Software Requirements Specification For Space Invaders

Version 0.4.0

Prepared by Group 1 Specification
Kingsley Samuel
Yifeng Zheng
Raeean ahmed
Adam Chen
Jacky Chen

Table of Contents

Introduction

1.1 Purpose

The purpose of this document is to present a detailed description of the development of software Space Invaders. It will explain the purpose and features of the software, the interface of the software, what the software will do and the constraints under which it must operate. This document is intended for the users of the software and the developers.

1.2 Document Conventions

The document was created based on IEEE standards for System Requirement Specification Documents.

1.3 Intended Audience and Reading Suggestions

- Programmers who are required to develop on this project as well as fixing existing bugs
- Average/ Typical User such as students, who want to use the software for entertainment purposes
- Professional User such as engineers, researchers, teachers who want to understand the development process and procedures.

1.4 Product Scope

Space Invaders is a game purely made for entertainment purposes. Users can use it to test the skills and pursue challenges to gain the top score.
Source Code
Fonts

Consist of a collection of type fonts, the default font should be helvetica. You can place whichever font you wish to use within this folder as long as it is a .ttf file. Formats such as .otf, .ttc, or .dfont are unsupported font file types.

Sounds

This folder is a list of sound effects that would work as an extension to the actual game. .WAV files will be the only supported music file type because it is the most reliable in terms of software compatibility with OS.

Images

This folder consists of images such as the aliens, ship, background images etc. all virtualized images would be included in this folder.

Assumptions:

- For Apple user issues, a workaround is possible to implement with a reasonable amount of effort and risk. Otherwise, all we can do is accept bug reports until native apple support for the new chipset becomes compatible.
- The project will be done as a collective of 4 groups Specifications, Graphics, Quality Assurance, Backbone.

Constraints:

- Application should be created and functional by May 13th 2021 (End of Semester).
- There is a small chance that does not create full functionality of all objects in the product

Product exclusion:

