Documentation

Bomberman 3D

Marcin Bąk
Kamil Depta
Marcin Kozioł
Artur Frankowski

Table of content

1. Introducing

1.1. Project scope

The aim of our project was to design and create a 3D game written in Python. The game we chose as the prototype for our designed game was the popular Bomberman 2D. The main task was to transfer the functionality of the game from the level of a two-dimensional platformer to a three-dimensional world and to implement the rules of the game.

An additional goal was to create technical documentation to facilitate the understanding of game development with the Ursina engine and make it available on GitHub. In this way, users wishing to explore this free engine can better understand the methods and arguments of the available classes in a practical application.

1.2. Functional requirements

- First person view
- Collision mechanism with the edges of the map, walls and enemies.
- Opponents tasked with defeating the player and who could be destroyed with weapons.
- Bombs
- Messages that start and end the game.
- Sounds that indicate important in-game activities.
- Upgrades, power-ups that can be collected by the player.

2. User documentation

2.1. Requirements

Python and Ursina engine installed on the computer.

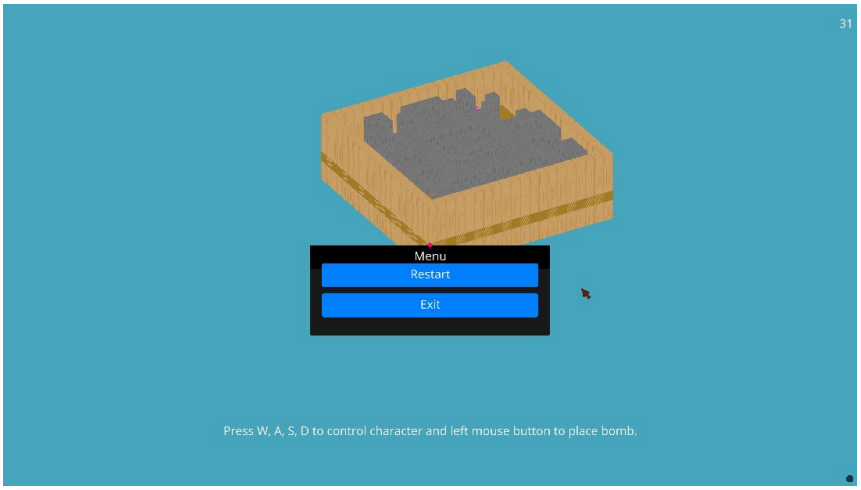
2.2. The basics of the game

In the game, we will face 3 opponents. The goal of the game is to eliminate the opponents on the map. We have bombs at our disposal, which we set when none of our other bombs are in use. After a while, an explosion takes place, destroying nearby walls and any enemies. Along with the destruction of the map infrastructure, we have a chance to get an upgrade. The upgrade can give you a greater bomb range or a greater number of bombs you currently have.

