Requirements Analysis Document
Project group 5 - Zombieweek

Neda Farhand
Tobias Hallberg
Daniel Johansson
Erik Öhrn

2015-05-31

Contents

1 Introduction

1.1 Purpose of application

This project strives to create an easily played game in 2D that provides the touch and feel of a more advanced gaming experience by stripping down the number of available commands without limiting the player.

1.2 General characteristics of application

The application will be a single-player, non-networked game with a graphical user interface for the Windows/Linux/Mac operating systems.

The application will be based on the player's ability to move and progress through different levels, finishing the game once the last level and final antagonist has been beaten - eg. a multi level, single player game. The player has a specific amount of time to finish the entire game (eg. beat all levels) as made visible by a timer in the bottom right corner. If the game is cancelled the system will enable the player to commence from the last checkpoint. The GUI will feature a number moving characters such as the player, antagonists and projectiles thrown by player.

1.3 Scopes of application

The application does not contain a multi-player mode, and is thereby only able to provide a single-player experience. Neither does it offer an online mode, related to previously stated limitations that prove the necessity of an online mode to be lacking.

1.4 Objectives and success criteria of the project

1. It should be possible to complete a single player game by passing through every available level, which should be at least one level consisting of a minimum of three rooms.

2. The system should automatically save an ongoing game at checkpoints, thereby also registering how far into the game the player has progressed.

3. It should be possible for the player to choose a level they have already completed when returning to the game after having finished a previous gaming session.

4. It should be possible to fire projectiles, and thereby attacking the antagonists.

5. It should be possible to be attacked by an antagonist.

1.5 Definitions, acronyms and abbreviations

Box2D - a physics simulator.

Checkpoint - a point that, when "reached" in the game, calls to the system to register it as the most recent player location.

FPS - short for "frames per second". See frame rate.

Frame rate - how often the screen is updated. This is normally, for example, 24 FPS for movies, 30 FPS for console games and up to 60 FPS for computer games. A higher number yields smoother looking video or game play, although a number higher than the screen's update frequency can cause tearing.

GUI - Graphical User Interface

Gradle - an Enterprise build automation.

Jar - a Java executable file

Java - an object-oriented programming language

Level - a section of the game, often more difficult to finish that its predecessor but simpler than the one which follows.

LibGDX - a gaming engine.

Player - the actor.

Resources - items at the player's disposal such as ammunition (projectiles) and number of lives.

System - the application as rooted in the host computer.

Tiled - a tool used to create environments within the game using virtual "tiles".

(in alphabetical order)

2 Requirements

2.1 Functional requirements

The user should be able to:

- Start a game
- Start a new one
- Continue on a previous played game
- Select an already completed level to play
- Complete a level
- Move the player using the keyboard
- Aim using either keyboard or mouse
- Shoot by either using keyboard or mouse
- Defeat enemy using the weapon
- Pick up potion that gives the player different features
- See current player health and ammunition in a HUD
- Pick up ammunition that missed its target from the floor

2.2 Non-functional requirements

2.2.1 Usability

Zombieweek should be usable by anyone with a desktop or laptop that runs the latest version of either Windows, OS X or a recent Linux distribution running Java 8 or later. The game requires only a keyboard to function; using a mouse is optional.

2.2.2 Reliability

The finished product is stable, thereby no crashes should occur.

2.2.3 Performance

The game should be performance light, meaning that basically any desktop, laptop or ultrabook from the last three years should be able to run it without any frame rate drop or lag. We are aiming for a consistent frame rate of 60 frames per second.

2.2.4 Supportability

The game is provided "as-is" with no cost of purchase or ownership. Thus no support or warranty is given.

2.2.5 Implementation

2.2.6 Packaging and installation

The game will be packaged as a runnable jar file. No installation process is required.

2.2.7 Legal

The game and its files are the sole property of the group consisting of Erik Öhrn, Neda Farhand, Daniel Johansson and Tobias Hallberg. The group claims no ownership of Java, Libgdx, Box2d or any associated files.

2.3 Application models

See Appendix for UML diagram.