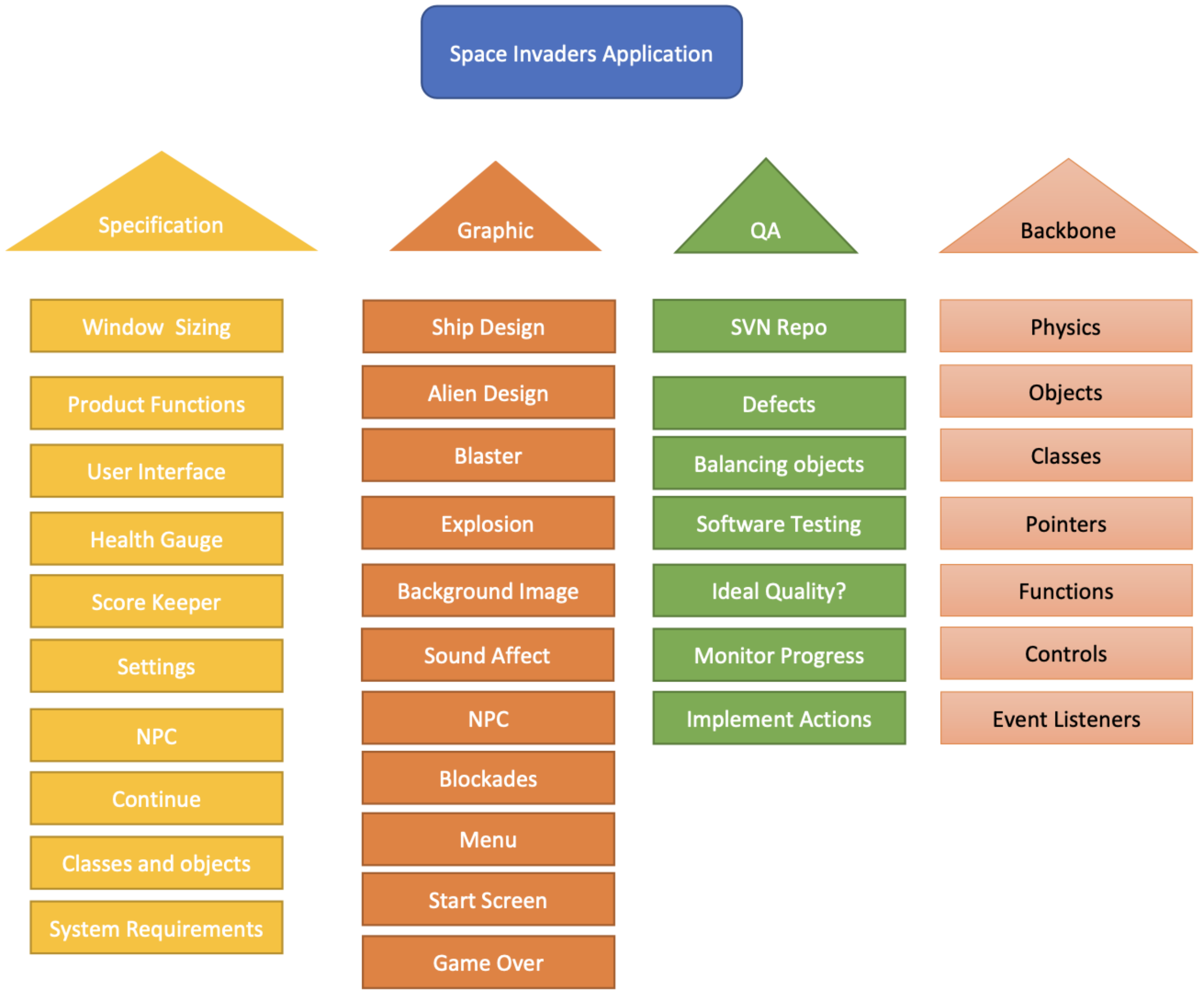
- IOS support
- UI/UX dynamic changes

Scope Statement

The project will be developed based on the information provided by this Specification documentation.
First of all, the team will analyze the project and understand what task must be done and completed to progress through the development of this software. The goal is to have a completed software in which the Specifications provided are met.

This project would be divided into four teams which include Specs, Graphics, QA, and Backbone. Specifications provide the specific standards and expectations for this project.

Graphics produce physical features to be assigned to pointers within the software. Quality Assurance provides a functional svn repo in which changes can be studied and rollback if need be. Backbone develops all primary functions needed for the software to run.



1.5 References

<THIS SECTION IS RESERVED FOR REFERENCES TO ANY SOFTWARE USED IN THE DEVELOPMENT. UPDATE AS MORE TOOLS ARE USED>
<IEEE reference citation>

Overall Description

2.1 Product Perspective

Space Invaders was developed for everyone who is interested in playing a retro game and for developers to test their skills and understanding of material we've learned thus far.
It is an open source project meaning its original source code is freely available and can be modified. It has a very active and determined development team to support it and will provide feedback to users. It was developed to run on <PLACEHOLDER>

2.2 Product Functions

1. Homepage
    a. Homepage should include play button, settings button which may includes a sound button
2. Play Button
    a. The play button should render the screen that displays the game
3. Settings Button
    a. The settings button should include a list of limitations/changes that affect the UX of the game
    b. List must included sound, restart, homepage, exit
    c. Should the settings button be clicked, the game should pause and only resume if the exit button is clicked
    d. The settings button should be in a fixed position where users can access it in the homepage and the game page
4. Sound Button
    a. The sound button creates an option option to remove the sound produced from the game including any sound effects
5. Restart Button

        a. The Restart Button resets the game to the preset values of the game
6. Homepage Button
    a. Homepage button should reset the game and take user to the homepage
7. Exit Button
    a. Once clicked, ther game resumes and the render of the settings overlay should be removed
8. Button Inputs
    a. LEFT and RIGHT arrow keys
        i. LEFT arrow key will move <X> amount of pixels left
        ii. RIGHT arrow key will move <X> amount of pixels right
    b. SPACE Key
        i. SPACE key will fire a ball/eclipse from the top of the spaceship positioned in the center
        ii. The bullet will fire <X> amount of pixels up
9. Score Keeper
    a. A score keeper will be centered at the top of the screen in the middle to track the amount of points a users gained per session
10. Health gauge
    a. Keeps track of how many lives a user has
    b.

