Requirements Analysis Document (RAD) for Blast in the Past

Version: 2.0
Date: 2015-05-31
Authors: Bjarne Gelotte, Jonas Graul Sagdahl, Mattias Johansson

This version overrides all previous versions.

1. Introduction

"Blast in the Past" - A present day soldier gets sent back in time to the Middle ages to fight knights. Top-down shooter with 2D-graphics and enemies with simple AI, powerups, levels, etc.

1.1 Purpose of application

A game solely made to be fun to play. It should also be a challenge so the user will be proud of him-/herself if they beat it.

1.2 General characteristics of application

The game will be a top-down, single-player 2D-shooter with retro graphics and lots of action. It will take place in the Middle ages, where the user plays as a character that has been sent back in time. There will be different levels and the character will be able to pick up weapons and defeat enemies (medieval warriors). The character will have a health bar and some sort of inventory which will allow the user to switch between weapons or equipment.

1.3 Scope of application

The game should be 10 levels long. Each level should adhere to an overarching medieval theme but also be distinguishable from each other, both graphically and in terms of how the level plays out (difficulty, which enemies appear and the like). The levels should be between 2 and 5 minutes long. Every even numbered level is a boss stage, meaning that the level ends with the user facing a particularly strong enemy or sets of enemies. There should be an inventory system which can be used to manage items and equipment. The game should have 3 difficulty settings: easy, medium and hard. Harder difficulties will increase the amount of health enemies have and decrease the amount of damage the PC can take before the user loses. On harder difficulty settings the enemy AI will be more aggressive towards the character. The game should include some puzzles where the user must find or activate certain objects (levers, buttons) to progress. The game should include power-ups like bullet time and double damage. There should be 5 different normal enemy types and 5 different boss monsters. The levels should have some basic obstacles like rocks and trees. Music and sound effects should be included in the game. The sound effects should accompany appropriate actions like firing weapons, walking, picking up power-ups and so on. The user will also need to solve simple puzzles in order to progress (activating switches, picking up keys to unlock doors and chests, etc.).

1.4 Objectives and success criteria of the project

The game should consist of two maps (two levels), on level two there should be a boss encounter. The user should be able to move the character freely in a room where there are no obstacles. Each map/level should have its own weapon which is hidden somewhere within the map that the user should be able to loot. The user should then be able to choose which weapon to equip through an inventory system.

1.5 Definitions, acronyms and abbreviations

HP - Health Points, signifies the amount of damage a character can absorb before dying.
2D - Two dimensions
AI - Artificial Intelligence
Ammo - Ammunition

Boss - A particularly tough and dangerous enemy
Power-up - An item which gives the PC certain positive abilities like increased damage or faster movement for example. Expires after a short time (less than a minute).
NPC - Non-Player Character, a character that is not being controlled by the user.
PC - Player Character
GUI - Graphical User Interface

2 Requirements

In this section we specify all requirements.

2.1 Functional requirements

A user should be able to:

1. Move in eight different directions (N, NE, E, SE, S, SW, W, NW)
2. Fire a weapon
3. Kill an enemy.
4. Die when his/her HP is depleted
5. Pick up weapons
6. Switch between weapons
7. Automatically come to the next level once he/she has cleared a level
8. Turn volume on/off
9. Change keybindings
10. Pause the game to open a menu
11. Exit to main menu via in-game menu
12. Create a new save file which will save the users progress
13. Load saved games
14. Overwrite previous saves
15. Start the application
16. Close the application
17. Activate power-ups dropped by enemies by walking over them
18. Aim the weapons with a mouse
19. Get a clear indication of where the PC is aiming
20. Create a new game
21. Be damaged by an enemy
22. Damage an enemy
23. Pick up ammo
24. Reload weapon
25. Collide with an enemy
26. Collide with obstacles on the map

An enemy should be able to:

1. Damage the user
2. Be damaged by the user
3. Reload their weapons
4. Die when their HP is depleted
5. Drop ammo when they die
6. Drop a power-up when they die
7. Damage other enemies

8. Collide with the PC and other enemies
9. Collide with obstacles on the map

Items marked with red have not been implemented.

2.2 Non-functional requirements

2.2.1 Usability
Keybindings should be of conventional nature. For example to move you should use “W”, “A”, “S”, “D”, reload with “R”, use items with “E”. The game should give clear feedback of what happens in the game when someone takes damage or loots a weapon. The menu system should be easy to use and intuitive.