2.3. Working with program

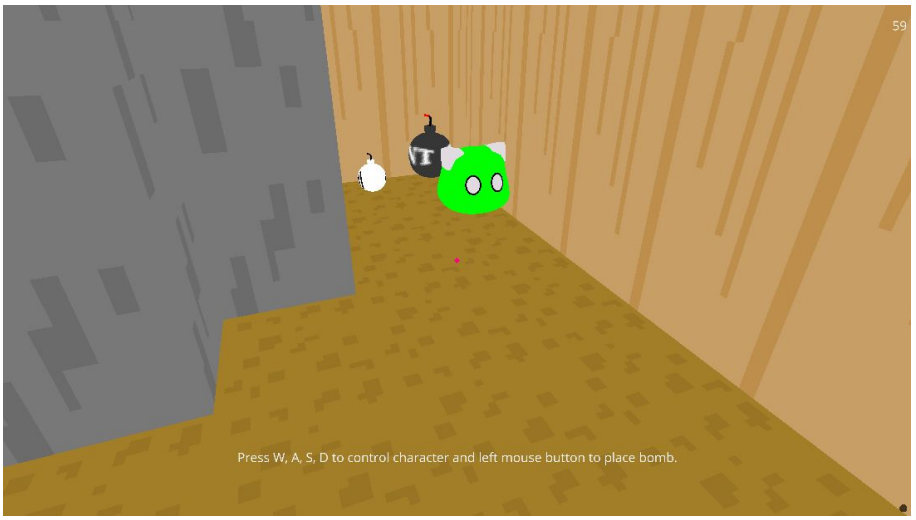
2.3.1. Start of the game

At the start, the player and opponents appear in the corners of a symmetrical map. We have at our disposal such functionalities as moving around the map or placing bombs. Under the escape key, we have a menu from which we can restart the game or exit it.



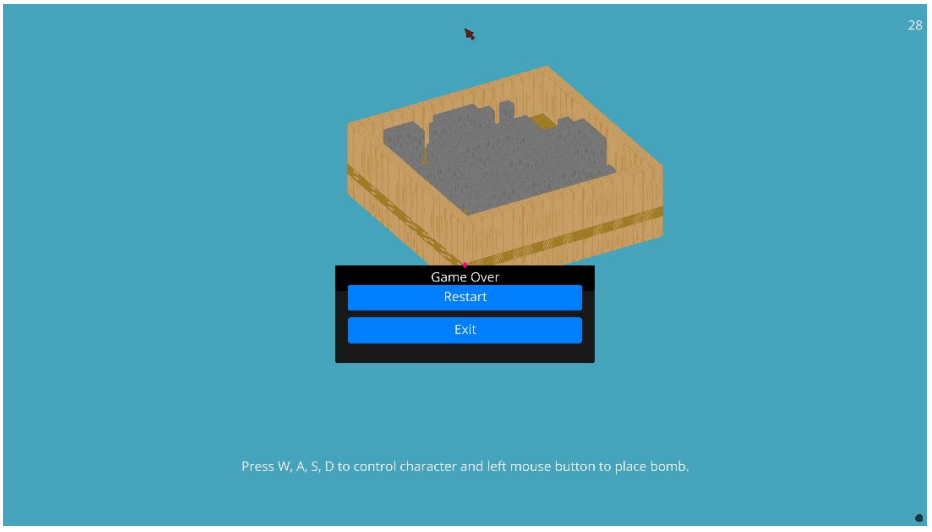
2.3.2. Opponents

Each opponent moves around the map, placing bombs at the last encountered obstacle. Like the player, he can get upgrades. Unlike the player, opponents do not die from their own bombs.



2.3.3. Death

Death occurs when the player is within the bomb's firing range when the bomb explodes. It doesn't matter who owns the bomb. A window will appear on the screen with information about the loss and adequate music will be played.