2.3.1 Use case model

The use cases included are:

- Move

    MoveUp

    MoveDown

    MoveLeft

    MoveRight

    MoveDiagonally

    MoveNorthWest

    MoveNorthEast

MoveSouthWest
MoveSouthEast

- NewGame
- ChooseLevel
- Exit
- Aim
- PickUpItem
  PickUpBook
  PickUpPotion
  PickUpHealthPotion
  PickUpSpeedPotion.

2.4 Use case priority

All use cases apart from PickUpPotion and its two subcases are of high priority. These are of medium priority, seeing as they are a feature and not a programme vitality.

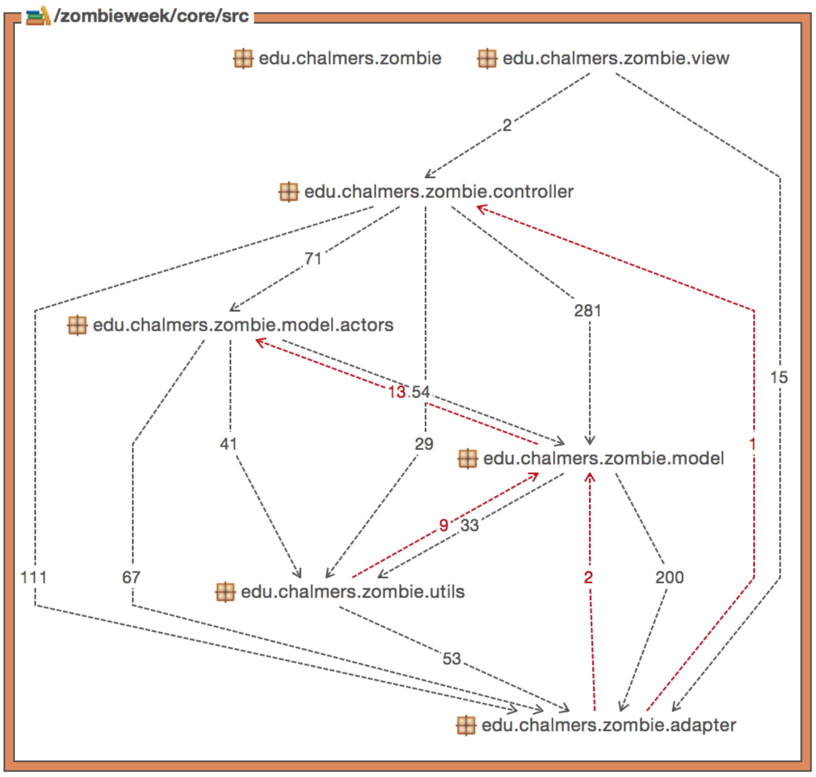
2.5 Analysis model

See Appendix

2.6 User interface

Bibliography

A Analysis model



/zombieweek/core/src
edu.chalmers.zombie
edu.chalmers.zombie.view
2
edu.chalmers.zombie.controller
71
281
15
edu.chalmers.zombie.model.actors
13
54
1
41
29
edu.chalmers.zombie.model
9
33
111
67
edu.chalmers.zombie.utils
2
200
53
edu.chalmers.zombie.adapter