2.2.2 Reliability
Saved data should not be lost and unexpected crashes shouldn’t occur. The saved data in this case is high scores for when a player get enough points to make it into the high score list.

2.2.3 Performance
The game should run smoothly in terms of framerate and be able to be tabbed down and up easily. The application should start fairly fast when executed.

2.2.4 Supportability
Help should always be accessible in the menu. It should be described in a README file how the program can be executed.

2.2.5 Implementation
The code should be well written and well structured. The code should also be written in Java to achieve platform independence.

2.2.6 Packaging and installation
The game will be packaged to a jar-file using Gradle and no installation will be needed. A README-file will also be included.

2.2.7 Legal
N/A

2.3 Application models

2.3.1 Use case model
Our detailed use cases include:

- Fire weapon
- Loot from chest
- Pick up power-up
- Projectile collision with character
- Collision between characters

See appendix for UML diagram and textual descriptions.

2.3.2 Use cases priority

1. Fire weapon
2. Projectile collision with character
3. Collision between characters
4. Loot from chest
5. Pick up power-up

2.3.3 Domain model

See appendix.

2.3.4 User interface

See appendix for sketches.

The in-game UI consists of:

- Heart icons, to symbolize the characters health
- Top-down view of the map with the PC in the center of the screen
- An indicator of which weapon is currently equipped and how much ammo you have, in the bottom left corner of the screen

You will aim with the mouse, which will look like a crosshair. The inventory will hide itself when not active, to keep the UI clean.

The main menu will consist of the following items:

- New game
- Load game
- High scores
- Options
- Quit

The in-game menu will consist of the following items:

- Continue
- Save game
- Options
- Exit to main menu

Appendix

GUI

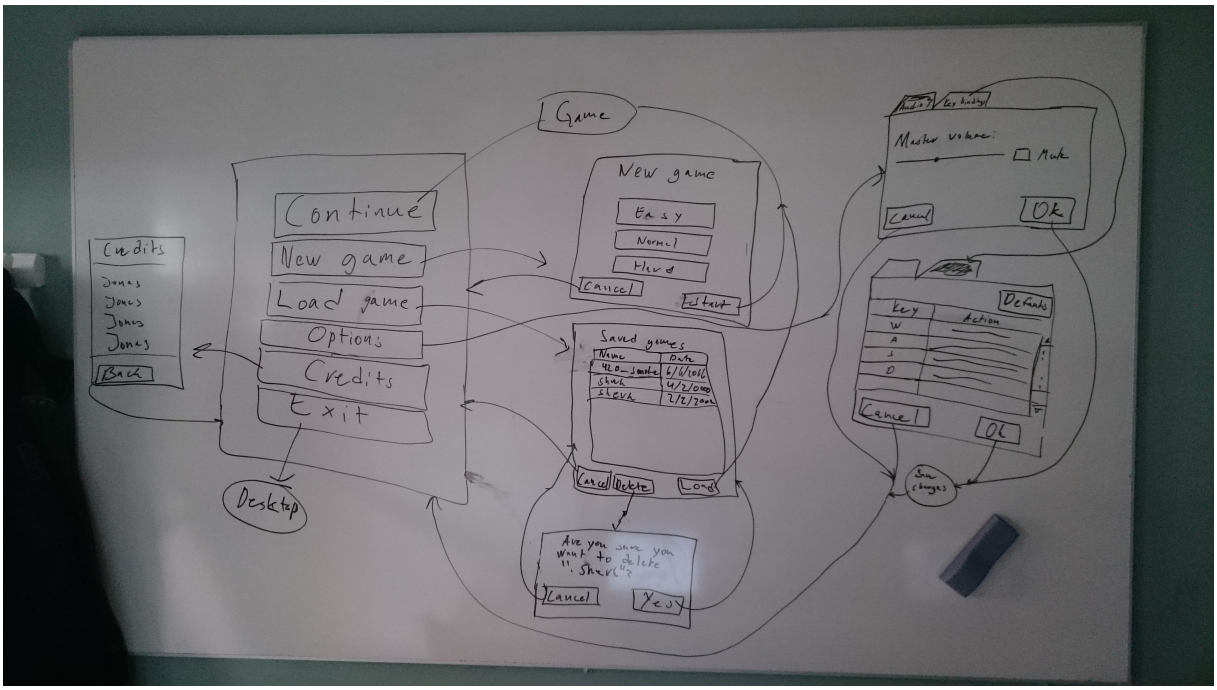

Figure 1. Main menu model.

