SE 3XA3: Software Requirements Specification
Send Help - Snake 2.0

Team #34, Team Send Help
Harrison Lau (lauh3)
Geroge Mo (moz)
Vanessa Truong (truonv1)

November 21, 2017

Contents

List of Tables

Table 1: **Revision History**

Date	**Version**	**Notes**
September 29	1.0	SRS first created.
October 3	1.1	Section 4 completed.
October 6	1.2	Sections 1 through 3 completed.
November 21	1.3	Revision 1 changes

This document describes the requirements for our reimplementation of the classic arcade game, Snake. The template for the Software Requirements Specification (SRS) is a subset of the Volere template (Robertson and Robertson, 2012). If you make further modifications to the template, you should explicity state what modifications were made.

1 Project Drivers

1.1 The Purpose of the Project

The purpose or objective of this project is to re-master or revamp the classic Snake game with our Snake 2.0, giving a new and modernized twist to an old classic. **The scope of this project** was to use Java to re-create snake and add additional functions including, but not limited to changes in map size and shape (at least 1) and power ups for the 'snake' (at least 1, aim for 2 or more). **The motivation of this project** comes from the fact that a lot of classic games are being forgotten by the newer generations as their simplicity and lack of additional elements appear to be less attractive. By adding new elements perhaps the youth of today can also be interested in games of our youth.

1.2 The Stakeholders

1.2.1 The Client

The client is the professor of the SE3XA3 course.

1.2.2 The User

The user is anyone who uses the finished program (anyone who plays Snake 2.0).

1.2.3 Other Stakeholders

The project development team are the other stakeholders.

1.3 Mandated Constraints

Design Constraints:

The design shall retain all the aspects of the original game while incorporating new, simple but effective added aspects. The final design should not be over complicated and should be kept within our skill set as well as project purpose.

Safety Constraints:

The re-mastered design shall be safe for all users. Additional features shall look to keep the game playable for all, even those with potential medical issues, such as those suffering from epileptic attacks.

1.4 Naming Conventions and Terminology

Naming convention should be simple and easily understandable.

'Snake' will refer to the original classic game.
'Snake 2.0' will refer to the re-mastered version of 'Snake'.
'the snake' will refer to the user controlled character within the game.
'pellet' will refer to the object that 'the snake' consumes to grant points and/or 'powerup's.
'powerup' will refer to a type of 'pellet' that grants an additional effect.
'wall' will refer to the end areas of a map where the 'snake' can die **'map'** will refer to the field of the game, which ends in 'walls'.

1.5 Relevant Facts and Assumptions

It is assumed that those using Snake 2.0 have a basic and fundamental understanding of the game, Snake.

2 Functional Requirements

2.1 The Scope of the Work and the Product

The scope of this project is to use Java to re-create snake and add additional functions including, but not limited to, changes in map size and shape (at least 1) and power ups for the 'snake' (at least 1, aim for 2 or more).

2.1.1 The Context of the Work

This project is for the purpose of learning more about the software documentation process as well as attaining stronger skills in the actual coding aspects. To elaborate, we aim to grasp a better understanding of the steps and details taken from initiating a software project to finalizing the product. We also hope to develop practical experience in building a program alongside a software team, and refine our skills in coding with Java and GUI libraries.

2.1.2 Work Partitioning

All project members will equally contribute as developers and assume equal division of work. The modules for the project will be assigned according to each member's strongest area of interest or skills. Work partitioning will be discussed more during team meetings and when implementation begins.

2.1.3 Individual Product Use Cases

Use Case UC-1: PlayGame

Related Requirements: REQ1, REQ2, REQ3, REQ6, and REQ7 as stated in next section
Initiating Actor: Player
Actor's Goal: To enjoy the full functionality of the game and have it respond in accordance to their inputs.
Particpating Actors: ViewInstructions, ChangeSettings
Preconditions: The system displays the startup screen with the menu listing choices of "Instructions" and "Settings."
Postconditions: The system will start the game with all related requirements fulfilled. The points counter will be recorded when the game execution

is complete.
Flow of Events for Main Success Scenario:

1. **Player** opens the game and is brought to the start-up menu screen.
2. **Player** clicks the "Play" button and **PlayGame** function executues.
3. **PlayGame** execution finishes and records 'points counter' score.
4. Option for "Play Again?" pops up.
5. **PlayGame** re-executes if necessary.

