RAD

Version: 0.1

Date: 25/3

Author: grupp 16

This version overrides all previous versions.

1 Introduction

1.1 Purpose of application

The purpose of the project is to create an entertaining platform game wherein the player controls a character, running and jumping to move from one point to the next.

1.2 General characteristics of application

The application will be a desktop, stand alone (non-networked), single-player application with a graphical user interface for the Windows/Mac/Linux platforms. The game flow provides continuous feedback. The character of the game will carry a flashlight which will provide almost all of the light in the game, therefore the game will have a dark background and the character can barely be seen.

The game will provide a certain amount of levels which will vary in difficulty. To get to the next level, the gamer must complete a number of puzzles, for example collect keys or other objects to unlock doors and so on. There will be several obstacles in the different levels, like enemies who will try to kill the character and chasms the character can fall into. The enemies will only be able to approach the character when light falls on them, so the gamer can move without fright of them when not pointing the flashlight at them.

1.3 Scope of application

One player game. The player can save their progress. Four save-slots. Will have sound. Multiple light sources, with light sources as part of levels.

1.4 Objectives and success criteria of the project

There will be at least two levels with complete functionality. At least a single light source.

1.5 Definitions, acronyms and abbreviations

Active object - Active or interactive objects which exist within the game but are impossible to pick up, such as light switches or movable crates.
FPS - frames per second
JRE - the Java Run time Environment. Additional software needed to run an Java application.
GUI - Graphical User Interface

Viewport - Screen space in which the game world is drawn, as opposed to interface controls.

2 Requirements

2.1 Functional requirements

The player should be able to:

1. Start a game.
2. Create a save file.
3. Move the character around (includes jumping and crawling).
4. Move the flashlight around.
5. Pick up certain items from the world.
6. Move certain items around in the world.
7. Go through doors to access new levels.
8. Save their progress to the save file.
9. Load a active save file.
10. Exit the application.

2.2 Non-functional requirements

2.2.1 Usability

“Learning by doing”-approach, some kind of instruction of the controls and some in-game-tips of how to play.

2.2.2 Reliability

N/A

2.2.3 Performance

The game should be efficient enough to reliably generate at least 30 FPS on modern desktop or laptop computer systems.

2.2.4 Supportability

The application should be build in such a way that porting it to another device should not be more work than approximately 1 man-month. As such it should be built following the principle that the model of the game should be detached from it’s view and input method.

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

JAR-file

2.2.7 Legal

NA

2.3 Application models

2.3.1 Use case model

UML and a list of UC names (text for all in appendix)

2.3.2 Use cases priority

1. Walk
2. Jump
3. Move flashlight
4. Enemy moving
5. Interact with interactable objects
6. Crouch
7. Crawl
8. Go through door
9. Pick up item
10. Drop item
11. Use item
12. Pause the game
13. Change control-buttons
14. Saving a game-session
15. Loading a game-session

2.3.3 Domain model

See appendix.