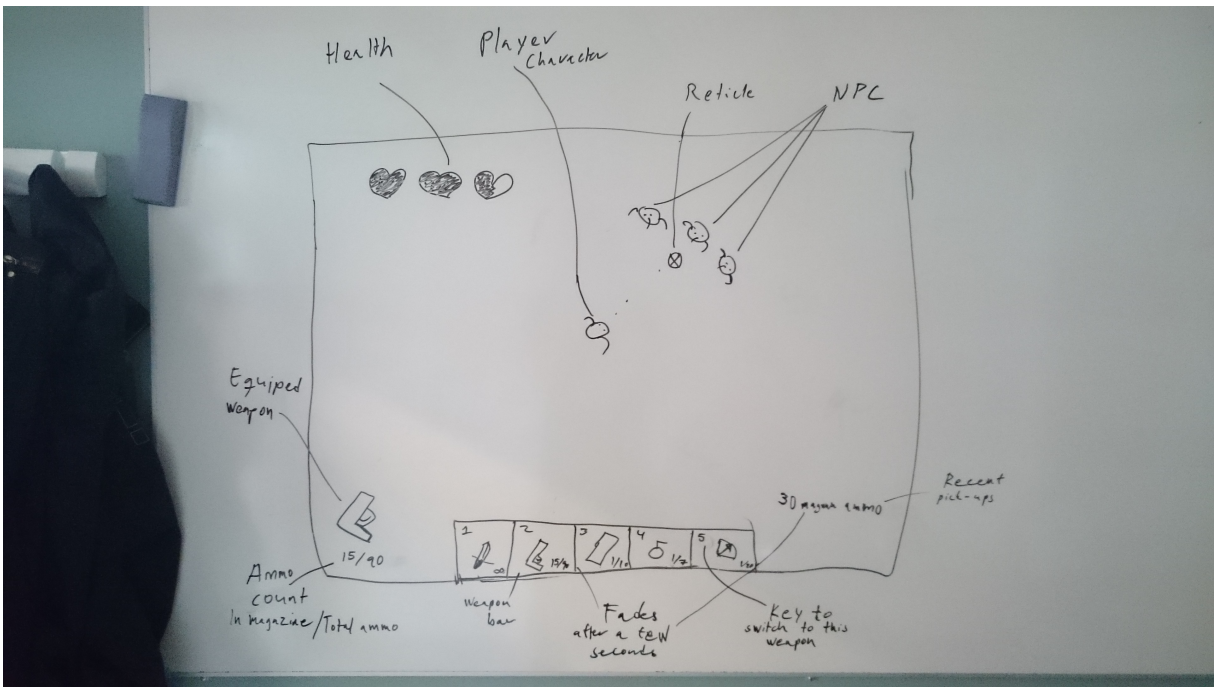

Figure 2. In-game GUI.

