{EPITECH}

WOLFENSTEIN3D

IT'S YOUR TURN TO CREATE YOUR GAME!

WOLFENSTEIN3D

binary name: wolf3d
language: C / CSFML
compilation: via Makefile, including re, clean, fclean rules

The totality of your source files, except all useless files (binary, temp files, objfiles,. . .), must be included in your delivery. Error messages have to be written on the error output, and the program should then exit with the 84 error code (0 if there is no error). The size of your repository (including the assets) must be as small as possible. Think of the format and the encoding of your resource files (sounds, music, images,..). An average maximal size might be 30MB, all included.

Introduction

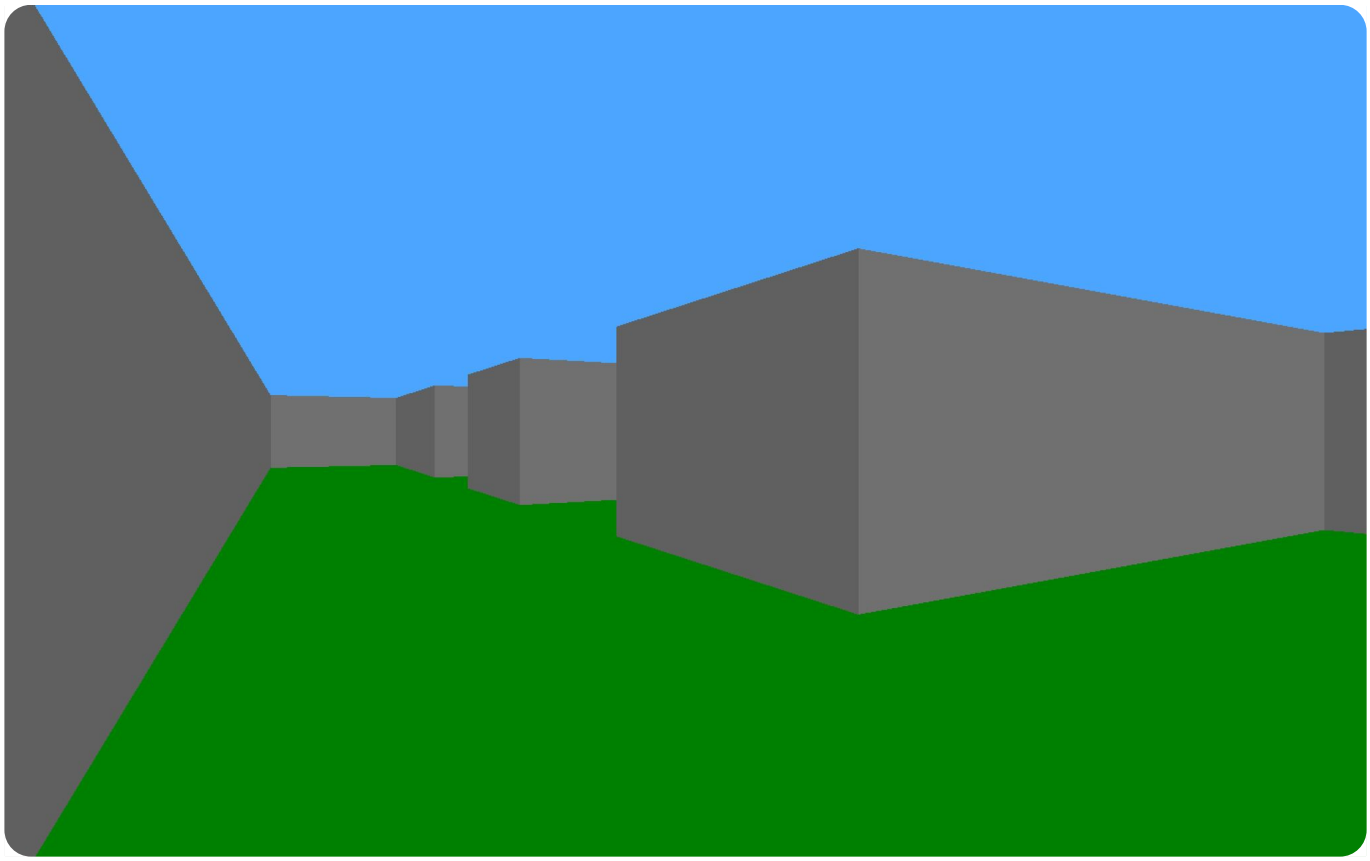
This project consists of creating a clone of the classic game Wolfenstein 3D. The goal is to develop a 3D first-person shooter using a simple rendering technique, raycasting This project will put skills in programming, software development and game design into practice. Your main challenge for this project will be to create a complete product using everything that you and your team know.

