Requirements and Analysis Document for Pyromaniacs

Version: 2.1
Date: 2012-05-08

Author: Andreas Rolén, Adrian Bjugård, Joakim Persson, Viktor Anderling

This version overrides all previous versions.

Table of content

1. Introduction

1.1 Purpose of application

The goal is to build a game application, similar to Bomberman. The main goal is to create the game just like how the original game looks and then if there's any time over add extra features like saving your profile, play over network and more boosters for you character in game.

1.2 General characteristics of application

The game will be a desktop application with same-keyboard multi-player support. The idea is to start with two players and then extend it to four, either on the same keyboard or with external Xbox controls. When a new game starts, every player is placed in each corner of the gamefield. Then the mission is to place out bombs to blast boxes and pillars to reach the other players, and then blast the players. In some of the boxes that you blast there is placed power-ups which can give you higher speed, more bombs or bombs with better range. The pillars require more bomb power to destroy than the boxes. During every match the players will be able to purchase upgrades. Some examples of possible upgrades are: health, more bombs or even more powerful bombs etc. The idea is that a game consists of a several matches. Every match consists of three rounds and between every match the players will be able to buy upgrades for the points they earned.

1.3 Scope of application

The game will only support multiplayer, at the same computer and not playing against a player controlled by the computer.

1.4 Objectives and success criteria of the project

The objective is to be able to play a game with more than one round with two players in any GUI based platform.

1.5 Definitions, acronyms and abbreviations

All definitions and terms regarding the core Pyromaniacs game are as shown in the references section.

- GUI, graphical user interface.
- Java, platform independent programming language.
- JRE, the Java Run time Environment. Additional software needed to run an Java application
- Host, a computer where the game will run.
- Round, one complete game ending in a winner.
- Match, several complete rounds ending in a winner.
- Score, based on several different factors.

2. Requirements

2.1 Functional requirements

The players should be able to:

1. Select how many players for the game. Two, three or four.
2. Start a new match.
    a. Move the character.
    b. Place a bomb (Which will explode).
    c. Pick up an item.
    d. Get an “item-boost” (Speed, bombrange, numbers of bombs).
    e. Get hit by a bomb.
3. Enter the shop.
    a. Buy an ending “boost” (Hit points, life, more score/item etc.) that only last a match.
4. Save the score.
5. See the highscore of the game.

2.2 Non-functional requirements

2.2.1 Usability

The usability is very important in this game. A new user should be able to start a new game in a short time and understand how it works in less then a half minute.
Tests should be performed by two persons to see if the games usability is in a clear fashion.
The language of the game is English and we don’t see any need of another version since there’s not much text.

2.2.2 Reliability

NA

2.2.3 Performance

Any action by a player should not exceed a 300 ms. response time in worst case.

2.2.4 Supportability

The application should be implemented so that the GUI is easily modifiable to suit the web.

Also, after we implemented the network based version it should be able to modify it to other platforms such as mobile apps, pads, etc.

2.2.5 Implementation

To achieve platform independence the application will use the Java environment. All hosts must have the JRE installed and configured. The application needs to be installed on all hosts where it will run (possibly downloaded).

2.2.6 Packaging and installation

2.2.7 Legal

There could be legal issues regarding rights to the Bomberman game and trade mark. This is not covered here.

2.3 Application models

2.3.1 Use case model

See appendix for UML and textual description.

2.3.2 Use cases priority

1. Move
2. Place bomb
3. Bomb explode
4. Destroy block
5. Hit player
6. Kill player
7. Turn end
8. Pick up item
9. Boost player
10. Match end
11. Scoring