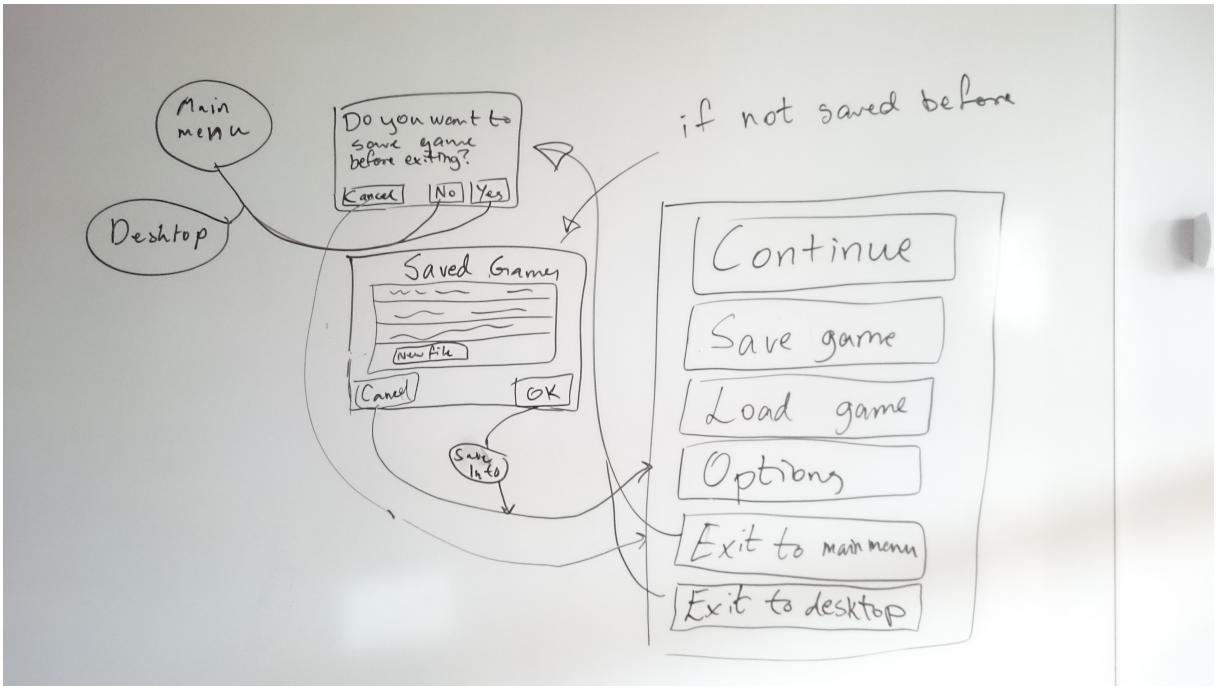

Figure 3. In-game menu.

Domain model

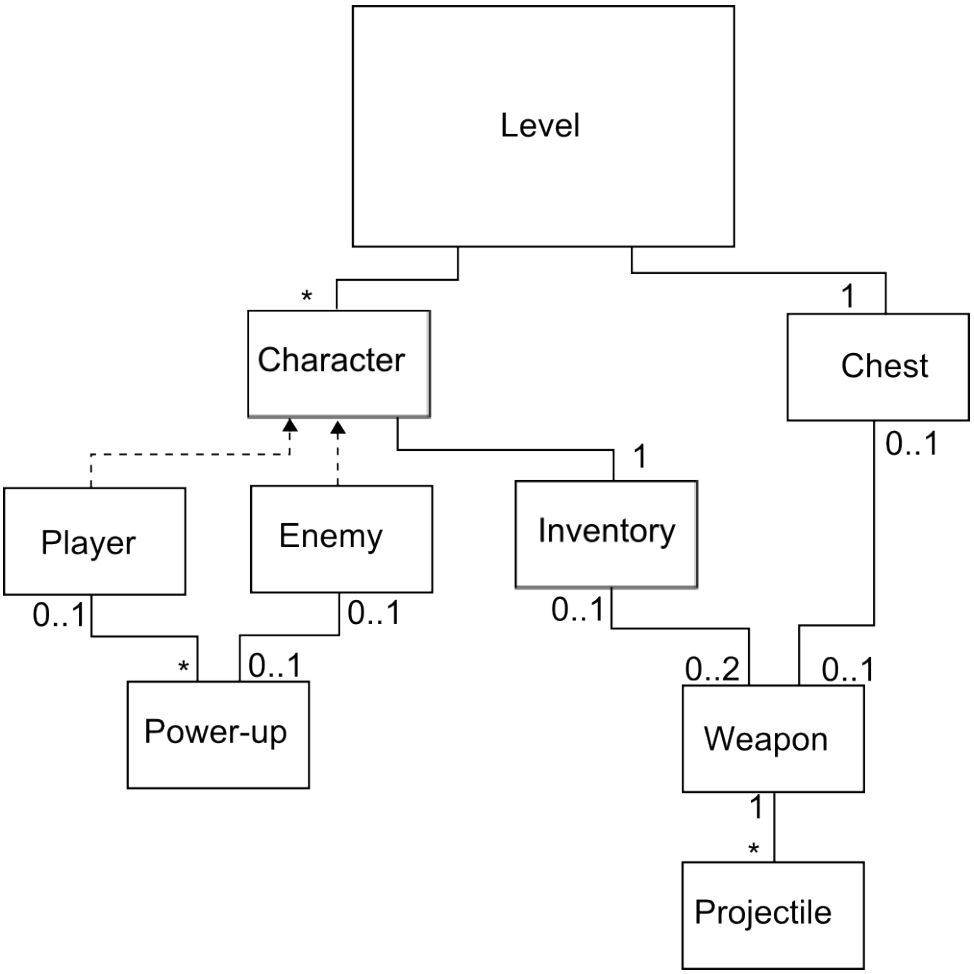

Figure 4. Domain model.