By the end of your project, you are supposed to have:

- ✓ A pleasant and user-friendly interface.
- ✓ A coherent universe, including consistent visual assets, audio, and scenario.
- ✓ A fun game where the player has at least one clear goal.
- ✓ A game with a beginning and an end, offering a complete gameplay experience.

Context

This project involves creating a first-person view inside a maze. You will have to program this perspective view as well as the movement of the camera on the map. The drawing principle that you must employ is raycasting.
To do this, you will need to be able to load map files containing one or more labyrinths, a starting position, and the locations of various game elements.
You are free to use any map format. However, we encourage you to design a unique format that allows for easy map exchange.
The player must be able to move in real time on the map using the keyboard.
The color of the walls should vary depending on their orientation or have a texture.



Technical Requirements

To successfully validate your project, you must ensure that the following four aspects are properly implemented and considered.

- ✓ Raycasting: Use the raycasting technique to render walls and environments in 3D.
- ✓ Sprites: Use sprites for enemies and objects.
- ✓ Animations: The animations must be independent of the frame rate and timed by clocks.
- ✓ Particle Systems: Implement a simple particle system for visual effects (such as shots).

Features

MUST

General

- ✓ The window may be closed using events.
- ✓ The windows may have differents mode:
  - `Window mode`
  - `Full-screen mode`
- ✓ The game manages inputs from the mouse click and keyboard.
- ✓ The game contains animated sprites rendered thanks to sprite sheets.
- ✓ Animations in your program are frame rate independent.
- ✓ Animations and movements in your program are timed by clocks.

Gameplay and mechanics

- ✓ Movement: Basic movements (forward, backward, left and right).
- ✓ First person : First-person perspective.
- ✓ Textures and sprites : Use of textures and sprites for objects.
- ✓ Light (flashlight).
- ✓ A basic weapon (knife or pistol): possibility to hit the walls without enemies, enemies are NOT a MUST feature.
- ✓ Save system.

Environment

- ✓ Environment display (floor, ceiling and walls).
- ✓ Collision system.

Sounds and Graphics

- ✓ At least one animation: Movement, shots, enemies etc.
- ✓ Sound effects and music: Sounds and music to enhance the gaming experience.
- ✓ Particle effects: Use particles for explosions, shots, etc

User interface

- ✓ HUD: Interface displaying essential information such as health, ammunition, etc.
- ✓ Options and Settings
- ✓ The main menu must contain:

```
- Start the game
- Settings: volumes and windows
```

- ✓ Leave the game
- ✓ Volume options: for sound and music.
- ✓ Window resolution options: for sound and music.

Should

- ✓ Weapons: Several weapons ranging from knife to missile launcher, through machine guns.
- ✓ Map or mini map: Display of the level map to help navigation.
- ✓ Procedural Labyrinthine Levels: Procedural generation of levels with walls, doors, floor, ceiling.
- ✓ Various enemies: Different types of enemies with distinct behaviors.
- ✓ Inventory: Interface to manage the collected items.
- ✓ Localized Damage System: Different damage based on the affected areas on enemies.
- ✓ Health system: Player's health indicator.
- ✓ Ammunition: Ammunition management for each weapon.
- ✓ Doors and secrets: Locked doors requiring keys, hidden walls with secrets.
- ✓ Collectibles: Treasures, keys and other collectibles.
- ✓ Environmental Interactions: For example, firing on explosive barrels or triggering traps

Could

- ✓ Ladder management for different floors: Mechanical to move between different levels.
- ✓ Advanced health system: Understand health effects such as bleeding or bandages.
- ✓ Special weapons: Unique weapons with special effects.
- ✓ Dynamic Weather: Integrate changing weather conditions affecting visibility and gameplay.
- ✓ Difficulty levels: Different difficulty levels adjusting the game settings.
- ✓ Score and time: Scoring and timing system to measure player performance.
- ✓ Destructible environment: Elements of the decor that can be destroyed or modified.
- ✓ More complex shapes: display and manage shapes other than squares.
- ✓ Have a map editor.

v2