2.3.3 Domain model

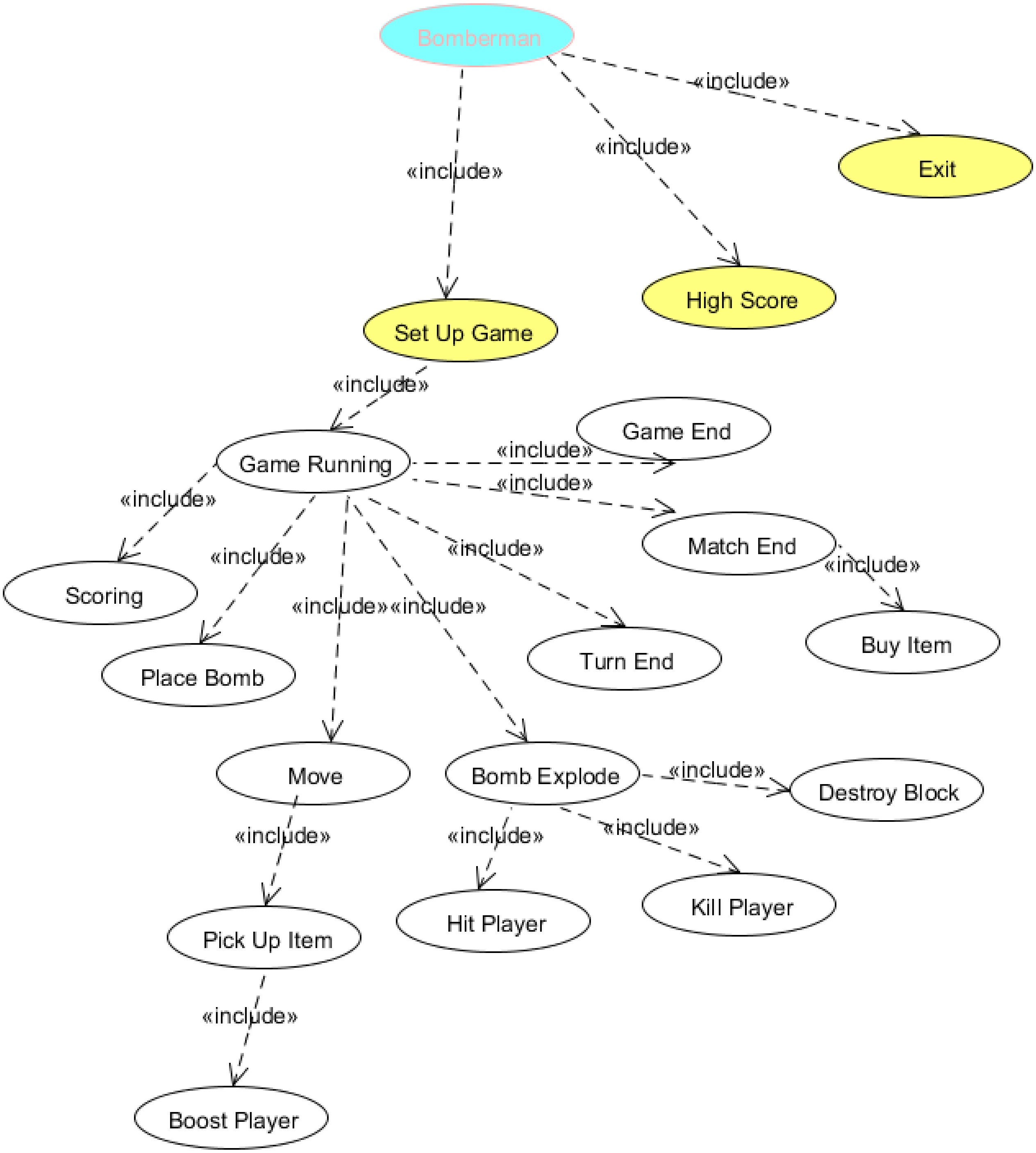
See appendix.