Sequence diagrams

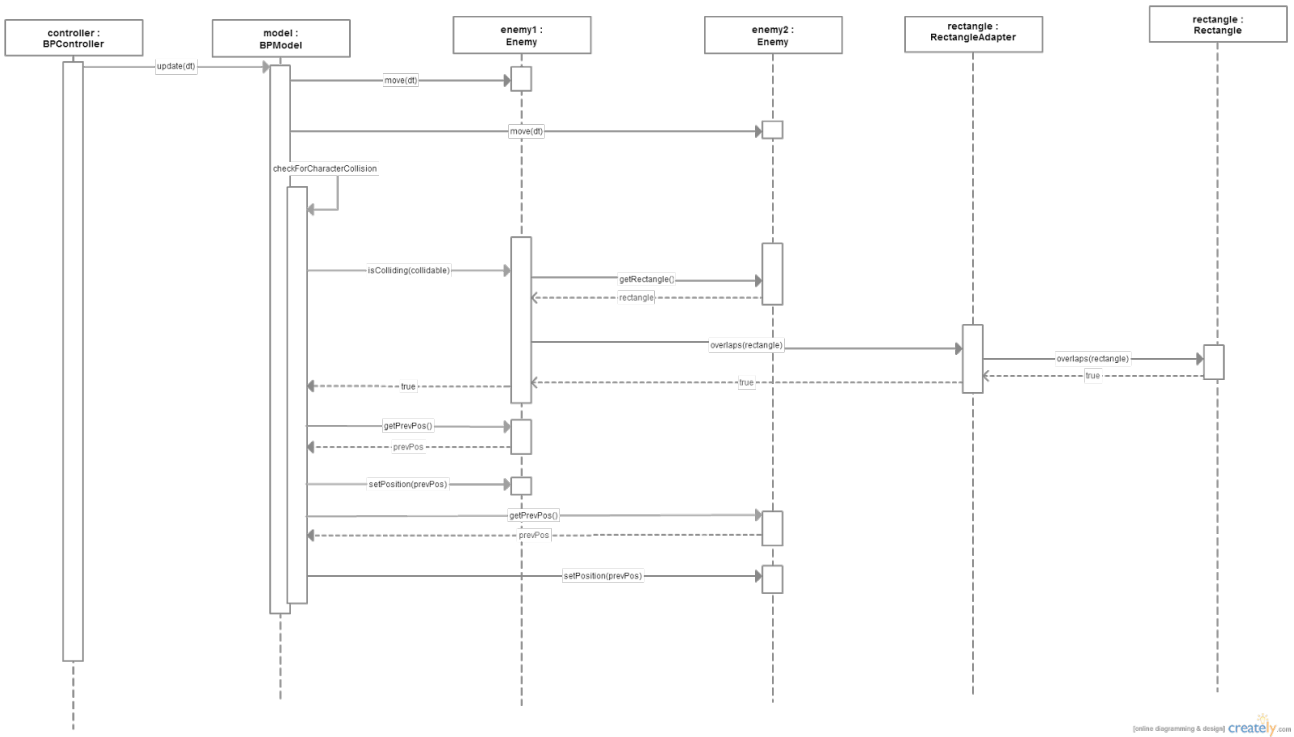

Figure 5. A sequence diagram which depicts the use case where two character collide (in this case enemies).

