Requirements and Analysis Document for the RTS survival project.

Contents

Version: 1.0

Date: 2012-05-21

Author: Filip Brynfors, Markus Ekström, Björn Persson Mattsson, Jakob Svensson

1 Introduction

This section gives a brief overview of the project.

1.1 Purpose of application

The project aims to create a real-time-strategy survival game. The goal of the game is to build a base and defend it for as long as possible against incoming waves of enemies.

1.2 General characteristics of application

The application will be a desktop, standalone (non-networked), single-player application with a graphical user interface for the Windows/Mac/Linux platforms.

The application is a real-time-strategy game where the user can select and interact with units and buildings. Examples of interaction would be moving, building and attacking. The game ends when the player's base is overrun by enemies and is destroyed.

1.3 Scope of application

The application includes a simple AI that controls the attacking enemies. The application does not support saving games.

1.4 Objectives and success criteria of the project

It should be possible to select one unit or building at a time, construct buildings, train and move units, gather resources and attack incoming enemies. The application should use simple OpenGL graphics.

1.5 Definitions, acronyms and abbreviations

JRE, the Java Runtime Environment. Additional software needed to run a Java application.

2 Requirements

In this section we specify all requirements

2.1 Functional requirements

See also 2.4.2. The player should be able to:

- Start a new game. During the game the player should be able to:
    - Select a unit or building.
    - Move a selected unit.
    - Order a worker unit to gather resources.
    - Order a unit to attack an enemy.
    - Order a worker unit to construct a building.
    - Order a building to train a unit.
- Choose a difficulty setting.
- Exit the application.

2.2 Non-functional requirements

2.2.1 Usability

Usability is of some importance. Anyone familiar to RTS's should be able to figure out what to do. There should also be a README file if anyone needs help.

2.2.2 Reliability

NA

2.2.3 Performance

The game should update at least 30 times per second on a decent computer.

2.2.4 Supportability

The application should be usable on Windows/Mac/Linux.

2.2.5 Implementation

To achieve platform independence the application will use the Java environment. All hosts must have the JRE installed and configured.

2.2.6 Packaging and installation

The application is packaged as a jar file. The jar file is startable in windows with the StartGame.bat file. To run the jar file in linux, you simply use the command "java -jar FinalDawn.jar". If the jar file by any reason wont work, it's possible to open up the project in eclipse and run it from eclipse, alternatively create a new jar file.

2.2.7 Legal

The images taken from Starcraft I and Warcraft II may be a legal issue.

2.3 Application models

2.3.1 Use case model

See APPENDIX for UML diagram and textual descriptions.

2.3.2 Use cases priority

A list

High priority

- Select
- MoveUnit
- GatherResources
- UseAbility
- TrainUnit
- Attack
- ConstructBuilding

Low priority

- ExitGame
- Select Difficulty
- Restart Game

2.3.3 Domain model

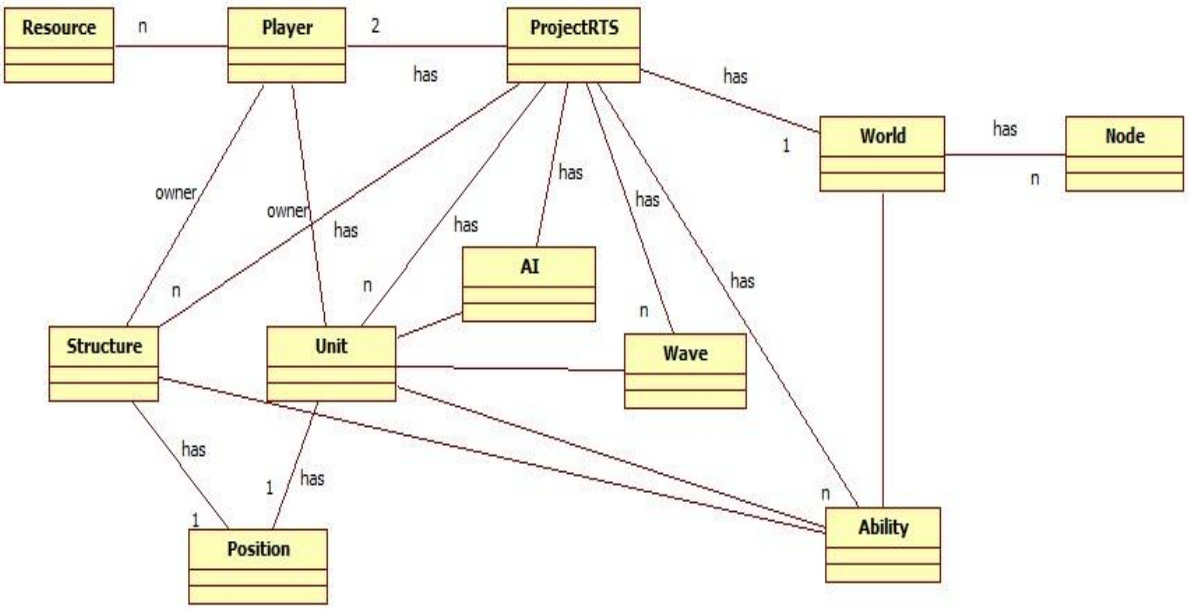
See APPENDIX for diagram.

2.3.4 User interface

The application will use a fixed (non-skinnable, non-themable) GUI following standard conventions.