Figure 1: An UML diagram over the project

B GUI

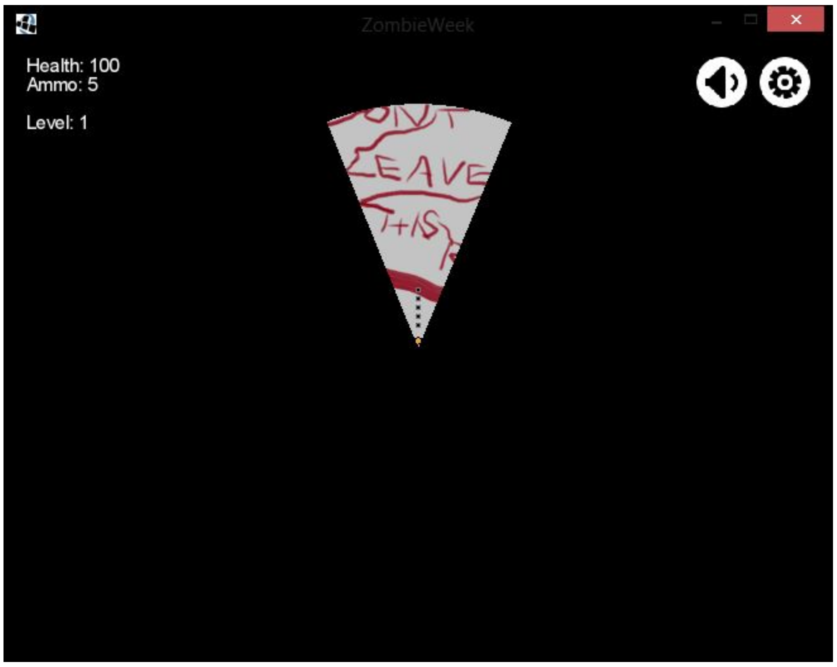


Figure 2: The GUI when "fear of the dark mode" is enabled

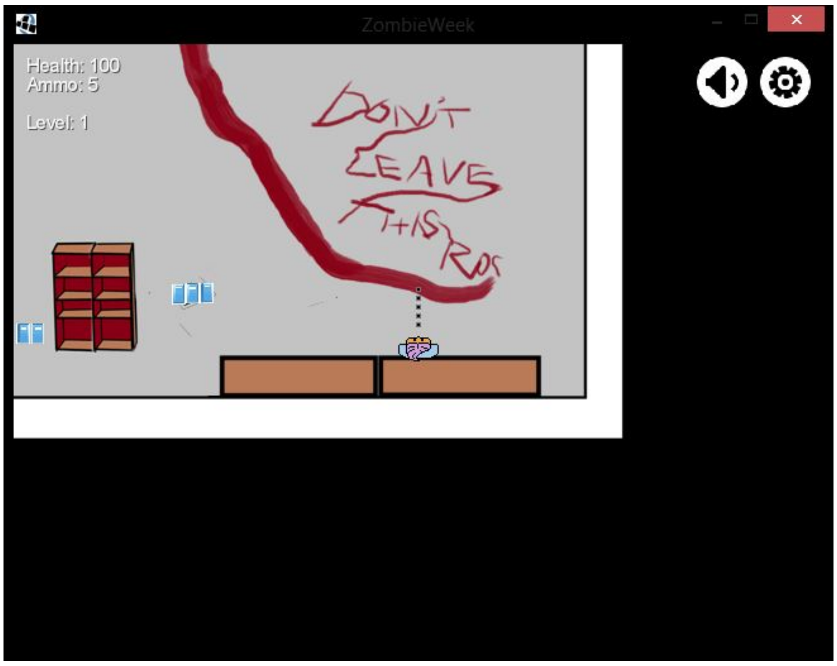


Figure 3: The GUI when "fear of the dark mode" is disabled

C Use cases

Use Case: Move

Summary: This is how the player moves their avatar around inside the grid. UC Attack is available while this UC is being performed and vice versa.

Priority: High

Extends: –

Includes: UC MoveUp, UC MoveDown, UC MoveRight, UC MoveLeft, UC MoveDiagonally

Participators: Actual player

Normal flow of events

A standard procedure Move.

	Actor	System
1	Presses one of the WASD keys on keyboard (see included UCs).	
2		Avatar moves in correct direction (see included UCs) on grid in current room.

Alternate flows

Flow 1.1: Player walks into wall

	Actor	System
1.1.1	Moves into a wall.	
		Avatar remains still facing wall until one of the other WASD keys is pressed.

Flow 1.2: Player collides with zombie

	Actor	System
1.2.1	Moves into a zombie.	
		Zombie attacks. The number of lives the player has is decreased by a certain amount depending on the amount of damage the zombie is capable of. Should the player's lives go below zero, the game is over.