2.3.4 User interface

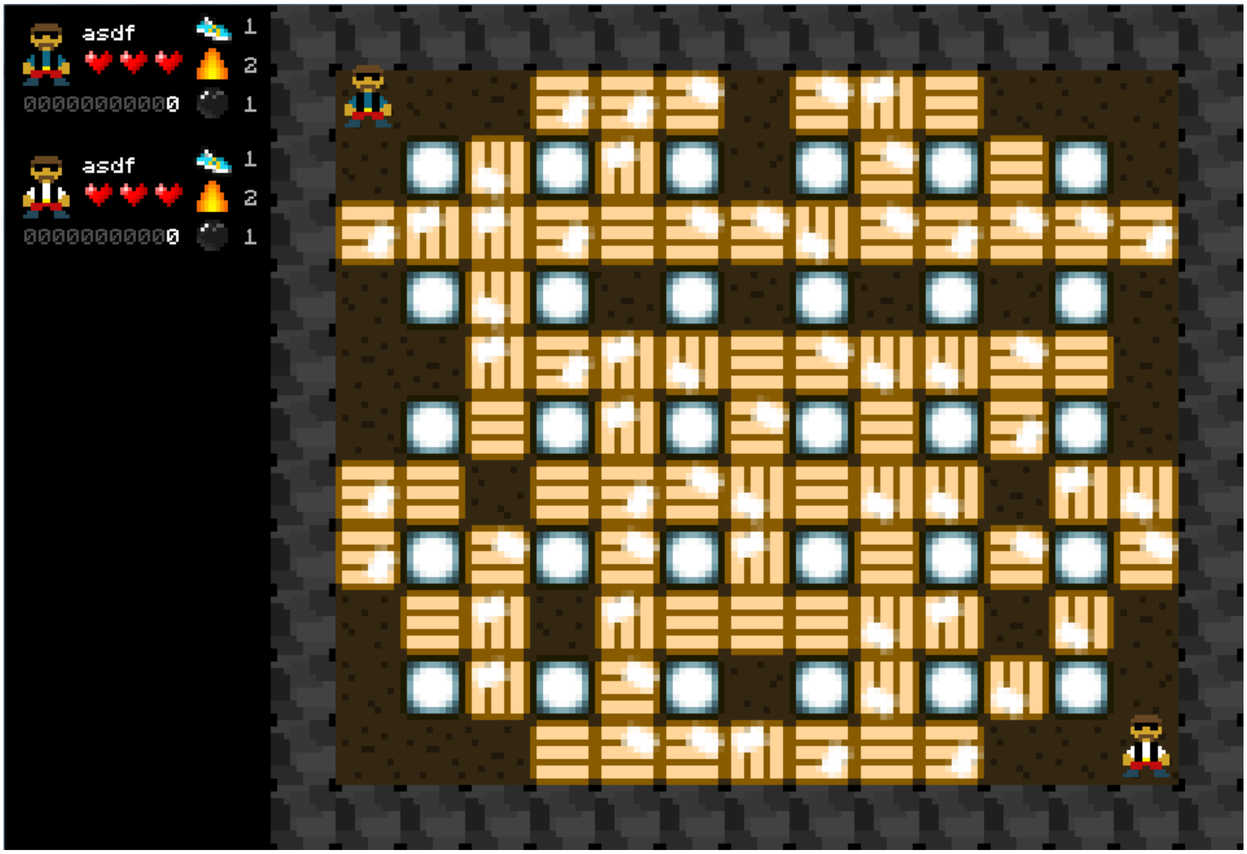
The GUI will be a 955x650. See the appendix for print screens.

