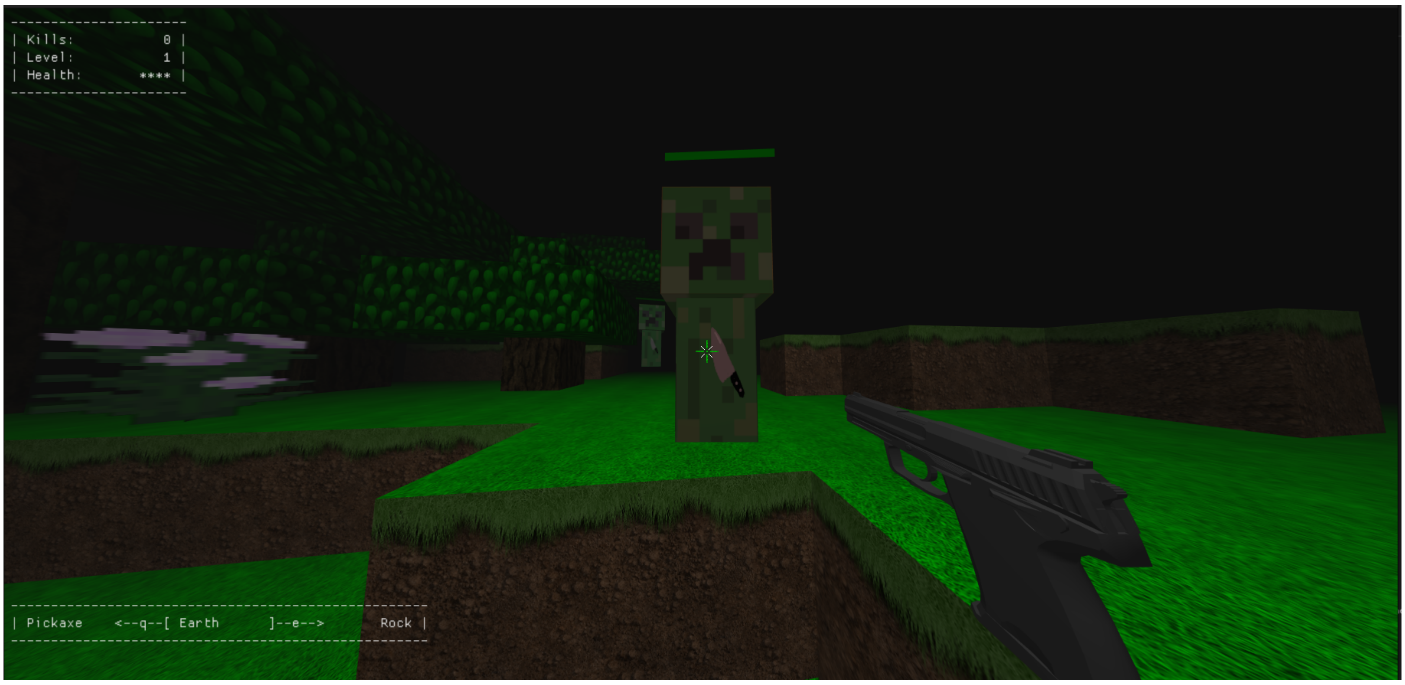
A mix between Minecraft and Doom! Build your world on Creative or fight creepers in Survival, hoping to last another level...

Made on C++ with the CGP library (built on top of OpenGL) as an learning experiment with graphics programing.

The game is free to play and it uses free assets.

Originally a project for the INF443 course at Polytechnique.

Authors: Guile Vieira (guilevieiram), João Tanaka (jptanaka) and Rafael Nagai



📥 Downloading

To download the latest version of the game refer to the releases tab of the GitHub page. After downloading the package (.tar for linux and .zip for windows), go into the folder Doomicraft, then into build and run (double click) on Doomicraft! The path is going to look like: `Doomicraft/build/Doomicraft[.exe]` ;

The game should start running on your machine after that!