Flow of Events for Extensions (Alternate Scenarios):
2a. **Player** clicks the "Play" button, **PlayGame** does not properly execute.

1. The system will display a pop up message of an error occurring.
2. The system will close the application entirely.

Use Case UC-2: ViewInstructions

Related Requirements: REQ4 as stated in next section
Initiating Actor: Player
Actor's Goal: To learn and understand the game's objectives and features.
Particpating Actors: PlayGame, ChangeSettings
Preconditions: The system displays the startup screen with the menu listing choices of "Instructions" and "Settings."
Postconditions: The system will bring the player to the "Instructions" page and provide a full outline of how to play.
Flow of Events for Main Success Scenario:

1. **Player** opens the game and is brought to the start-up menu screen.
2. **Player** clicks the "Instructions" button.
3. **ViewInstructions** function executues.
4. **Player** clicks the "Back" button.
5. **Player** returns to start-up menu screen.

Flow of Events for Extensions (Alternate Scenarios):
2a. **ViewInstructions** does not properly execute.

1. The system will display a pop up message of an error occurring.
2. The system will return to the start-up menu screen.

Use Case UC-3: ChangeSettings

Related Requirements: REQ4, and REQ5 as stated in next section

Initiating Actor: Player
Actor's Goal: To change the attributes of the game, such as the map size.
Particpating Actors: PlayGame
Preconditions: The system displays the startup screen with the menu listing choices of "Instructions" and "Settings."
Postconditions: The system will bring the player to the "Settings" page and change the game settings according to user's inputs.
Flow of Events for Main Success Scenario:

1. **Player** opens the game and is brought to the start-up menu screen.
2. **Player** clicks the "Settings" button.
3. **ChangeSettings** function executues.
4. **Player** changes settings and new settings are applied.
5. **Player** returns to start-up menu screen.

Flow of Events for Extensions (Alternate Scenarios):
2a. **ChangeSettings** does not properly execute.

1. The system will display a pop up message of an error occurring.
2. The system will return to the start-up menu screen.

4a. New settings are not applied to **PlayGame**.

1 **ChangeSettings** will deliver a message to notify user of improper changes.

2.2 Functional Requirements

The game shall have a base map of appropriate size. The game shall include a 'snake' character model that can 'grow' upon eating 'pellets' and 'die' upon self collision or collision with the 'wall' of the map. A summary of the functional requriements is listed below:

1. **FR1.** The 'snake' should only be able to move in the (4) ordinal directions.
2. **FR2.** The game shall generate (1) pellet at a random location at all times that is within the map and at an empty spot.
3. **FR3.** The game shall generate (1) powerup sporadically at a random location within the map at an empty spot.
4. **FR4.** The game shall have an intro screen that includes instructions on the game and map selection.

5. **FR5.** The game shall allow for different sized maps.

6. **FR6.** The game shall have a 'points counter' that displays the points from pellets eaten and powerups

7. **FR7.** The game shall use a GUI to display said map and components.

3 Non-functional Requirements

3.1 Look and Feel Requirements (NFR1)

Our reimplentation of Snake will embody the original game's overall simplicity while also projecting a modern feel for players. It will exhibit upgraded properties and enhancements to the gameplay that will epitomize how the basic objectives of a classic game can be revamped with stimulating upgrades. By customizing the physical aspects, we can attain a look and feel for the game that will attract modern-day gamers and spark a refreshing interest and appeal to the community of old school gamers as well.

Snake 2.0 will differentiate from the original game in terms of map layout and design. It will incorporate more complex map shapes as opposed to the standard simple box-shaped map, hence increasing difficulty and creating a more interesting feel for the game. The interface's colour scheme will also be chosen in favour of a more exciting and lively atmospheric experience, and will most likely consist of colours that pop out more to the eye.

