# BOMBER MAN

## Code Design Documentation

### Content

This document describes the architecture and design for the Bomberman application developed for WeThinkCode_ - Johannesburg, South Africa

Tanaka Malaba
mfmalaba@student.wethinkcode.co.za

# Introduction

This document describes the architecture and design for the Bomberman C++ application developed by WeThinkCode_ students in Johannesburg, South Africa. Bomberman is a hugely popular video game with over 70 adaptations worldwide. This is one such adaptation of the classic game in 3D.

## Premise

The player has to destroy enemies with bombs placed on the ground. To complete a level the player has to kill all gads by strategically placing bombs. A player may play one bomb at a time, each with a personal timer. The enemies also known as gads have patrol roads, which they do not leave and thankfully have only 1 hit point. The bombs cannot break the labyrinth's walls and kill gads on the other side of them. The bomb's detonation kills all gads and Bomberman if they are inside the range. Unlike the gads Bomberman is harder to kill. He has 3 hit points, which can be reduced to 0 by suicide bombing or getting caught by gads.

## General Implementation

This Bomberman follows a particular implementation:

- It is a solo game requiring only one player
- Contains a main menu allowing player to start a new game, load a save, adjust game settings and exit the game
- In game experience very similar to the original Bomberman
- Three levels must be won to win the game
- Multimedia aspects and interactions throughout the game

## Technical Implementation

Technically the game is:

- Written in C++
- Makes use of OpenGL for its graphics
- Utilizes ample 3D models and animations
- Allows for the customization of; screen size, key bindings and volume
- Capable of saving progress in persistence

## Design Goals

There is no absolute measure for distinguishing between good and bad design. The value of a design depends on stakeholder priorities. For example, depending on the circumstances, an efficient design might be better than a maintainable one, or vise versa. Therefore, before presenting a design it is good practice to state the design priorities. The design that is offered will be judged according to how well it satisfies the stated priorities.

The priorities for the design that follows are:

- The design should minimize complexity and development effort.
- The design should take into account the development environment which is a team of 4 with complementary skills that work across time and space (teams are not co-located).
- The design shouldn't inhibit reusability. The two previous design goals are more important, but the ability to reuse components is also desirable.

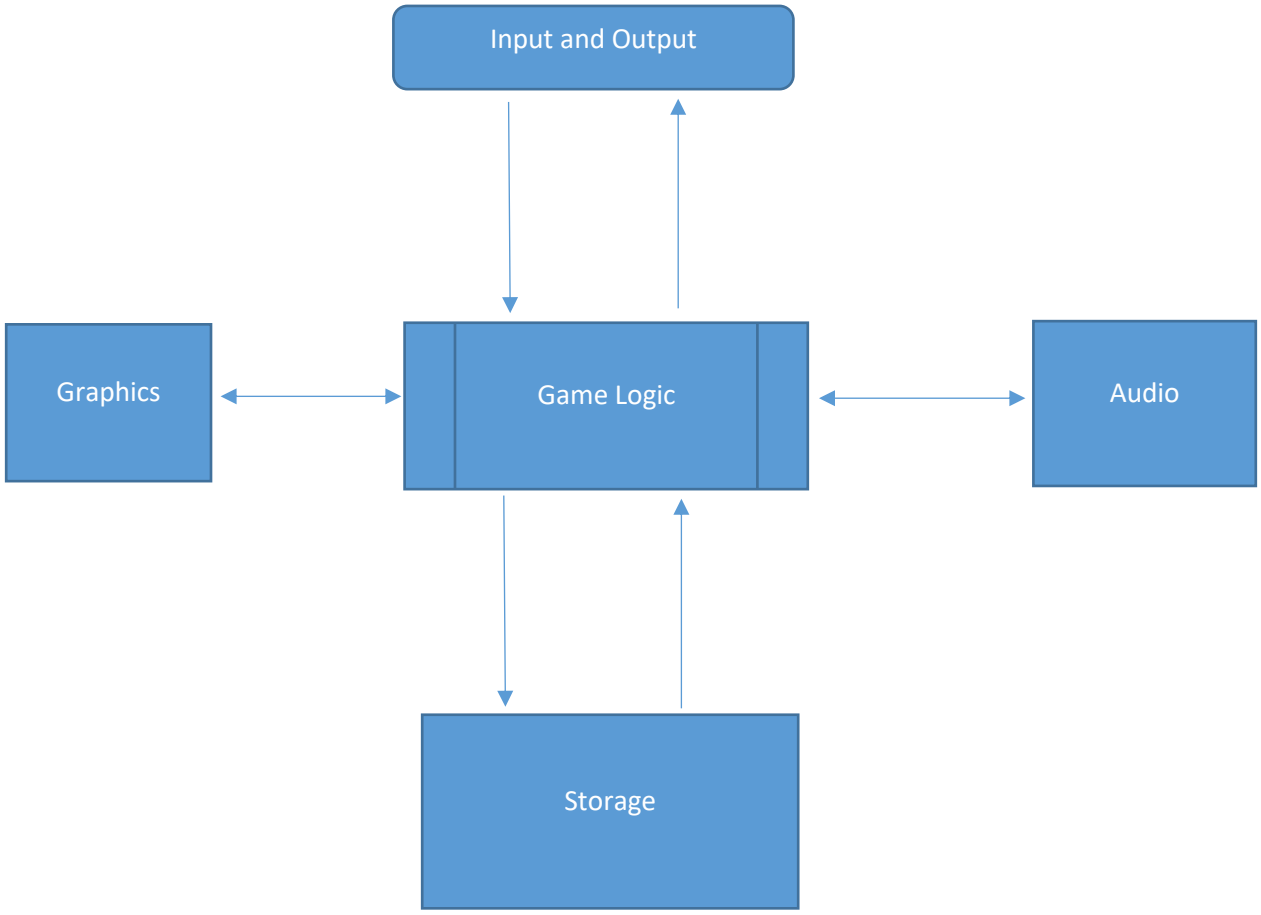
## Logical View

The logical view describes the main functional components of the system. This includes modules, the static relationships between modules, and their dynamic patterns of interaction.

### High-Level Design

In this section the modules of the system are first expressed in terms of high level components.

- Input and output is handled by the keyboard and screen respectively
- The Game Logic makes the necessary computations and renders the appropriate shaders and audio
- Audio provides the sounds
- Graphics handle the 3D viewing
- Storage is used to store game data and progress in a persistent fashion




### Mid-Level Design


Graphics
OpenGL
Shaders
Loaders
Rendering
Entities
Game Logic
Player
AI Opponents
Constraints
PowerUps
Audio
Sounds
Parameters
Storage
Settings
Status
General