Flow 1.3: Player walks through door

	Actor	System
1.3.1	Moves through door.	
1.3.2		New room on screen; avatar enters.

Flow 1.4: Multiple keys pressed at once

	Actor	System
1.4.1	Presses more than one of the WASD keys on keyboard (see included UCs) simultaneously.	
1.4.2		Avatar moves in direction given by the *most recently pressed key. Will move diagonally* (see UC MoveDiagonally) *should the combination be AW, AS, SD, or WD.*

Exceptional flow

Flow 1.5: Non-assigned keys pressed

	Actor	System
1.5.1	Presses keys without an assignment in an attempt to move.	
1.5.2		Avatar remains still.

Use Case: MoveUp

Summary: This is how the player moves their avatar up inside the grid, where 'up' is considered increasing the positional value along the y-axis on the two-dimensional playing field.

Priority: High

Extends: UC Move

Includes: –

Participators: Actual player

Normal flow of events

A standard procedure MoveUp.

	Actor	System
1	Presses "W" key.	
2		Avatar moves in the direction *up* on grid in current room.

Alternate flows

See UC Move (this UC included).

Exceptional flow

See UC Move (this UC included).

Use Case: MoveDown

Summary: This is how the player moves their avatar *down* inside the grid, where 'down' is considered decreasing the positional value along the y-axis on the two-dimensional playing field.

Priority: High

Extends: UC Move

Includes: –

Participators: Actual player

Normal flow of events

A standard procedure MoveDown.

	Actor	System
1	Presses "S" key.	
2		Avatar moves in the direction *down* on grid in current room.

Alternate flows

See UC Move (this UC included).

Exceptional flow

See UC Move (this UC included).

Use Case: MoveLeft

Summary: This is how the player moves their avatar *left* inside the grid, where 'left' is considered decreasing the positional value along the x-axis on the two-dimensional playing field.

Priority: High

Extends: UC Move

Includes: –

Participators: Actual player

Normal flow of events

A standard procedure MoveLeft.

	Actor	System
1	Presses "A" key.	
2		Avatar moves in the direction *left* on grid in current room.

Alternate flows

See UC Move (this UC included).

Exceptional flow

See UC Move (this UC included).

Use Case: MoveRight

Summary: This is how the player moves their avatar right inside the grid, where ‘right’ is considered increasing the positional value along the x-axis on the two-dimensional playing field.

Priority: High

Extends: UC Move

Includes: –

Participators: Actual player

Normal flow of events

A standard procedure MoveRight.

	Actor	System
1	Presses “D” key.	
2		Avatar moves in the direction *right* on grid in current room.

Alternate flows

See UC Move (this UC included).

Exceptional flow

See UC Move (this UC included).

Use Case: MoveDiagonally

Summary: This is how the player moves diagonally on the two-dimensional playing field.

Priority: High

Extends: UC Move

Includes: UC MoveNorthWest, UC MoveNorthEast, UC MoveSouthWest, UC MoveSouthEast

Participators: Actual player

Normal flow of events

A standard procedure MoveDiagonally.

	Actor	System
1	Presses a pair of the WASD keys apart from the combination W and S simultaneously.	
2		Moves avatar diagonally across screen.

Alternate flows

See UC Move (this UC included).

Exceptional flow

See UC Move (this UC included).

Use Case: MoveNorthWest

Summary: This is how the player moves in the direction *northwest* across the grid, where ‘northwest’ is considered increasing the positional value along the y-axis while decreasing it along the x-axis on the two-dimensional playing field.

Priority: High

Extends: UC MoveDiagonally

Includes: –

Participators: Actual player

Normal flow of events

A standard procedure MoveNorthWest.

	Actor	System
1	Presses ‘W’ and ‘A’ keys simultaneously.	
2		Avatar moves *northwest* across the grid.

Alternate flows

See UC MoveDiagonally (this UC included).

Exceptional flow

See UC MoveDiagonally (this UC included).

Use Case: MoveNorthEast

Summary: This is how the player moves in the direction *northeast* on the two-dimensional playing field, where 'northeast' is considered an increasing positional value along both the x- and y-axis.