2.3.4 User interface

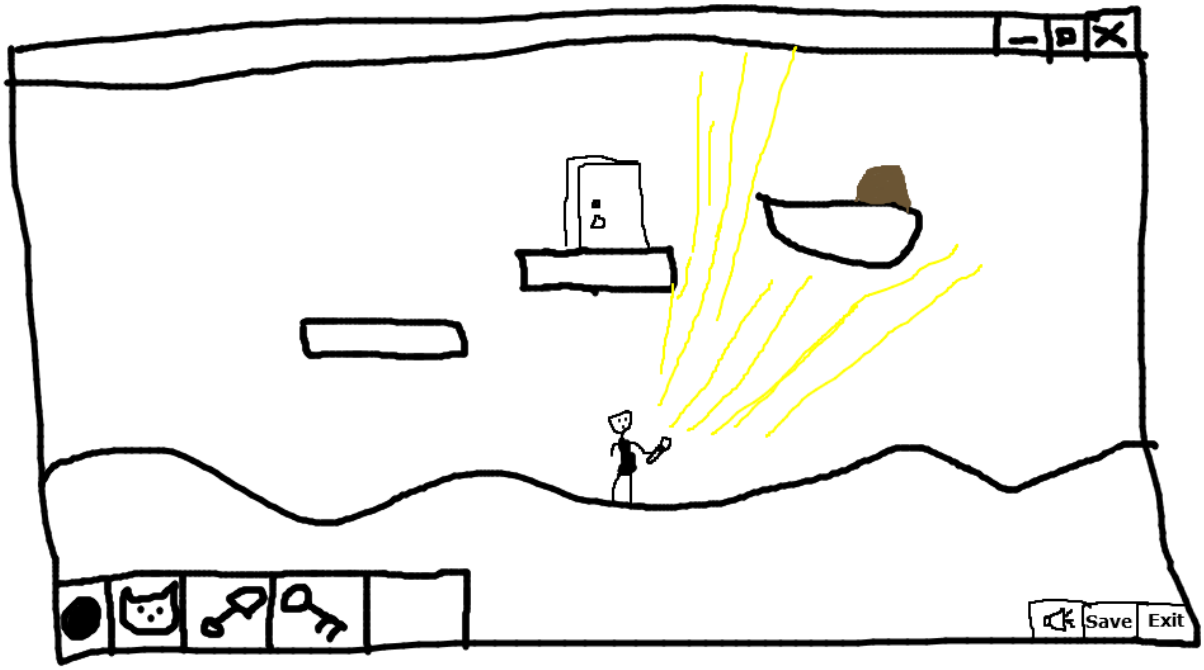
The game will use a simple interface, with most of the screen space consisting of the viewport. Interface elements include an inventory, in which the items the player is currently carrying are shown, and meta controls such as saving, exiting, and controlling sound volume.

