Bilkent University
CS 319

Bombplan

Design Report

Group #1

Asena Rana Yozgatli - 21000132
Berk Yurttas - 21200581
Mehmet Furkan Sahin - 21201385
Saner Turhaner - 21100475

Ugur Dogrusoz

1. Introduction

Bombplan is a brand-new version of classical Bomberman game. The main aim in Bomberman is to reach to exit door without being killed by the monsters in the game. The main character has the power to plant the bombs in different locations on the map. The map is in form of a maze and there are some wall blocks that can be destroyed by the bombs and some others that can never be destroyed. The character starts from one end of the maze and by destroying the suitable walls and running away from the monsters, tries to reach to the exit door. The bombs are not only affective on the walls, if there is the main character or one of the monsters in the range of the bomb, they can also be destroyed.

In this report, the main design of the Bombplan will be discussed. We will provide Bombplan with an OOP implementation. The following sections will lead to discussions related to requirement analysis, and the analysis with object models and dynamic models. During the requirement analysis part we will examine functional and non-functional requirements, constraints, scenarios, use case models, and user interface of Bombplan. At the end, we will conclude with a general revision of the whole report.

2. Requirement Analysis

1. Overview

Bombplan is a desktop application that will be created with Java. Game instructions will be as following;

- The main character (from now on, it will be referred as bomber) will have only the ability to plant a bomb.
- A bomb has a range of one wall block from 4 sides. The waiting time for the explosion of the bomb is 3 seconds, default. The bomb features can be changed by the taken tokens that are randomly distributed inside of the walls on the map. The features that can be changed are as following;
  - Bomb explosion can be made up to the user. By the help of a taken special bonus, bombs do not explode themselves in 3 seconds, instead, they wait for the user to explode it.
  - The range of the bomb is 1 wall block from 4 sides -default-, but it can be extended by a special token. After the token, the range increases 1 more wall block in depth from 4 sides. Since, user can take from that special token more than once, at some point, range can be 4 wall blocks in depth from 4 sides.
  - Normally, user cannot plant one more bomb if there is already a planted bomb. However, by the help of another token, user can plant multiple bombs. Each token increases 1 the maximum number of the bomb that can be planted at once.
- Bomber has 3 lives at starting. This can be increased with hearth shaped tokens by one and of course, it decreases by dying.
- Bomber can die by being in the range of an exploding bomb, fail to kill all of the monsters and find the door in the given time or touching to one of the monsters.

- There are 2 types of monsters -slow and fast- in the game. The number of the monsters from these types change according to the level. Total number of monsters also change according to the level.
- There are 2 types of walls in the game. One of them can never be destroyed by bombs, the other can. Their design changes according to the type of the wall and of course all of the tokens are in the walls that can be destroyed.
- Number of blocks will be definite for each level. However, the blocks will be distributed randomly on the map. Of course there will be some constraints for the distribution of the walls to provide a more homogenous map.
- To pass a level, bomber need to kill all of the monsters and find the secret door inside of one of the wall blocks in the given time.
- To finish the game, bomber should pass 3 levels of maps or spend all of his lives.
- During the game, user earns some points with destroying walls, killing monsters, taking tokens or passing levels. At the end of the game, the collected point is put in high-score list if it is one of the highest 10 scores ever played on that computer.

2. Functional Requirements