Requirements Elicitation

1) Asteroids Problem Statement

a) Functional Requirements

Find all functional requirements and actors in the Asteroids Problem Statement below and represent them as a use case model.

b) Non-functional Requirements

Write down all non-functional and pseudo requirements in the Asteroids Problem Statement, categorized into Usability, Reliability, Performance, Supportability, Implementation, Interface, Operation, Packaging and Legal.

c) Mapping

For each non-functional requirement, find all relevant use cases from the problem statement.

This is the Asteroids Problem Statement as provided by the client:

Asteroids Problem Statement

1. Problem

The computer market for space games is moving towards complex, photo-realistic 3D simulation games. The problems are long startup times and high learning curves for the player. Furthermore these games require a large amount of CPU time, which forces the players to upgrade their computer system to the hardware needs for these games.
Solitaire, Minesweeper, or Moorhuhnjagd are examples for the success of simple and easy to use games. They have a high fun factor, no learning curve and short startup times. As the duration of a game is short, they can be played in nearly every situation.
The goal of Asteroids is to provide a simple, fast and cheap space game that has a high fun factor. Players have to survive in a shuttle in a region in outer space full of asteroids. It should not require any special hardware.

2. Objectives

The objectives of the Asteroids game are to:

- provide a simple, fast and interactive space shuttle game with a high fun factor.
- be platform independent.
- require no extra administrative knowledge. The player should be able to do all administrative work (e.g. installing).

3. Scenarios

This section presents typical scenarios of system usage for illustration.

3.1. Actor Instances

Actor Instance: Eve
Eve is a video game player with no special computer knowledge. She likes space shuttle games and plays asteroids.

3.2. Scenarios

Scenario: Configuring Asteroids
Eve can configure the frame rate, the number of the initial asteroids, and the images used for the illustration of asteroids and the space shuttle. When configuration is finished, the game board reflects all changes immediately.

Scenario: Losing an Asteroids game
If any asteroid collides with the space shuttle, a 'You lost!' dialog window appears.

Scenario: Playing an Asteroids game
Eve initiates the game. All asteroids start to move in different directions with different speeds. The space shuttle, navigated by Eve, is located at the center and waits for instructions from Eve. An instrument panel displays the shuttle's current speed and position. Eve can control the shuttle's speed and direction. Shuttle and asteroids change their direction when they hit a game board boundary. The angle of incidence is the angle of reflection.

Scenario: Starting the Asteroids application
The Asteroids user interface displays a game board, which consists of a space shuttle and an arbitrary number of asteroids, a tool bar with game control buttons and an instrument panel. There are two different kinds of asteroids: small ones and big ones. Small asteroids move faster than big ones. All asteroids are computer controlled. The space shuttle starts in the middle of the game board.

Scenario: Winning an Asteroids game
Eve can fire rockets. When a rocket collides with a small asteroid, the asteroid explodes and disappears from the game board. When a rocket collides with a big asteroid, it explodes and many small asteroids appear instead of the big one. Eve wins the game when all asteroids are destroyed.

4. Requirements

The Asteroids player should be able to:

- download and install the Asteroids game.
- run the game on multiple platforms.
- start the Asteroids application.
- stop the Asteroids application.
- start a game.
- learn how to play while playing the game, i.e. Asteroids must be simple to play: The user interface and the rules of playing Asteroids must be intuitive.
- play in real time. The player should see the result of a mouse click immediately. The game must provide short response time. Even the startup time of the game must be low.
- fire rockets.
- observe the smooth movements of asteroids and the space shuttle.
- collide the user-navigated space shuttle with an asteroid.
- see all the relevant information of the space shuttle during the asteroids game. This includes all properties needed for navigating the shuttle (shuttle speed, position etc.).
- stop a game.
- change the direction of the navigated space shuttle.
- change the speed of the navigated space shuttle.
- configure the number of asteroids of each kind. There are two different kinds of asteroids. One is small and fast and does not change its direction until it reaches the boundary of the game board. The other is big and slow, and it may change its direction any time. Both kinds of asteroids should be able to change their speed. They have a different minimum and maximum speed.
- customize the game. The game must be adaptable to both North American and Germany markets, i.e. it must support different metric systems, languages and imperial measurement systems.

5. Target environment

The Asteroids application must be built in Java using the Eclipse IDE.
The game must run at least on MS Windows (2000 and above), Ubuntu Linux (7.04) and Mac OS X (10.4.1 and above).

2) Joe's Online Pizza Problem Statement

a) Functional Requirements

Find all functional requirements and actors in the Pizza Ordering System problem statement above and represent them as a use case diagram. You don't need to write textual descriptions.

b) Non-functional Requirements

Write down all non-functional and pseudo requirements in the Pizza Ordering System problem statement, categorized into Usability, Reliability, Performance, Supportability, Implementation, Interface, Operation, Packaging and Legal.

Joe's Online Pizza Problem Statement

Joe's Pizza Delivery wants to speed up the ordering process, reduce losses caused by misunderstandings on the phone and attract new customers. A new web-based pizza ordering system that allows customers to enter orders in their web browsers is supposed to solve all three issues. The system must be built for the WebObjects platform using the Xcode IDE and integrate in an existing Apache environment.

The ordering system must be easy to use, as customers of all ages and expertise levels are supposed to use it. Customers may order pizzas with three different types of dough, thick or thin, and various toppings. Customers must be able to register for a customer account. A customer account stores address information and preferences, but no payment details for security reasons. Orders should be possible with or without a customer account. For privacy reasons, customer data must be stored in encrypted form only. The system must be usable with all major web browsers (i.e. Internet Explorer, Firefox, Safari and Opera) and be able to handle at least 10 customers ordering at the same time.

The cook can request a list of all open orders. When he has finished making a pizza, he marks an order as "ready for delivery". A delivery note with the customer's address, to be attached to the pizza by the cook, is printed automatically.