2.4 References

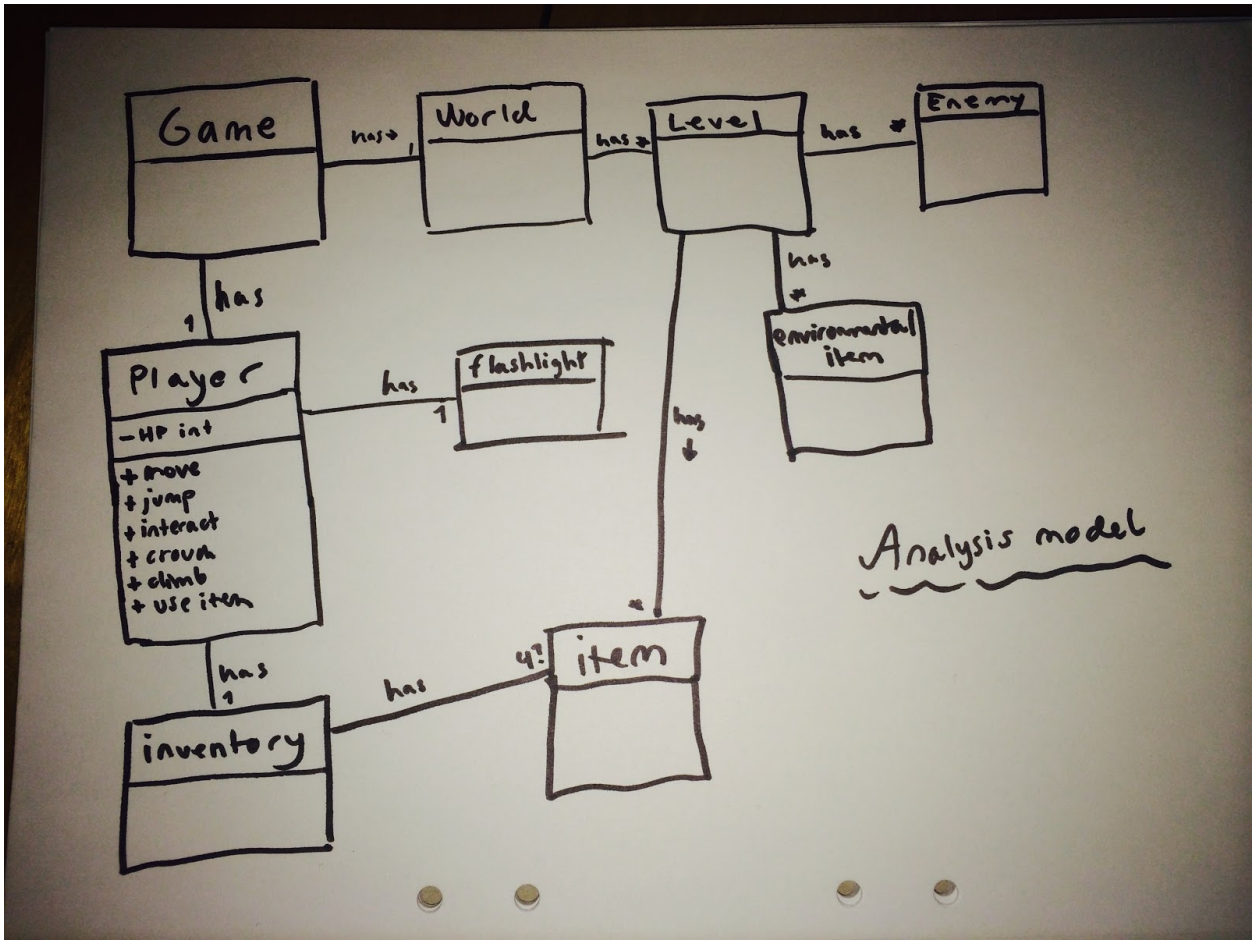
APPENDIX
GUI
Domain model
Use case tex

APPENDICES

Appendix A



Superduperpreliminärt skissförslag



first version of analysis model

Appendix B

Use Cases:

Use Case: Walk

Summary: This is how the player moves in the game area. Walking left and right is equivalent but the player presses the different buttons to activate it.

Priority: high

Participators: Player

Normal flow of events:

	Actor	System
1	Player presses and holds a “walk” button	
		Character moves in the direction given.

Alternate flow

Flow 2.1 (Walks against a wall)

	Actor	System
1	Player presses and hold a “walk” button	
		Keeps the character still and displays it pushing against the wall.

Flow 2.2 (Walks against a enviromental object)

	Actor	System
1	Player presses and hold a “walk” button	
		Moves the character and the enviromental object in the given direction.

Use Case: Move flashlight

Summary: This is how the player sees things clearly in the dark setting, by moving around their flashlight.

Priority: high

Participators: Player

Normal flow of events:

	Actor	System
1	Moves cursor	
		illuminates the new area, where the cursor is pointing.

Use Case: Jump

Summary: The ability for the player to make the character jump.

Priority: high

Participators: Player

Normal flow of events:

	Actor	System
1	Presses the jump button	
		Launches the character upwards
		The character constantly accelerates downwards.
		When the character hits a platform the acceleration downward stops.

Alternative flow:

	Actor	System
1	Presses the jump button	
		Launches the character upwards
		The character constantly accelerates downwards.
		The character hits spikes on the way down and the player is killed. The text “Game Over” is displayed.

Use Case: Pick up item

Summary: This is how the Player picks up items in the world.

Priority: Low

Participators: Player

Normal flow of events:

	Actor	System
1	Presses the “interact”-button when the character is ontop of a item	
		Adds the item to the character’s inventory

Exceptional flow:

	Actor	System
1	Presses the “interact”-button when the character is ontop of a item	
		Displays “the inventory is full” on screen.

Use Case: Crouch

Summary: Lets the player make the character crouch to avoid obstacles.

Priority: medium

Participators: Player

Normal flow of events:

	Actor	System
1	Presses the crouch button	
		Displays the character as crouched

Use Case: Crawl

Summary: Lets the player make the character crawl to avoid obstacles and get through tight spaces.

Priority: Medium

Participators: Player

Normal flow of events:

	Actor	System
1	Holds down the crouch button while pressing one of the movement buttons	
		Displays the character crawling along the floor in the given direction

Alternate flow:
Flow 2.1 (Crawling against a wall):

	Actor	System
1	Holds down the crouch button while pressing one of the movement buttons.	
		Keeps the character still and facing the wall.

