Requirements

1. Introduction

1.1 Purpose

The product brief was the first read for the team. The initial scope was then established through discussing thoughts and ideas the team had. The goal was to then come up with a list of requirements we could put forward to our customer.

The team decided that we should use stakeholder driven techniques for eliciting information from the customer. To do this we had to make sure that we had knowledge of the organisation "employing" us (which we did as members of the university) and that we had knowledge of the domain (i.e computer games). Our main stakeholder is the customer. A good stakeholder fits the following criteria: is in a relevant position, has a level of domain expertise, has had exposure to the problems and has influence in systems acceptance [1]. The team felt that our customer fit all of the criteria above.

1.2 Document Conventions

The requirements were then split into two categories, functional and non-functional requirements. The functional being anything that had to do with game play and interaction with the user, the non-functional was everything that was required in the backend for this to work on our system. We made requirement "headers" based on a requirements engineering book by Professor Axel van Lamsweerde [1].

The confidentiality and security requirements did not seem to fit in our project as there will be no personal information stored on the game. Therefore, we have taken these headers out of our list. We have placed quality and performance requirements under the same section as they have many coinciding requirements for our project.

1.3 Intended Audience and Reading Suggestions

The reader should have some basic computer science knowledge to understand the requirements. They should be understandable for a customer. The reader should have also consulted the product brief (appendix 1).

1.4 Product Scope

Our goal is to have a completed product by the 3rd of May 2017. Please refer to "Organization" and "Planning for the Future" (found in the methodology PDF file) for further information.

[1] A. Lamsweerde, *Requirements engineering: from system goals to UML models to software specifications*. Chichester, England: John Wiley, 2009.

2. Overall Description

2.1 Product Perspective

This product is the first generation and has had no product history. It is not linked to any other software of its kind and will run independently.

2.2 Product Functions

- Create a 5-phase turn-based game for 2 players.
- Allow players to make actions during appropriate phases, such as: purchasing tiles, managing "roboticons", and harvesting and trading resources.

2.3 User Classes and Characteristics

For this we have used UML to create an abstract overview of the user classes and characteristics. These can be found later in the document.

2.4 Operating Environment

- The final game size should not be larger than 1GB in size as to make sure the distribution of the finished software is not too difficult.
- The game should run on Windows 10 operating system with a current Java installation (at least).
- The game should be playable on the computers in CSE 069/070 and CSE 270.

2.5 Design and Implementation Constraints

Or software will be used for promotion of the university on open days and UCAS days, and as such, it should promote a positive image of the university. The product will be limited to English only. The design must comply with the system features in section 3.

2.6 Requirement Risks

We have included general requirement risks; these can be found in the risk assessment document.

2.7 User Documentation

We will supply a full java documentation once the software is complete.

3. System Features

3.1.0. *The GUI must present a map of the university, subdivided into plots*

- The game must contain plots.

3.1.2. *Players must be able to purchase and own plots*

- Plots may be owned by a player or the "market".
- All plots initially belong to the "market".
- It must be possible to change the ownership of a plot.
- All the plots should be worth the same amount of money.
- The player shouldn't start with any resources however should start with some money.

3.2.0. *There must be roboticons*

- The game must contain roboticon objects.

3.2.1. *Players must be able to purchase roboticons from the market*

3.2.1.1 *It must be possible for roboticons to be owned*

  - Each roboticon must be associated with a plot or in an "unassigned" status.
  - Unpurchased roboticons must be owned by the market.

- It must be possible to change the ownership of a roboticon.
- Each roboticon must have a relationship with a plot.
- It must be possible to change the plot a roboticon is related to.
  - There can only be one roboticon per plot at any one time.

- The set of plots a roboticon may be assigned to must be restricted to the set of plots owned by the owner of the roboticon.

3.2.2. *Unassigned roboticons cannot produce resources*

3.2.3. *Roboticons must have a specialization unless unassigned*

- Each roboticon must have a set of three values, which act as a base production rate.
  - All roboticons must start unspecialized.
- It must be possible to change the specialization of a roboticon.

  3.2.3.1. *Roboticons should only be able to produce one resource at a time*
  - There should be no limit to how many resources there are on a plot of land.

3.2.4. *Specialized roboticons must produce resources when assigned to tiles*

- Each roboticon must have a calculated final production value.
  - The base production rate is a flat value from the roboticon's specialisation.
  - The plot modifier is a multiplier from the assigned plot being worked.
  - The event modifier is a multiplier from the roboticon's owner.

3.3.0. *There must be a market*

3.3.1. *Players must be able to buy and sell resources at the market*

- There should be a limit on how much the market is able to buy.
- The market must respond to supply and demand.
  - The market must be able to display these values to players.
- The market must represent the prices of each resource.
  - There must be a mark-up on prices (i.e. the prices to buy and sell should be different).

3.3.2. *Players must be able to buy roboticons from the market*

- The player should be able to move the roboticons and sell them back to the market.
- The market must track the set of roboticons it owns.

  3.3.2.1. *The market should produce roboticons from ore*
  - The market must be able to instantiate new roboticon objects.
  - There must be a representation of the ore-roboticon exchange rate.
  - The market must have conditions for producing roboticons.
    - Production cap per round.
    - Required ore to begin production.

3.3.3. *The market must have a gambling system*

- The market should process gambling requests from players.

3.4.0. *There must be random events*

- Each event must have a weight; these may be equal.

  3.4.1. *It must be possible for nothing to happen*
- You can choose which player you want to "strike" (i.e use the random event as a weapon).

3.5.0. *Rounds*

3.5.1. *The rounds must be divided into phases*
- There must be a way for players to indicate they are ready to move to the next state.

3.5.2. *Rounds 2 and 3 must have time limits*
- The game must indicate the elapsed or remaining time to the players.

3.6.0. *Game end and scoring*

3.6.1. *The game should end after the round in which the last plot is bought*
- The game must recognize when all plots are owned by players; this may be done by checking the number of plots owned by the "market".
- At the end of the round, the game must begin the endgame procedures, instead of starting another round.

  3.6.1.1. *The game should last approximately 20 to 30 minutes*
  - There must be an appropriate number of plots to result in the required duration.

3.6.2. *There must be a scoring system*
- The game must maintain a set of values representing the score for each of:

        - Unit of food, ore, and energy owned by players; money owned by players; plots owned by players; roboticons owned by players.
    - The game must calculate the total score for each player.
        - For each player, the game must get the quantity of each type of resource that player owns, multiply by the appropriate score values, then sum the totals.
    - The game must compare these scores and select the player with the highest score.

4. Other Non-functional Requirements

4.1 Quality and Performance requirements

- There should be no discernible lag or stuttering in the game as this would take away from the gaming experience.
- The game should be enjoyable, through rigorous feedback sessions [2].

4.2. Integrity requirements

- The rules of the game must be consistently enforced.

4.3. Reliability requirements

- The game should boot properly 98% of the time, after being tested 50 times.

4.4. Interface requirements

- The game should be playable from one computer, and both players should be able to play on it.

4.5. Compliance requirements

- The game should comply with any relevant legislation on defamation.

[2] F. Fu, et al., " EGameFlow: A scale to measure learners' enjoyment of e-learning games" *Computers & Education*, vol. 52, no. 1, pp. 101-112, January 2009.