3.2 Usability and Humanity Requirements (NFR2)

Likewise to the original game, players should have no difficulties with maneuvering our redevelopment and the interface should be user-friendly and simplistic enough for players of all ages, for users as young as 6 years of age. Although it will feature several upgrades, such as power-ups and obstacles, it will not overwhelmingly complicate the game or distract from its basic functionality.

The gameplay will still retain its classic objectives and obstacles, however, with the additional features of changing the map shapes and incorporating power-ups and more obstacles, it will reasonability increase the game's level of difficulty and attract users who are looking to challenge themselves further.

Our product will also focus on keeping the interface minimilistic, thereby increasing its user-friendliness towards gamers of various experience levels, from amateurs to professionals.

3.3 Performance Requirements (NFR3)

A breakdown of the performance requirements is as follows:

- Speed Requirements (**NFR3a**)

  The reimplemented game should be quick and responsive to the user's inputs. Outputs and responses should be almost instantaneous, in a timely manner of half a second.

- Safety Critical Requirements (**NFR3b**)

  There should be no incompability issues between the game and the device running it. By that, the game will not inflict any hardware or software damage onto the running device. It will also not emit any dangerously bright light or flashes that may affect users whom are sensitive to light.

- Precision Requirements (**NFR3c**)

  All user inputs will be processed properly by the product's interface. The inputs to the game's controller will be accurately displayed and reflected by the output to the screen. For instance, if the player moves the snake to the left, the product will process that input and output it to the screen within half a second.

- Reliability and Availability Requirements (**NFR3d**)

  Players should be able to access the game at any time of day (24 hours), and 365 days a year. The redeveloped game will run smoothly and responsively, in the absence of bugs and software errors.

- Capacity Requirements (**NFR3e**)

  The reimplementation must be able to run for an indefinite amount of consecutive games. It will only allow for one player per game.

3.4 Operational and Environmental Requirements (NFR4)

Players will most likely be using the game indoors on their computers or mobile devices. As users will most defiintely have different operating systems on their PC's, our reimplementation will be expected to run on various operating systems, graphics cards, and processors, though it should be feasible to run on any technological device.

3.5 Maintainability and Support Requirements (NFR5)

To aid in the mainitainability of our product, testing and debugging will be performed every week from the first day of implementation to ensure that there are no bugs or errors. Any bugs and errors that are found will be recorded for future reference and coonvenience. If an unexpected error does occur at any given time, we will address it during the next testing and debugging session. It will also be expected to run across different operating systems (Windows, Apple, and Linux).

3.6 Security Requirements (NFR6)

All code files, documents, and reports for the project are included in the Git repository, which has been set to private and thus will not be available to the public. Only the team members, the course instructors, and teaching assistants will be able to access the private repository and all of its included files. The team members will be responsible for all auditing and logging of current and future documents, reports, and code. The log history will be tracked through the Git repository and will include details on what project files have been updated and added by the contributor at the given time. The course instructors and teaching assistant will also have the same permissions to the repository as the team members, however, for the purposes of this project, they will most probably only be accessing the files for grading.

3.7 Cultural Requirements (NFR7)

The reimplementation will not contain discriminative, offensive, or sensitive content pertaining to any country, religion, or race. It will avoid all implicit and explicit references to cultural and polictal issues that may be offensive or sensitive to users.

3.8 Legal Requirements (NFR8)

The source code that we have attained for reference for our reimplementation of the game has been provided under the MIT License, which grants us permission to use, copy, modify, publish and redistribute the software, so long as we comply with the MIT License policies.

3.9 Health and Safety Requirements (NFR9)

For the comfort of those users prone to health conditions, such as motion sickness and in more severe cases, epilepsy, our redevelopment will avoid use of bright and dramatic flashes and lights as well as loud and startling noises, which may trigger these health concerns. As the prolonged use with any technological device emitting blue light can inflict eye strain, the game will include health warnings and text that will remind the players to relax or take a break.

4 Project Issues

4.1 Open Issues

Though the game's basic functionality will not necessarily prove to be an issue to implement, the added features we wish to implement may be an issue. For example, one of the power ups could make the player be able to bounce off walls instead of losing the game for a short period of time. But that begs the question of how does it bounce off? What happens if it gets stuck and can't bounce in any direction? A number of these problems, however, can be solved with design decisions or adding to the game's rules as the team develops the game. Therefore, there aren't any forseeable open issues in development.