Flow 2.2 (Crawling against a enviromental object):

	Actor	System
1	Holds down the crouch button while pressing one of the movement buttons.	
		Keeps the character and the enviromental object still. Character facing the object.

Use Case: Enemy moving

Summary: Describes the movement of enemies in the game. Very basic but further alternate flows can be added when more enemy behaviors are added.

Priority: medium

Participators: Player, Enemy

Normal flow of events:

	Actor	System
1	Moves the flashlight over a enemy.	
		Makes the enemy move towards the player

Use Case: Use item

Summary: Lets the player use a picked up item at the appropiate place.

Priority: low

Participators: Player

Normal flow of events:

	Actor	System
1	Presses the “use Item” button.	
		If able to use the item at that

		position, uses the item. Otherwise displays “Item not usable here”

Alternate flow:

	Actor	System
1	Presses the “use Item” button.	
		Displays “Item not usable here”

Use Case: Drops item

Summary: Lets the player drop a item so that he can pick up another one

Priority: low

Participators: Player

Normal flow of events:

	Actor	System
1	Presses the “drop item” button.	
		Drops the active item on the ground besides the player.

Use Case: Go through door

Summary: Lets the player go from the world into a level

Priority: low

Participators: Player

Normal flow of events:

	Actor	System

1	Presses the “interact” button in front of a door.	
		Shifts the view to the level corresponding to the door.

Use Case: Pause the game

Summary: Lets the player pause the game and access the Pause menu, including the Options menu.

Priority: low

Participators: Player

Normal flow of events:

	Actor	System
1	Presses the “pause” button.	
		Pauses the game and displays the pause menu.

Use Case: Change control-buttons

Summary: Lets the player change the buttons for walking, crouching, crawling and interacting with items.

Priority: low

Participators: Player

Includes: Pause the game

Normal flow of events:

	Actor	System
1	Pauses the game and clicks on the “Controls” button.	

		Displays a menu describing which button corresponds to which action.
2	Clicks on the name of the action that should be changed.	
		Displays a pop-up telling the player to press the new button to be used for this action.
3	Clicks the new button to correspond to the action	
		Displays the menu now displaying the new button corresponding to that action.

Use Case: Interacting with interactable objects

Summary: Lets the player interact with interactable objects in the world. For example a light switch or maybe a elevator button.

Priority: medium

Participators: Player

Normal flow of events:

	Actor	System
1	Presses “Interact” button while standing by a interactable object.	
		Performs the action the object controls.

Alternate flow:

	Actor	System
1	Presses “Interact” button while not standing by a interactable object.	

		Displays “Nothing interactable” on the screen.

Use Case: Saving a game-session

Summary: Lets the player save the current game-session to return to later.

Priority: low

Participators: Player

Includes: Pause the game

Normal flow of events:

	Actor	System
1	Pauses the game and clicks on the “Save game”-button	
		Displays a list of 3 save files.
2	Clicks on a save file which you want to save in.	
		Asks the player for a name for the save file in a textfield.
3	Writes a name for the save file.	
		Saves the current game-session at the chosen save file with the given name.

Alternate flow:
Flow 2.1 (Save file already filled):

	Actor	System
1	Pauses the game and clicks on the “Save game”-button	
		Displays a list of 3 save files.

2	Clicks on a save file which you want to save in.	
		Ask the player if it wishes to overwrite the game-session already saved in the file.
3	Answers yes.	
		Asks the player for a name for the save file in a textfield.
4	Writes a name for the save file.	
		Saves the current game-session at the chosen save file with the given name.

Use Case: Load a game-session

Summary: Lets the player load a previous game-session.

Priority: low

Participators: Player

Includes: Pause the game

Normal flow of events:

	Actor	System
1	Pauses the game and clicks on the “Load game”-button	
		Displays a list of up to 3 named save files.
2	Clicks on a save file to be loaded.	
		Loads the chosen save file.