You are encouraged to create a shortcut on your Desktop workspace if you want to access without going into the folder structure.

Compiling

If you want to compile the game by yourself on your linux machine, go through with the following steps:

1. Make sure you have CMake on version at least 3.8 and git installed on your machine.

2. clone the project:

```
git clone https://github.com/guilevieiram/Doomicraft
cd Doomicraft
```

2. Initialize the submodules

```
git submodule init
git submodule update
```

3. Create the build

```
cmake -B build
cd build
```

4. Compile everything

```
make
```

5. Run the game!

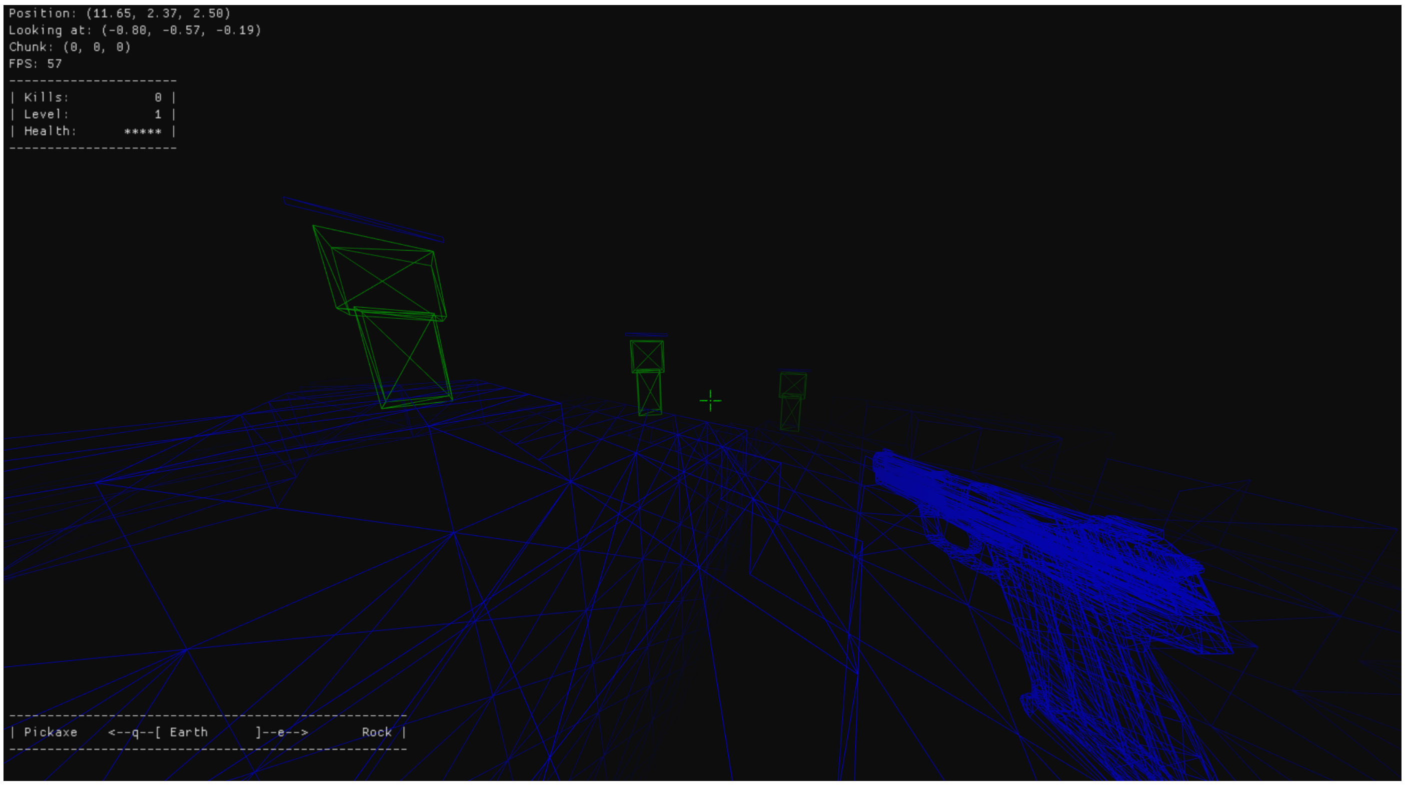
```
./Doomicraft
```

Galery

Videos of the gameplay showing the game features:

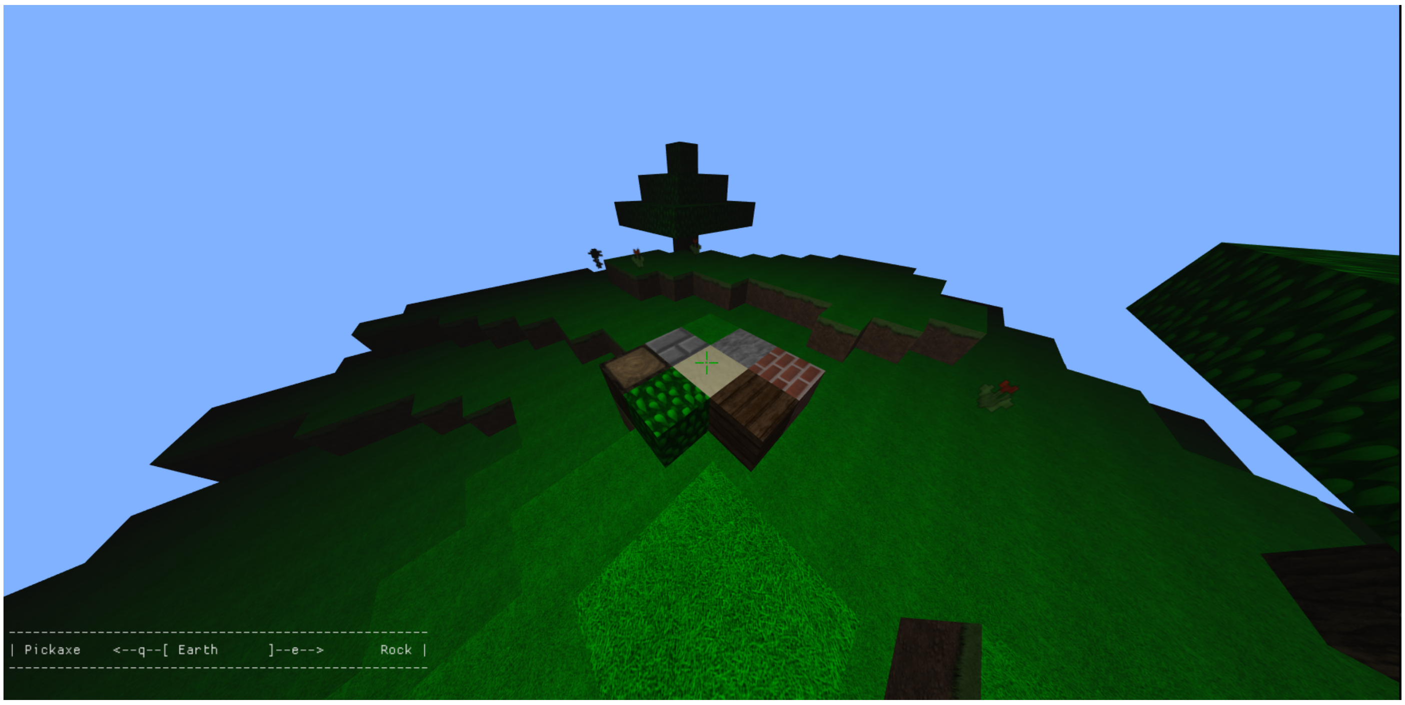
- Creative: https://youtu.be/Fd9LEMZV8Cs
- Survival: https://youtu.be/ag2cmsAGo3w

Some images from the gameplay.