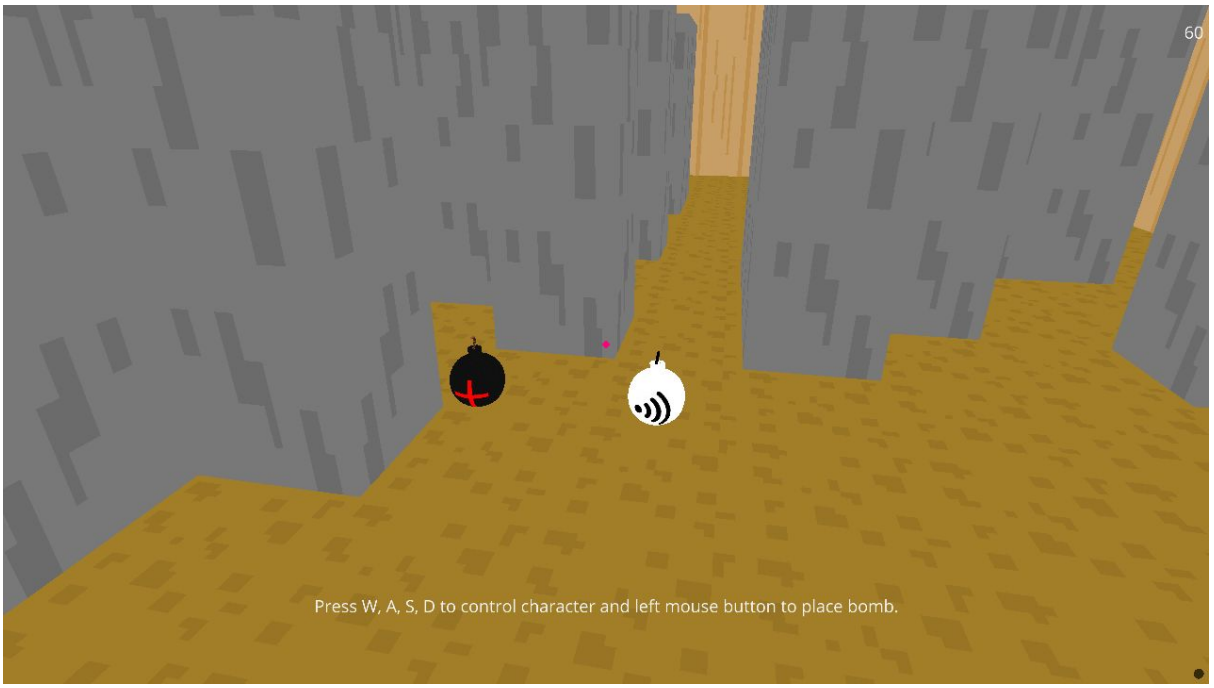


2.3.4. Buffs

In the game there are two types of buffs:

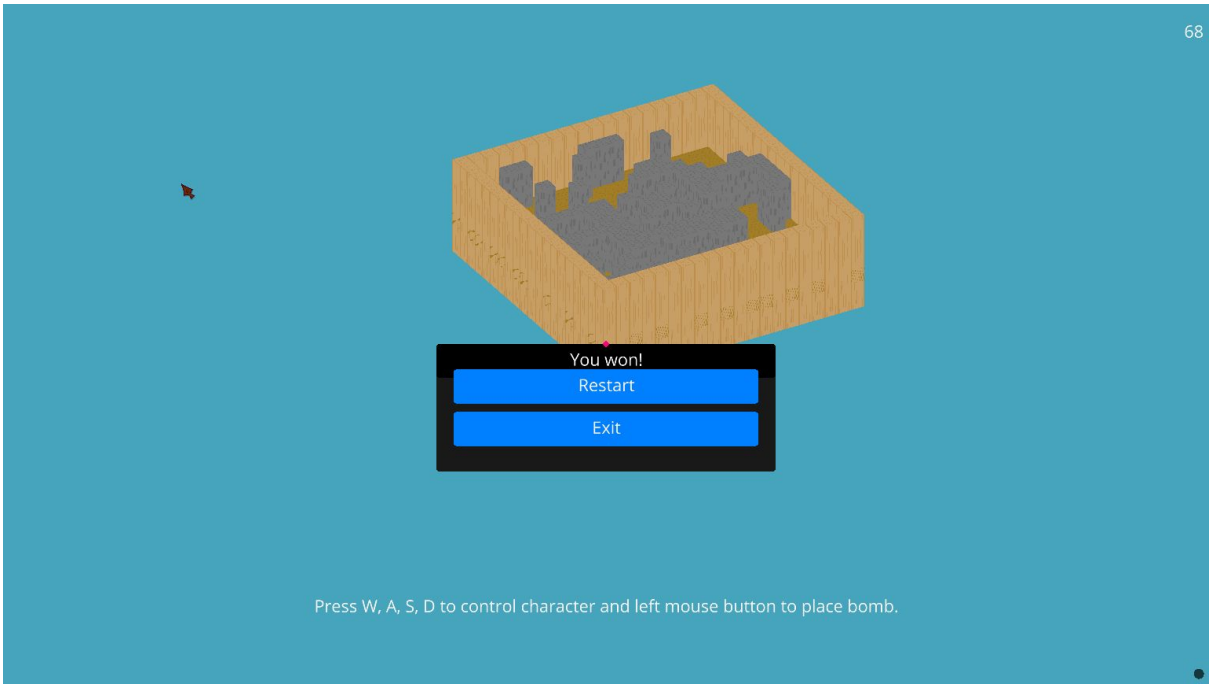
- increasing the limit of bombs that can be placed at the same time (black);
- increasing the range of a single bomb explosion (white).

