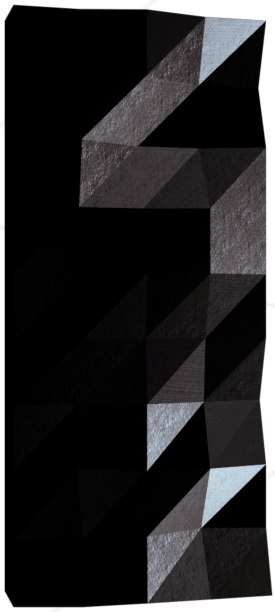


Bomberman

Summary: **Bomberman** *is one of the most famous video games ever. With more that 70 adaptations, from the first version on* **MSX**, **ZX Spectrum** *and* **Sharp MZ-700** *in 1983 to the last versions available on the* **PlayStation Network**, **WiiWare** *and the* **Xbox Live Arcade** *commercialized in 2010. More than 10 millions units have been sold.*

Version: 3.1

Contents

Chapter I

Introduction

Everybody loves playing `Bomberman`. Good news. The goal of this projet is to create a full implementation of a 3D `Bomberman` video game.

- An overview of existing `Bomberman` games for inspiration: `http://www.uvlist.net/groups/info/bomberman`.
- An online version of Bomberman for... You know... science !

Chapter II

Objectives

There are several distinct objectives in this project.

- Beeing able to produce a full video game, not just a pedagogical projet. This means, that the final produt should be usable by tiers, not just the students group who wrote it. One should not think "this is some early access crap" when playing your game.
- Learning the low level use of low level GPU SDK like `OpenGL`,`Vulkan` or `Metal` without the help of modern game engines, and why not start your own (tiny) little one ?
- Learning team work on a full scale project. Everybody can't work on everything, so it's an excellent opportunity to discover that communication, management and documentation are keys to success.

Chapter III

Generic instructions

- Your free to use any langage.
- You must use the last available version of `OpenGL`, `Vulkan` or `Metal` for anything related to graphics.
- You are free to use any additionnal library you like, but **NOT** full game engines like `Ogre` for instance.
- You **MUST NOT** push any library that you did not write yourself on your repository ! Doing so equals zero at the defense. For instance, if you use `OpenGL` (orly?), you `MUST NOT` push `OpenGL` and/or its sources on your repository. Same for `Boost` or anything else. To ensure that you will be able to compile your work during the defense, you must provide a script or run a tool to set up your environment, and download and install your dependencies if needed. You should have a look at `CMake` for instance, or any other tool you like, or do that work yourself. Please read this part again, we won't accept complaints after a failed defense because you were not able to compile your work on a fresh session, whatever the version of the dump.

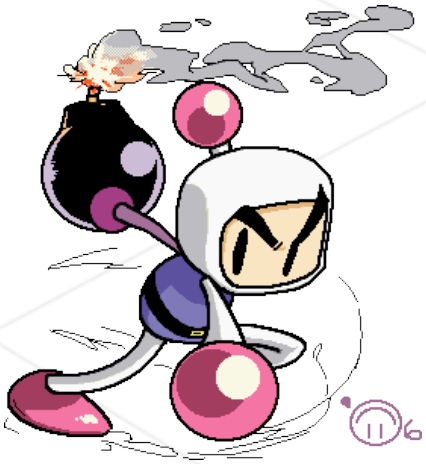
Chapter IV

Mandatory part

IV.1 Generalities

So, here we are. You must code a 3D version of `Bomberman`, functional, and complete with graphics, animations and levels,we expect at least:

- The player can acces a main menu allowing him to start a new game, load a save, adjust game settings and exit the game.
- In game, the feeling, pace and difficulty must be as close as possible from the original game.
- To beat the game, the player must win at least 2 different levels. It's up to you to code a longer game, but 2 levels is the minimum acceptable.
- If the player looses, the game must state it clearly and behave like a game should, like restarting the current level, or issuing a game over for instance. Likewise, if the player beats the game, it must also be stated clearly and offer a reasonable behaviour like displaying credits and getting back to the main menu.
- Sounds ! Musics ! A silent video game is a crappy one ! The main menu and the 3 levels must have their own music. Also, every relevant events must trigger a sound, like a bomb exploding, picking up a power up, killing an ennemi, dying, etc.
- Finally, one multiplayer local Arena that can be join (and play) by at least 4 players.



IV.2 Technical aspects

- The visual assets must be 3D, but the gameplay must stay 2D like the original game. For instance, if the player dies, the camera may shift its angle and spin around him while the death animation plays, or at the beginning of a level, the camera may fly around the level before focusing above the player. A good exemple of a 3D game with a 2D gameplay might be `Mario Wii`.
- Free animated 3D models of a bomberman and its ennemies might be hard to find online. As a consequence, you can skin your game as you wish and use any animated 3D model you like. The gameplay and visual feeling must be a `Bomberman` at first sight, but the actual models and textures can be anything.
- As stated in the previous section, your main menu must allow to customize the game settings. It must be possible to customize at least:
  - The screen resolution
  - Windowed or full screen mode
  - Key bindings
  - Music and sounds volumes
- The game must offer a way to save the player's progression and restore it on demand. It could be a regular save system or a campain mecanic. As long as the feature is present, it's up to you to integrate it in your game. Anyway, This persitance **MUST** remain even if the game is exited completly and executed again.

IV.3 Polishing and attention to details

As stated in the objectives section of this document, the goal of this project is not only to code a video game, but it's also to code a full finished video game. This means that the efforts you make to polish your game will count as much as the technical aspect during your defense. Do your best to avoid clunky animations or graphical glitches, manage your camera in a way that the "outside" of the game is never visible, pay attention to game design, level design and sound design, add variations in ennemies and environnements, etc.

A game that respects the technical and gameplay requirements but shows no effort in polishing won't result in a good grade. This project is about actually finishing a credible product, not toying with `OpenGL`.

IV.4 Team management

`Bomberman` could prove to be a difficult project for unprepared and/or cocky teams. Do not jump in your code as soon as you are done reading this document. Take time to organize your team and your work.

There are very different aspects in this project, and it's not a good idea to try to let every member of the team work on everything. Assign roles, communicate, plannify progression, communicate, design clean `APIs`, communicate, document your code, communicate, use `GiT` branches, communicate, and of course, don't forget to communicate.

Chapter V

Bonus part

Here is a list of bonuses:

- Accurate classes diagram
- Full code documentation
- Stand alone game installer for `OSX` or `Linux`
- Intro, cut scenes and outro to give your game some sort of plot
- More than 3 levels
- controller support
- Random layout of the levels

You can add some bonuses of your own creation if you like, but most of the bonus points will be reserved for the above bonuses.

Exceptionnaly, any kind of online won't be rewarded as a bonus in this project, so don't invest time on it, your game is a offline game. Netcode in video games must be done the right way !

`The bonus part will only be assessed if the mandatory part is PERFECT. Perfect means the mandatory part has been integrally done and works without malfunctioning. If you have not passed ALL the mandatory requirements, your bonus part will not be evaluated at all.`

Chapter VI

Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your folders and files to ensure they are correct.

Large projects using graphical and sound assets tends to be huge in terms of storage. As a consequence, you **MUST NOT** push any graphical and sound assets on your `GiT` repositoy. You will be allowed to fetch them at the begining of your defenses. It is a very good idea to consider your assets like dependancies, exactly like libraries, testing their presence and acquiring them likewise.

During the defense, the grader will focus on the "finished product" vibe your game has. Technical skills are important, but being able to fully finish a product is as much important.