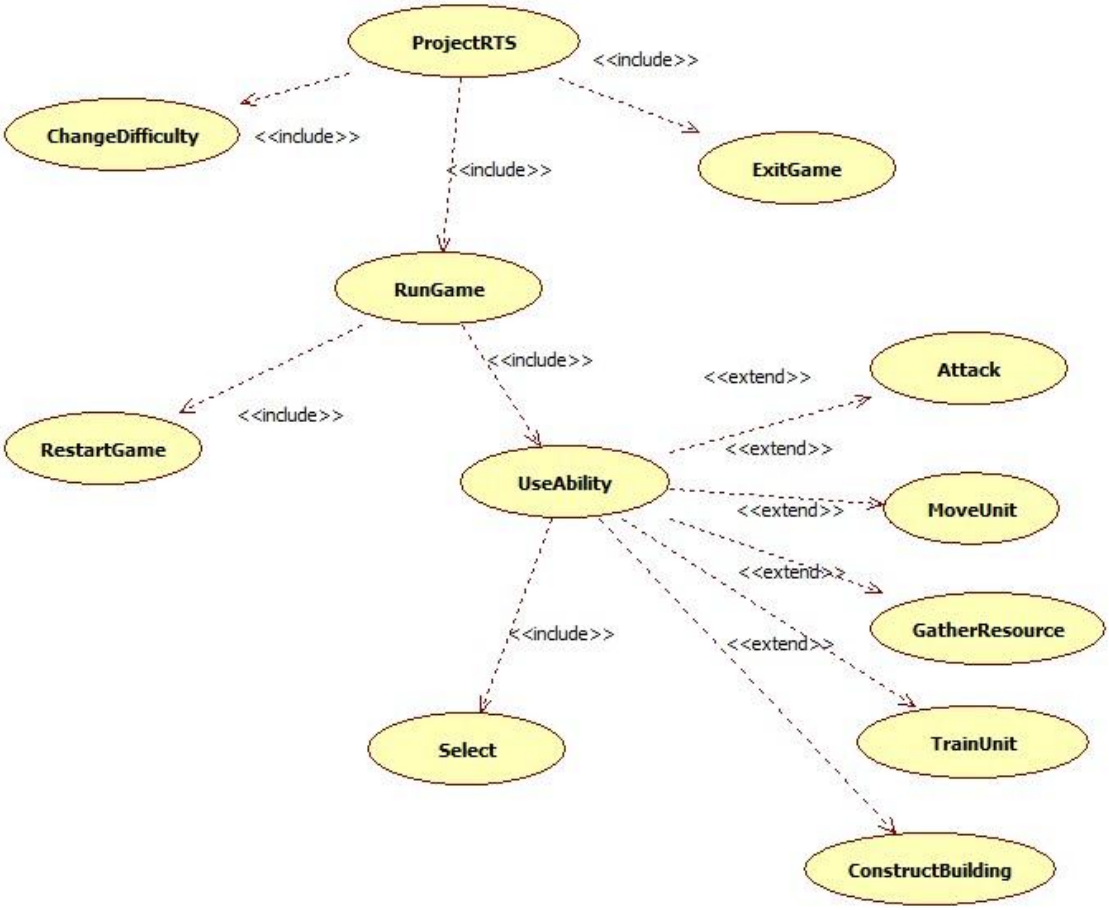
Appendix

Domain Model



GUI

Use Case Model



Use case texts

Use case: *Select*

Short description: How a user selects a unit/building.
Priority (high, mid, low): High

Participating actors

- Player

Normal flow of events

Player action	**Application response**
Clicks with mouse on unit/building	Circle around the selected unit/building and health bar and information is visible.

Exceptional flow:

No exception

Use case: *Attack*

Short description: How a user orders a unit/building to attack something.
Priority (high, mid, low): High

Extends or Includes: Includes Select.

Participating actors

- Player

Normal flow of events

Player action	**Application response**
Select a unit/building.	See use case: Select
Right-click on enemy that's within range.	Unit/building attacks the enemy.

Alternate flow: If unit cannot attack.

Move to the enemy.

Alternate flow: If building cannot attack.

Do nothing.

Alternate flow: If unit is not within range.

Try to move within range, then attack.

Alternate flow: If building is not within range.

Do nothing.

Exceptional flow:

No exception

Use case: *GatherResources*

Short description: How a user orders a unit to gather resources.
Priority (high, mid, low): Mid

Extends or Includes: Includes Select.

Participating actors

- Player

Normal flow of events

Player actions	**Application response**
Select a unit that can gather resources.	See use case: Select
Right-click on resource.	Unit moves to the resource and starts harvesting. When the unit has enough resources, it moves to a resource deposit and leaves the resource. The unit then returns to the resource and starts harvesting again (looping this sequence).

No alternate flow.

Exceptional flow:

No exception

Use case: *Move*

Short description: How a user moves a unit
Priority (high, mid, low): High

Extends or Includes: Includes Select.

Participating actors

- Player

Normal flow of events

Player action	**Application response**
Select a unit.	See use case: Select
Right-click on empty ground.	Unit moves to the targeted position.

Alternate flow: If clicked position is impossible to move to.

Move as close to the position as possible and stop.

Alternate flow: If clicked position is on an enemy.

See use case: Attack.

Alternate flow: If clicked position is on a resource and the unit is able to fetch resources.

See use case: GatherResources.

Exceptional flow:

No exception