Each time you collect an upgrade, it permanently increases the given statistic of your character. Upgrades appear in the place of a destroyed wall with a 25% chance, while the type of upgrade has a 50% chance for each type.



2.3.5. Win

The player wins when all opponents have been eliminated. The screen will display the information about the victory, and adequate music will be played.



3. Technical Documentation

3.1. Architecture description and file structure

3.1.1. blocks.py

It contains definitions of objects responsible for simple collisions with the player's character, they define before which block the player has to stop and where he can place the bomb. All class objects in this file have cube collisions and only differ in size along the Y axis.

Ground

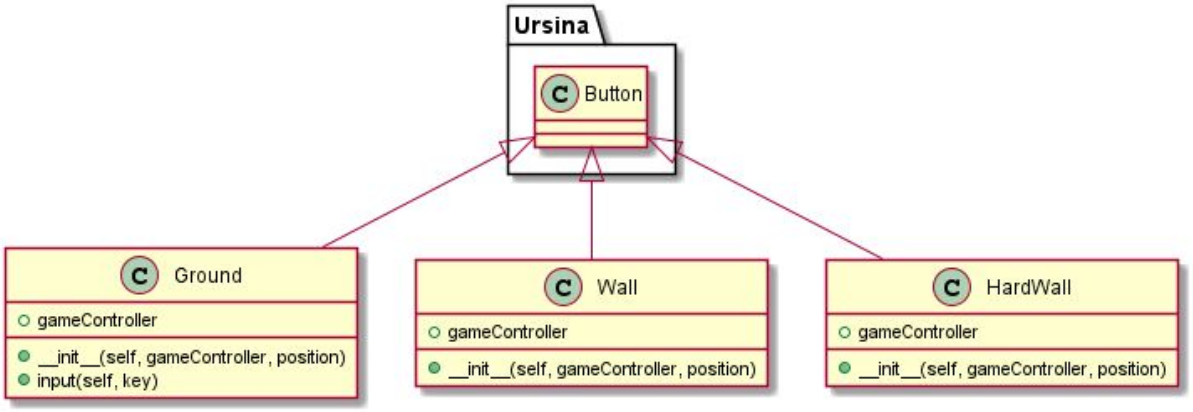
The Ground class is responsible for the player's interaction with the ground. Objects of this class cannot be destroyed by the player, and only on them the player can place the bomb.

Wall

Wall blocks are destroyed when interacting with objects from the Explosion class.

HardWall

Blocks based on this class cannot be destroyed by bombs and the player cannot place bombs on them.



3.1.2. bomb.py

This file has class implementations that define how the explosion mechanics work.