4.2 Off-the-Shelf Solutions

Since Snake is such a simple game in and of itself, there are a number of implementations created through various programming languages. As such, it is not difficult to find an implementation done in Java should the team get stuck during implementation. Of course, the original source code will be cited for any pieces of code that we did not write ourselves. Perhaps

instead of copying code, however, we can choose an implementation done in a language none of us are familiar with and try to follow along with their logic, thereby learning as we go along as opposed to copying code.

4.3 New Problems

The problem with the current environment is that not many people are still interested in an old game such as Snake. A fresh reimplementation that upgrades the original to feel like a new game would attract a variety of players. Children to adults to elderly people will be able to play the game without any trouble. That includes existing players as well. With new aspects such as power-ups being thrown in, existing players may see the potential of playing it more competitively and seriously, thus attracting them to our implementation.

A strong limitation with Java written programs is that not every single device is capable of running it compared to JavaScript, for example. There doesn't seem to be a fix for this aside from switching languages but that would be out of the question, at least given the amount of time we have.

4.4 Tasks

A general outline of tasks that need to be completed is:

1. Complete vanilla implementation of Snake.
    - Includes testing, documentation and presentation for Proof of Concept.
2. Add in our modifications one by one.
    - Allow for different sized stages.
    - Potential power-ups include speed up, slow down, collision immunity and mitosis (splitting into two separately controllable snakes).
    - Each addition will be tested rigorously to ensure basic functionality is not compromised.

The general task outline also serves as a basic development phase list. For example, about a week for vanilla implementation, two days for rigorous testing, two days for documentation, etc. Development phases will closely follow the timeline described by the project's Gantt chart.

4.5 Migration to the New Product

Migration to our reimplementation will be done through attraction and advertising. Considering Snake 2.0 doesn't make the vanilla version obsolete per se, there isn't so much a total migration as it is a shift in players. Like any game, there will be a surge of players in the beginning and eventually it will plateau to a smaller number (with most games, at least).

It is hard to imagine that there is a large number of existing users for vanilla Snake playing at the same time at any time. As such, migration from the old product would be scarce.

4.6 Risks

There aren't any inherent risks with the vanilla version of Snake but with our reimplementation, new risks may arise. For example, if we add a visual effect to the game such as when power-ups are in effect, we need to be careful not to induce epileptic attacks for players. Thus we need to be rather frugal with our use of special effects or sound effects, if any.

4.7 Costs

There should not be any costs associated with the project unless the team decides to monetize downloads. Though that would potentially involve moving the development environment to be in a mobile game development environment. Considering none of the members have experience in successful or at least functional application development, it would likely be out of the term's allotted time.

4.8 User Documentation and Training

In the starting screen before starting the game, there will be an option to choose a Rules and Instructions button that will bring users to a separate screen. There, the user will be provided with the general rules associated with vanilla Snake and then will describe each of the power-ups. Tutorials would be too cumbersome and unnecessary for such a simple game so an instructions screen shall suffice.

4.9 Waiting Room

Although mentions of special effects and sound effects are mentioned above, they are not a main priority and thus are categorised to be in the waiting room until basic implementations have been completed. Along the same note, making the game look appealing isn't a top priority so it will therefore be put on the backburner until development is far enough in its development and time permitting.

4.10 Ideas for Solutions

To reiterate:

- For the vanilla implementation, follow along with the readily available open source code projects of Snake. In desperate need, focus on Java implementations of Snake. Following this idea, make sure vanilla implementation is fully functional before adding anything to it.
- Be mindful of users prone to epileptic attacks and be frugal with uses of special effects (if any).
- Take the time to implement each power-up carefully to avoid breaking the game.a
- Create an Instructions screen to instruct players on how to play the game.

Index

The following section was added to this SRS. The index shall link to the functional and non-functional requirement labels earlier in the document.

- NFR6
- NFR7
- NFR8
- NFR9

References

James Robertson and Suzanne Robertson. *Volere Requirements Specification Template*. Atlantic Systems Guild Limited, 16 edition, 2012.