Priority: High

Extends: UC MoveDiagonally

Includes: –

Participators: Actual player

Normal flow of events

A standard procedure MoveNorthEast.

	Actor	System
1	Presses 'W' and 'D' keys simultaneously.	
2		Moves avatar northeast across the grid.

Alternate flows

See UC MoveDiagonally (this UC included).

Exceptional flow

See UC MoveDiagonally (this UC included).

Use Case: MoveSouthWest

Summary: This is how the player moves in the direction *southwest* on the grid, where 'southwest' is considered decreasing the positional value along both the y- and x-axis on the two-dimensional playing field.

Priority: High

Extends: –

Includes: –

Participators: Actual player

Normal flow of events

A standard procedure Choose Level.

	Actor	System
1	Presses 'S' and 'A' keys simultaneously.	
2		Avatar moves *southwest* across the grid.

Alternate flows

See UC MoveDiagonally (this UC included).

Exceptional flow

See UC MoveDiagonally (this UC included).

Use Case: MoveSouthEast

Summary: This is how the player moves in the direction *southeast* on the grid, where 'southeast' is considered decreasing the positional value along the y-axis while increasing the positional value along the x-axis on the two-dimensional playing field.

Priority: High

Extends: UC MoveDiagonally

Includes: –

Participators: Actual player

Normal flow of events

A standard procedure MoveSouthEast.

	Actor	System
1	Presses 'S' and 'D' keys simultaneously.	
2		Avatar moves *southeast* across the grid.

Alternate flows

See UC MoveDiagonally (this UC included).

Exceptional flow

See UC MoveDiagonally (this UC included).

Use Case: NewGame

Summary: This is how the player initiates a new gaming session, starting at the first level.

Priority: High

Extends: –

Includes: –

Participators: Actual player

Normal flow of events

A standard procedure New Game.

	Actor	System
1	Starts program (e.g. executes .jar file).	
2		Shows selection screen/welcome screen.
3	Clicks "New Game" button on screen.	
4		Shows character selection screen.
5	Chooses character to play as.	
6		Initiates new game.

Alternate flows

There are no alternate flows.

Exceptional flow

Screen is closed: Exit on close.

Use Case: ChooseLevel

Summary: This is how the player initiates a gaming session by choosing one of the unlocked levels. The system remembers which levels have been unlocked.

Priority: High

Extends: –

Includes: –

Participators: Actual player

Normal flow of events

A standard procedure ChooseLevel.

	Actor	System
1	Clicks “Choose Level” button on main-menu screen.	
2		Loads available levels as clickable buttons, remaining levels as non-clickable (grey).
3	Clicks on one of the available levels.	
4		Initiates game starting at chosen level.

Alternate flows

There are no alternate flows.

Exceptional flow

Screen is closed: Exit on close.

Use Case: Exit

Summary: This is how the player exits the game.

Priority: High

Extends: –

Includes: –

Participators: Actual player

Normal flow of events

A standard procedure Exit.

	Actor	System
1	Presses ‘ESC’ button.	
2		Pauses game; shows pause menu.
3	Clicks ’Quit’ button.	
4		System exit.

Alternate flows

Flow 1.1: Actor exits through main menu

	Actor	System
1	Presses ‘Quit’ button.	
2		System exit.

Flow 1.2: Actor exits through Game Over screen

	Actor	System
1	Presses ‘Quit’ button.	
2		System exit.

Exceptional flow

Screen is closed: Exit on close.

Use Case: Aim

Summary: This is how the player aims, e.g. controls in which direction the projectiles will be thrown. It is possible to switch between key-aiming and mouse-aiming.

Priority: High

Extends: –

Includes: –

Participators: Actual player

Normal flow of events

A standard procedure Aim.

	Actor	System
1	Presses 'up' OR 'down' arrow key.	
2		Arrow changes direction turning clockwise OR counterclockwise around the avatar.

Alternate flows

Flow 1.3: Non-assigned keys pressed

	Actor	System
1	Presses 'C' key.	
2		Changes from key-aiming to mouse-aiming.
3	Moves mouse.	
4		Arrow changes direction by following cursor.