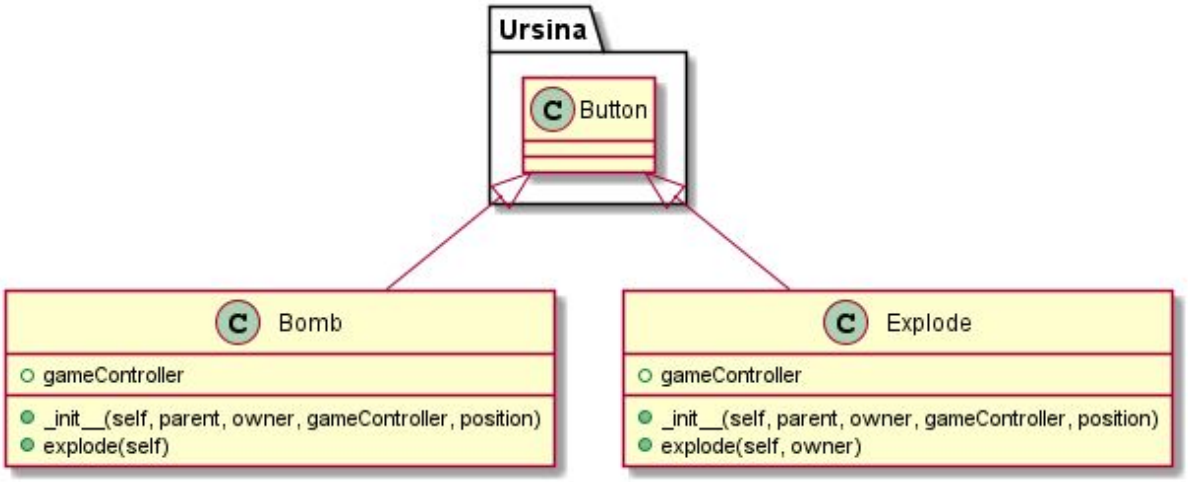
Explosion

They are created when the bomb timer ends. They destroy enemies and Wall class objects when they detect a collision. When they detect a collision with a player, the game is over and the end of the game screen is displayed. Each explosion has a specific range that tells you how many explosion objects are to be created side by side. Each explosion is assigned to an opponent or player. Enemies cannot be destroyed by explosions caused

by your own bombs. When an object is created, a timer is started that destroys it after a certain time.

Bomb

They are created by opponents at random intervals and by the player when clicking the left mouse button on Ground objects. Creates Explosion objects and is deleted when they are deleted.



3.1.3. bomber.py

Bomber

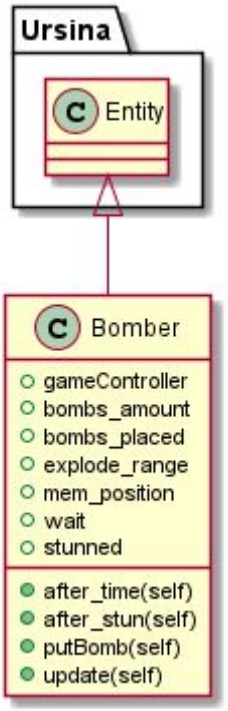
This class contains the logic of the opponents in the game. The most important methods are:

- **putBomb**

  The putBomb method creates a bomb at an appropriate distance from the opponent and determines how many bombs the opponent can place in a certain period of time.

- **update**

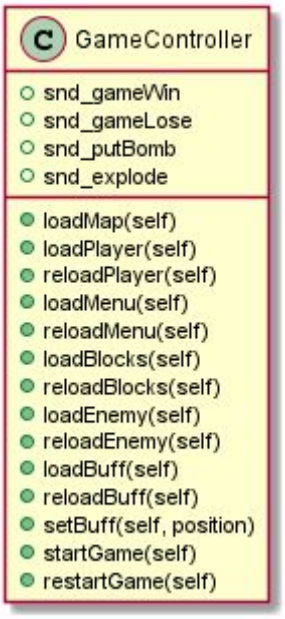
  This method is performed cyclically and includes the logic of the opponent's movement. It contains three Raycast class objects that create rays for collision detection, successively with bombs, enemies and walls. When a collision with a bomb is detected, the enemy will start to run away from it in the opposite direction. When it detects a collision with another enemy, it will plant a bomb and run away in a different direction. If a wall is detected, the enemy will change direction, or also he can place a bomb at the wall at random.