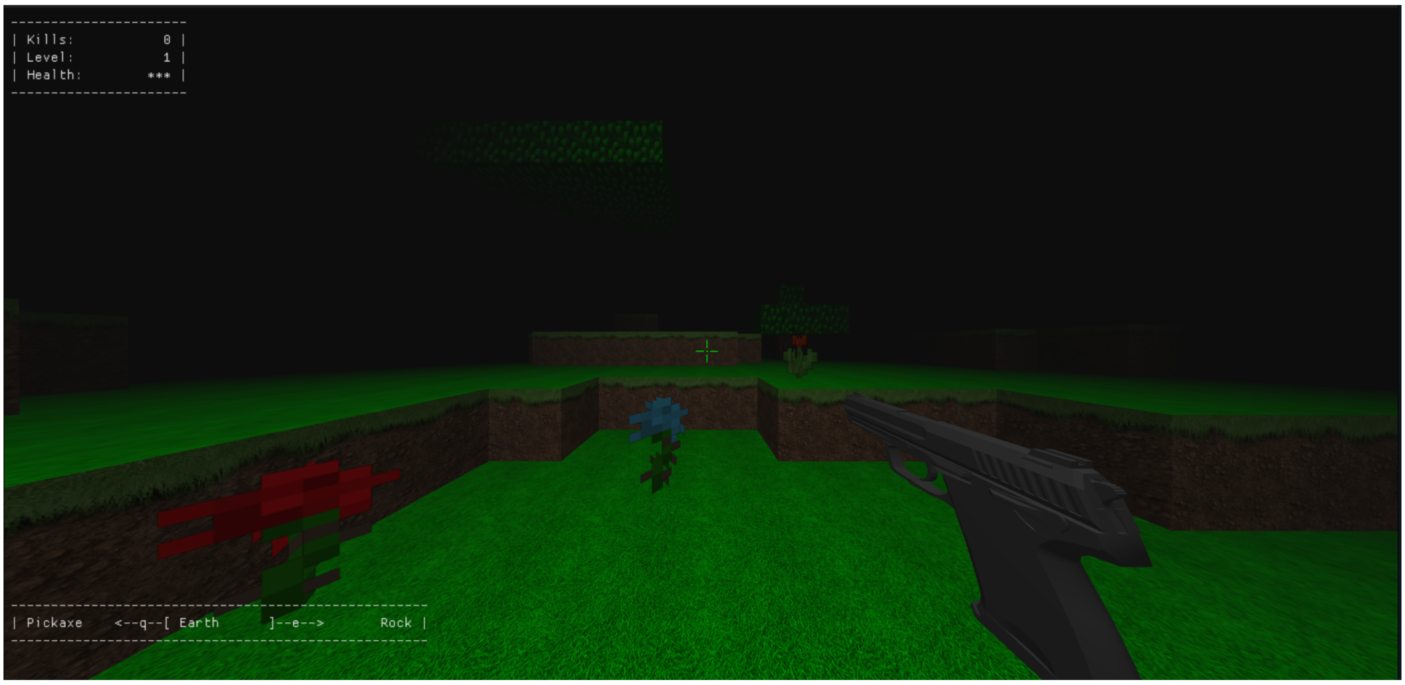


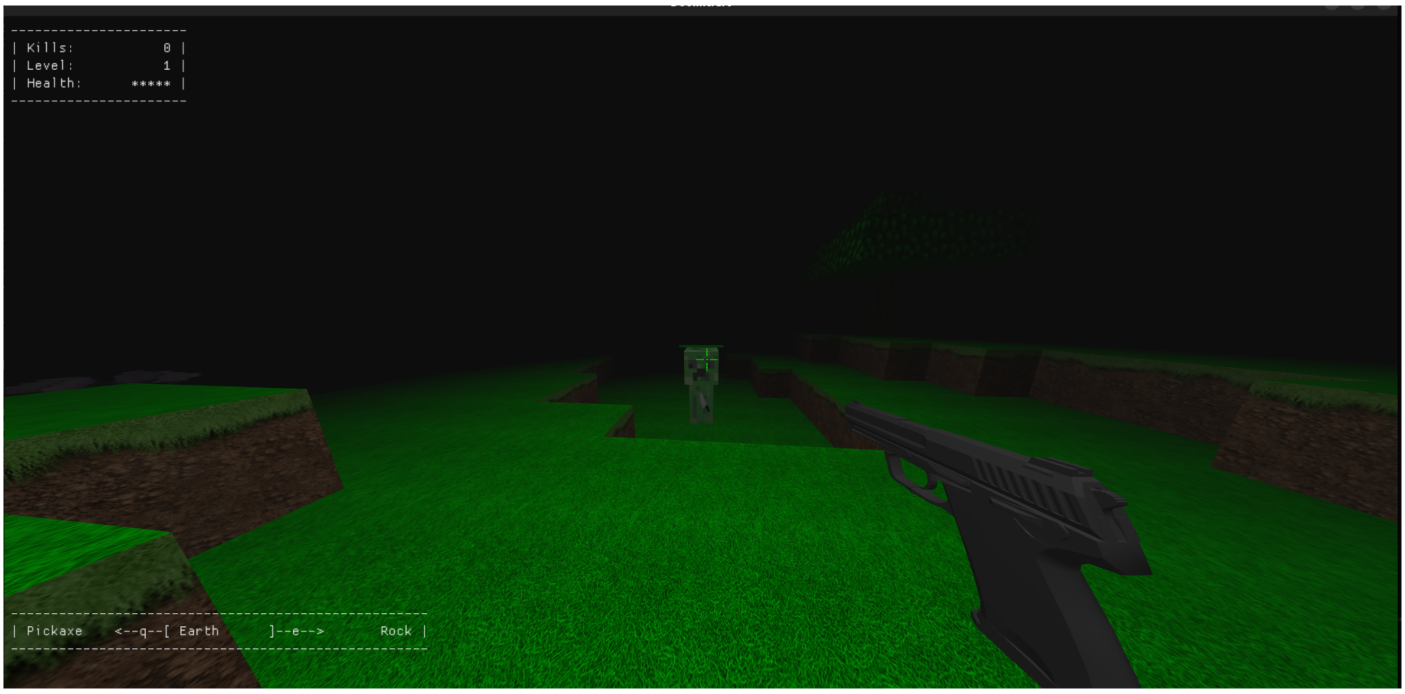
```
Position: (11.65, 2.37, 2.50)
Looking at: (-0.80, -0.57, -0.19)
Chunk: (0, 0, 0)
FPS: 57
----------------------
| Kills:            0 |
| Level:            1 |
| Health:       ***** |
----------------------
```

```
---------------------------------------------------
| Pickaxe    <--q--[ Earth       ]--e-->       Rock |
---------------------------------------------------
```



```
---------------------------------------------------
| Pickaxe    <--q--[ Earth       ]--e-->       Rock |
---------------------------------------------------
```





Implementation details

This game was completely implemented in C++ using the CGP library (linked as a submodule) for graphics rendering. OOP was the main paradigm chosen for the project even though some new C++ features (like lambda functions) were used from time to time.

Many algorithms and implementations were implemented in the construction of the program. We will in this section highlight the main ones.

Rendering

To achieve high performance on the game (at least 30FPS), some modifications on the block rendering were made. Initially, each block is a collection of 6 square faces, each one with a texture.

1. The meshes for each type of block are created and loaded on the GPU only once in the initialization of the program. The drawing of different blocks is made using translations. The same works for the flowers, they are loaded in the GPU only once.
2. The contact face when two blocks are touching are not rendered, which decreases significantly the render time for the scene.
3. A max depth to render blocks is defined to prevent blocks far away to be rendered. This, together with the Fog (implemented on the Shader) gives a smooth transition between the terrain and the background. Also the number of render calls decreses significantly with this optimization. The fog depth can be changed by the player on the menu on runtime!
4. Block faces that are facing away from the player are not rendered as well, saving a lot of runtime.