Exceptional flow

Flow 1.3: Non-assigned keys pressed

	Actor	System
1.5.1	Presses keys without an assignment in an attempt to aim.	
1.5.2		Arrow remains still.

Use Case: PickUpItem

Summary: This is how the player picks up an item from the ground.

Priority: High

Extends: –

Includes: UC PickUpBook, UC PickUpPotion

Participators: Actual player

Normal flow of events

A standard procedure PickUpItem.

	Actor	System
1	Moves toward object, moving over it.	
2		Item is registered as “picked up” and will be removed from the playing field followed by being added to the player’s properties.

Alternate flows

There are no alternate flows.

Exceptional flow

Flow 1.2: Item fails to be picked up

	Actor	System
1.5.1	Moves towards object, but not properly over.	
1.5.2		Item remains on playing field until being successfully picked up.

Use Case: PickUpBook

Summary: This is how the player picks up a book from the ground; whether it is one the player themselves have already thrown/fired or one placed on the ground beforehand, and what happens when they do.

Priority: High

Extends: UC PickUpItem

Includes: –

Participators: Actual player

Normal flow of events

A standard procedure PickUpBook.

	Actor	System
1	Picks up the item (book).	
2		Book is registered as picked up. Book is removed from playing field. Ammunition count is increased by one.

Alternate flows

See UC PickUpItem (this UC included).

Exceptional flow

See UC PickUpItem (this UC included).

Use Case: PickUpPotion

Summary: This is how the player picks up a potion from the ground.

Priority: Medium

Extends: UC PickUpItem

Includes: UC PickUpSpeedPotion, UC PickUpHealthPotion

Participators: Actual player

Normal flow of events

A standard procedure PickUpPotion.

	Actor	System
1	Picks up the item (potion).	
2		Potion is registered as picked up. Potion is removed from playing field. Player is affected by property changes.

Alternate flows

See UC PickUpItem (this UC included).

Exceptional flow

See UC PickUpItem (this UC included).

Use Case: PickUpHealthPotion

Summary: This is how the player picks up a health potion from the ground, and what happens when they do.

Priority: Medium

Extends: UC PickUpPotion

Includes: –

Participators: Actual player

Normal flow of events

A standard procedure PickUpSpeedPotion.

	Actor	System
1	Picks up the potion (health potion).	
2		The player's amount of lives is refilled by a count of 5, OR until it reaches 10 (full).

Alternate flows

See UC PickUpPotion (this UC included).

Exceptional flow

See UC PickUpPotion (this UC included).

Use Case: PickUpSpeedPotion

Summary: This is how the player picks up a speed potion from the ground, and what happens when they do.

Priority: Medium

Extends: UC PickUpPotion

Includes: –

Participators: Actual player

Normal flow of events

A standard procedure PickUpSpeedPotion.

	Actor	System
1	Picks up the potion (speed potion)	
2		The player's speed triples for 5 seconds.

Alternate flows

See UC PickUpPotion (this UC included).

Exceptional flow

See UC PickUpPotion (this UC included).

Use Case: TurnOnAndOffFlashLight

Summary: This is how the player operates the flashlight function in-game.

Priority: Medium

Extends: –

Includes: –

Participators: Actual player

Normal flow of events

A standard procedure TurnOnAndOffFlashLight.

	Actor	System
1	Chooses “fear of the dark”-mode when starting a new game.	
2		Loads a dark room.
3	Presses ’F’ key.	
4		Flashlight turns on.

Alternate flows

Flow 1.1: Player presses ‘F’ key multiple times

	Actor	System
1.1.1	Presses ‘F’ key.	
1.1.2		Flashlight turns on.
1.1.3	Presses ’F’ key.	
1.1.4		Flashlight turns off.

Exceptional flow

Flow 1.2: Player presses ‘F’ key while in lights-on mode

	Actor	System
1.2.1	Presses ‘F’ key.	
1.2.2		Nothing happens.