3.1.4. Game.py

GameController

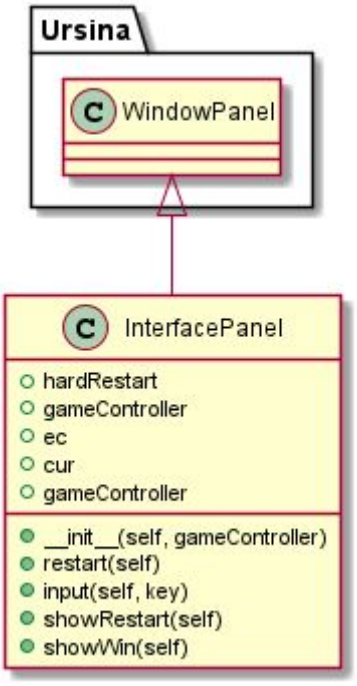
There is only one instance of this class in the game and it is responsible for creating Bomber, Ground, Wall, HardWall, and MyFirstPersonController objects. It contains a two-dimensional array of variables of the int type that stores information about the objects to be created on the board. Each object in the blocks.py file is assigned a different int value. Loads sound files into memory and allows you to reset all objects on the board.



3.1.5. menu.py

InterfacePanel

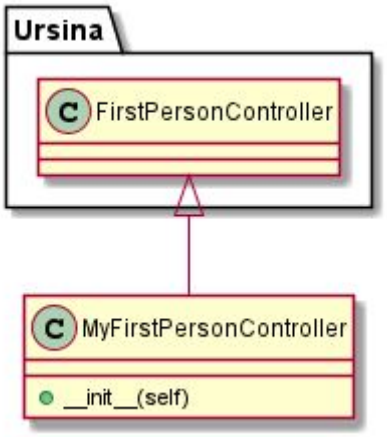
The InterfacePanel class inherits from the WindowPanel class from the ursina package, which allows us to create a simple user interface in the form of a window. If the object is created by pressing the Escape key, the text Menu will be displayed on the window, if you win! Game Over in case of defeat.



3.1.6. myFirstPersonController

MyFirstPersonController

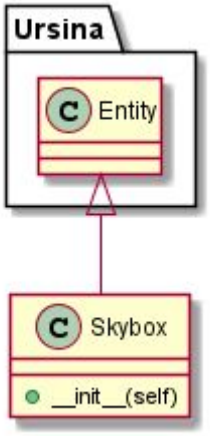
It inherits from the FirstPersonController class in the ursina package and specifies the basic parameters for the player's character. Size, range of the explosion, number of bombs placed at the moment, number of bombs, collider type, current and starting position.



3.1.7. Skybox

Skybox

Creates a simple skybox in the form of a sphere placed in the game. From the inside, this object contains the texture of the sky. Additionally, it displays the controls when you first start the game.

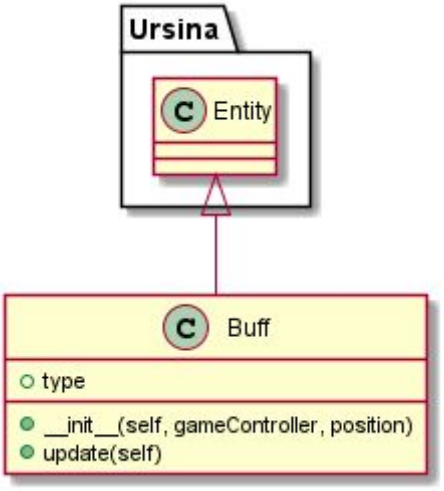


3.1.8. buffs.py

The class is responsible for the operation of buffs in the game. Objects of this class are created randomly when an Explosion object destroys a wall. When creating objects, buffs are drawn with their type. Currently, two types of improvements are implemented in the game, increasing the range of the explosion for a specified period of time and increasing the number of bombs that can be placed in the game.

3.1.9. constants.py

This file contains all constants used in our project.



4. Sources

- https://www.ursinaengine.org/cheat_sheet_dark.html
- https://www.turbosquid.com/3d-models/cartoon-bomb-obj-free/1034107
- https://www.zapsplat.com/