Collision

Two collision algorithms were implemented, one for the main player and one for the mobs/creepers. Both of them are based on AABB (Axis Aligned Bounding Boxes) collision that fits well the game theme.

1. **The Mob Collision: AABB vs AABB**. In this algorithm we check, at each point in time if the bounding box from the creeper (two blocks on top of eachother) is going to collide with the blocks around it. If there is going to be a collision, we filter the move direction to prevent it. By looking only at a achievable subset of blocks, this approach is pretty fast and adaptable to implement the mob AI. Nonetheless, it leads to undeterministic behaviour and tunneling, this is why we implemented another algorithm for the main player.

2. **The Player Collision: Swept AABB vs AABB**. For this algorithm the main idea is to simulate the player movement in time and stop at the earliest time of collision with another block. Although it is more expensive, it prevents tunnelling and allows for movement response. For a better playability, we implemented a sliding response to collision with the terrain and we modeled the player using a smaller parallepiped (to allow 1 by 1 tunnels). More information on this algorithm can be found on the page https://www.gamedev.net/tutorials/programming/general-and-gameplay-programming/swept-aabb-collision-detection-and-response-r3084/

Terrain Generation

To gerenate infinite terrain for the game, a few approaches were taken:

1. We used Perlin Noise to determine the height of the terrain.

2. Trees and flowers were placed by simulating a uniform random variable at each block. A check to see if another block was already in place was put in place.
3. A Chunk model was implemented to increase performance and allow for infinite worlds. A chunk is a 16 by 16 (on x and y) of the terrain. The terrain rendering and collisions with blocks are checked only in the surrounding chunks of the player, wich allows for a constant time check independently of the size of the world. New chunks are generated at need when the player moves. Breaking and placing blocks alter the chunk permanently so that changes persist for the rest of the game.

Modeling and Texturing

Most of the modeling was done using CGP primitive shapes (squares, cubes, ...) and some were made using blender.

The gun was downloaded as a free-to-use asset and modified to our needs using blender. The blocks and flowers textures were downloaded from free-to-use Minecraft texture packs and adapted to our system. The creeper art was made by hand on Gimp and used as a texture for the mobs.

SFX

To use sound effects in the game a header only library (miniaudio) was used. This API allows the C++ program to play sounds on the main audio driver of the computer, but this execution blocks the execution of the main thread. To surpass this and play different sounds at different times, we implemented a threaded system that spawns one thread for each SFX that stays alive until the end of the game. This thread accesses a set of variables called listeners that dictate if the given sound should be played or not. This allows for other objects (like the player or terrain) to activate sounds. The assets were taken from free-to-use libraries and adapted to our needs (volume, pitch, duration, ...) using the Audacity software.

Gameplay

Two main gameplay modes were implemented: Survival and Creative. On the survival mode the objective is to kill as many mobs as possible as the level increases. On the creative you have more time and resources to explore and build!

1. A mob group controller that keeps track of deaths, lifes and spawns. At every ~20 seconds (or when a mob dies) another creeper spawns.
2. Each mob is given an AI that follows the player. In encountering terrain, it jumps and continues moving towards the player.

3. The player was given 5 lifes (looses one at each touch of a creeper), a respawn timed system, the hability to place and destroy different blocks, ...
4. A UI with the player status (life, kills and level) and another with the blocks are given to allow for smoother gameplay. This was implemented using the ImGUI library.
5. A main menu was implemented to allow for game pausing, exiting and other configurations. These configurations can be accessed in the debug mode and include a creative mode, change fog depth, see wireframes and other functionalities.
6. A crosshair with a hitmark sensor was given to the player to allow aiming. To do this, the creation of timed GUI elements were needed. This system was then used to display other messages on the screen for a fixed amount of time.