APPENDIX

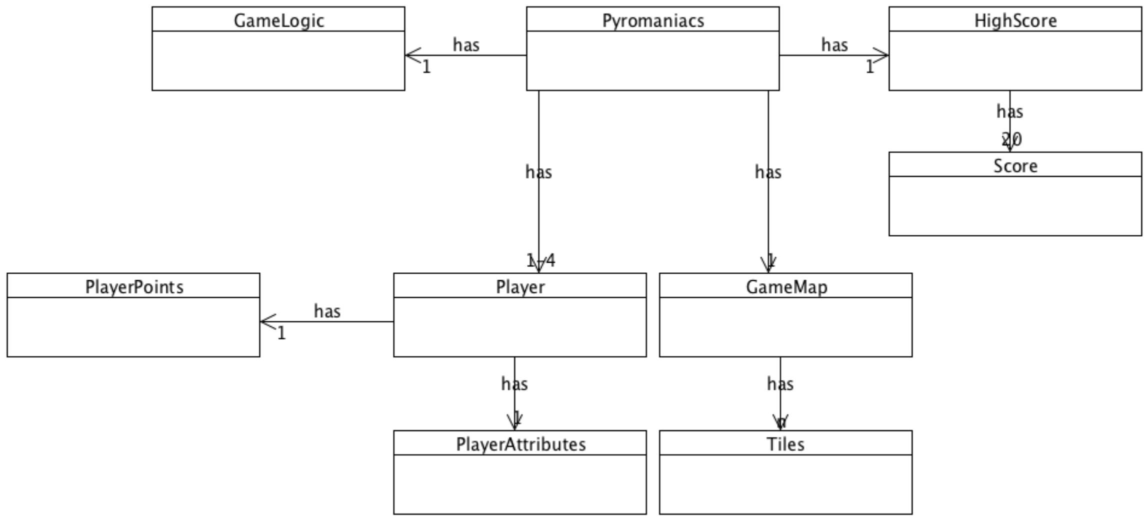
Use Case diagram



GUI



Domain Model



Use Cases

Use Case: *Bomb Explode*

Short description: When a placed bomb explodes
Priority (high, mid, low) : High
Extends or Includes (other use case): Includes "Hit Player", "Destroy Block"
Participating actors: The current placed bomb, All players, Blocks
Normal flow of events

Actor	System
Bomb exploded	
	The bomb explodes in four directions (East, West, North, South). Its range in the four directions is based on the players picked up items.
	Increases the player who placed the bombs BombStack.
If one or a combination of the Actors is in the range of the explosion. UC: (“Player hit", "Destroy Block")	

Alternate flow - No actor is in the range of the bomb

Actor	System
	The bomb only explodes

Alternate flow - Area bomb

Actor	System
AreaBomb explode	
	The bomb explodes in a area with a size decided by the items the player has.
	Increase the player who placed the

	bombs BombStack.
If one or a combination of the Actors is in the range of the explosion. UC: (“Player hit", "Destroy Block")	

Exeptional flow

None

Use Case: *Move*

Short description: How the user moves his character
Priority (high, mid, low) : High
Extends or Includes (other use case): Includes "Pick up item” and “Hit Player”.
Participating actors: The actual player
Normal flow of events

Actor	System
Presses any direction key	
	The players character moves pixelwise the same direction as the key pressed.
	If the players character and some other players character moves to the same place, the player with the highest player index, e.g. “Player 4”, will be drawn on top of the other.

Alternate flow: Speed-boost

Actor	System
Presses any direction key	
	The players character moves pixelwise the same direction as the key pressed. Depending of how many “Speed-boosts” the player picked up the player will move faster.
	If the players character and some other players character moves to the same place, the player with the highest player index, e.g. “Player 4”, will be drawn on top of the other.

Alternate flow: Get hit

Actor	System
Presses any direction key	
	The players character moves pixelwise the same direction as the key pressed.
	If the player moves his character into an exploding bomb, the character will get hit. See “UC: Hit Player”

Use Case: *Place bomb*

Short description: How a player places a bomb.
Priority (high, mid, low) : High.
Extends or Includes (other use case): Includes "Bomb explode”.
Participating actors: The actual player.

Normal flow of events:

Actor	System
Presses corresponding key for placing a bomb	
	The game shows a bomb placed on an exact position on the fields grid and plays a “bomb placed” sound. One bomb is temporarily removed from the players BombStack, which has a numerical representation in the GUI.

Alternate flow (bombstack is empty):

Actor	System
Presses corresponding key for placing a bomb	
	No bomb is placed.

Exeptional flow: No exception.

Use Case: *Hit player*

Short description: When a player gets hit by a bomb
Priority (high, mid, low) : High.
Extends or Includes (other use case): Includes "Kill Player”.
Participating actors: Any player.

Normal flow of events:

Actor	**System**
Player with more than one hit point left gets hit by running into or standing in the fire from any bomb.	
	Reduces the players hit points by one.

Alternate flow: Player dies (only one hit point left when getting hit by bomb)

Actor	**System**
Player gets hit in the same way as above with only one hit point left.	
	See UC: Kill player

Exeptional flow: No exception.