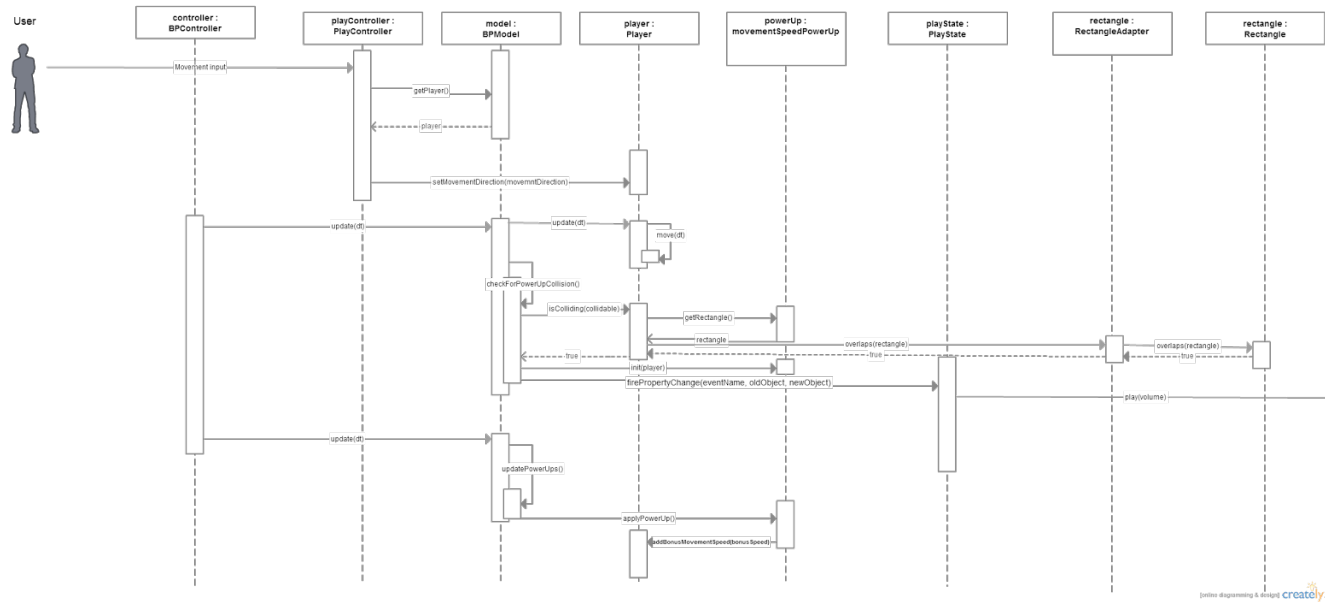

Figure 6. A sequence diagram depicting the use case where a player picks up a power-up.

Use case texts

Use case: Loot from chest
Summary: User moves towards a chest and loots it.
Priority: Mid
Participators: User and system

Normal flow

User	System
1. Presses movement-key	
	2. Character moves to chest
3. Presses interact-key	
	4. Character opens chest + sound effect
	5. Adds weapon to inventory
	6. Equips weapon

Alternate flow

User	System
	4.1. Chest is empty, nothing happens

Use case: Fire weapon
Summary: Fires a weapon towards an enemy.
Priority: High
Participators: User and system

Normal flow

User	System
1. Aim at enemy	
2. Press “shoot”-button	
	3. Check if weapon has ammo in magazine
	4. Launch projectile, lower ammo count in magazine, play an appropriate shooting sound
	5. Check if enemy is hit
	6. Decrease enemy HP, play a fitting hit sound

Alternate flow

User	System
	4.1. No ammo in magazine
	4.2. Reload the weapon
	4.3. Show a progress bar for the reload and play a reload sound

Use case: Pick up power-up.
Summary: PC picks up and activates power-up.
Priority: Mid
Participators: User and system

Normal flow

User	System
Press movement keys	
	2. Move PC over power-up
	3. Power-up activates, play fitting power-up sound

Alternate flow

	3.1. The PC already has a power-up of the same type active
	3.2. The time until the power-up runs out is reset
	3.3. Increase the potency of the power-up, play appropriate power-up sound

Use case: Collision between characters
Summary: A character walks into another character
Priority: Mid
Participators: User and system

Normal flow

User	System
1. Character walks in given direction	
	2. Characters collide
	3. Characters are placed at their previous positions

Use case: Projectile collision with character
Summary: Fired projectile hits character
Priority: High
Participators: User and system

Normal flow

User	System
1. Character launches a projectile	
	2. Moves projectile
	3. Projectile collides with another character
	4. Character that is hit takes damage
	5. Projectile is removed

Alternate flow

User	System
	3.1 Projectile collides with the character who launched it
	3.2 Projectile passes through character (without taking damage)