2.3 User Classes and Characteristics

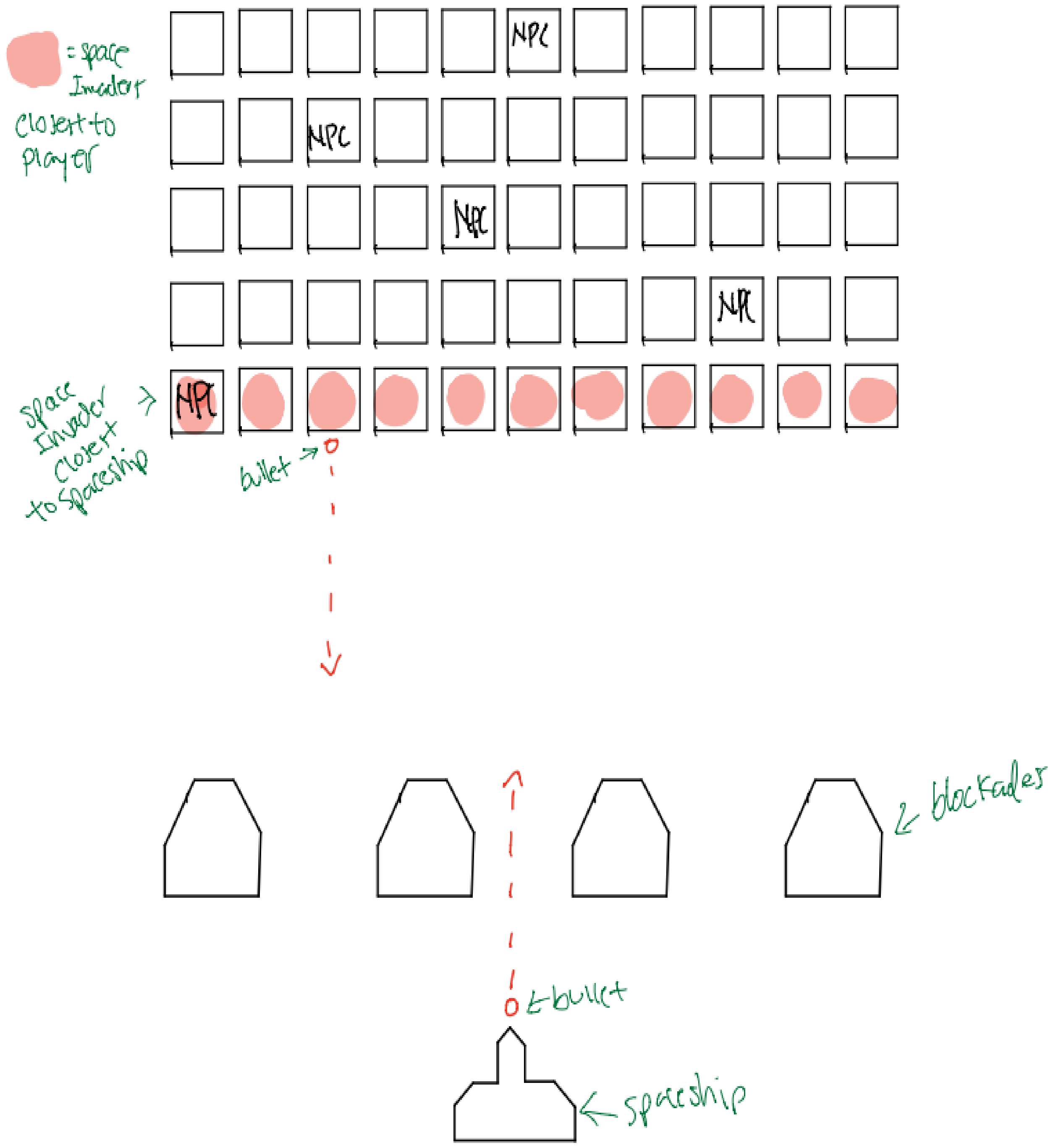
Users should have basic knowledge of computers and understand how to navigate through the basic web user interface. There should only be one class of users that has access to all functions of the Space Invaders Game

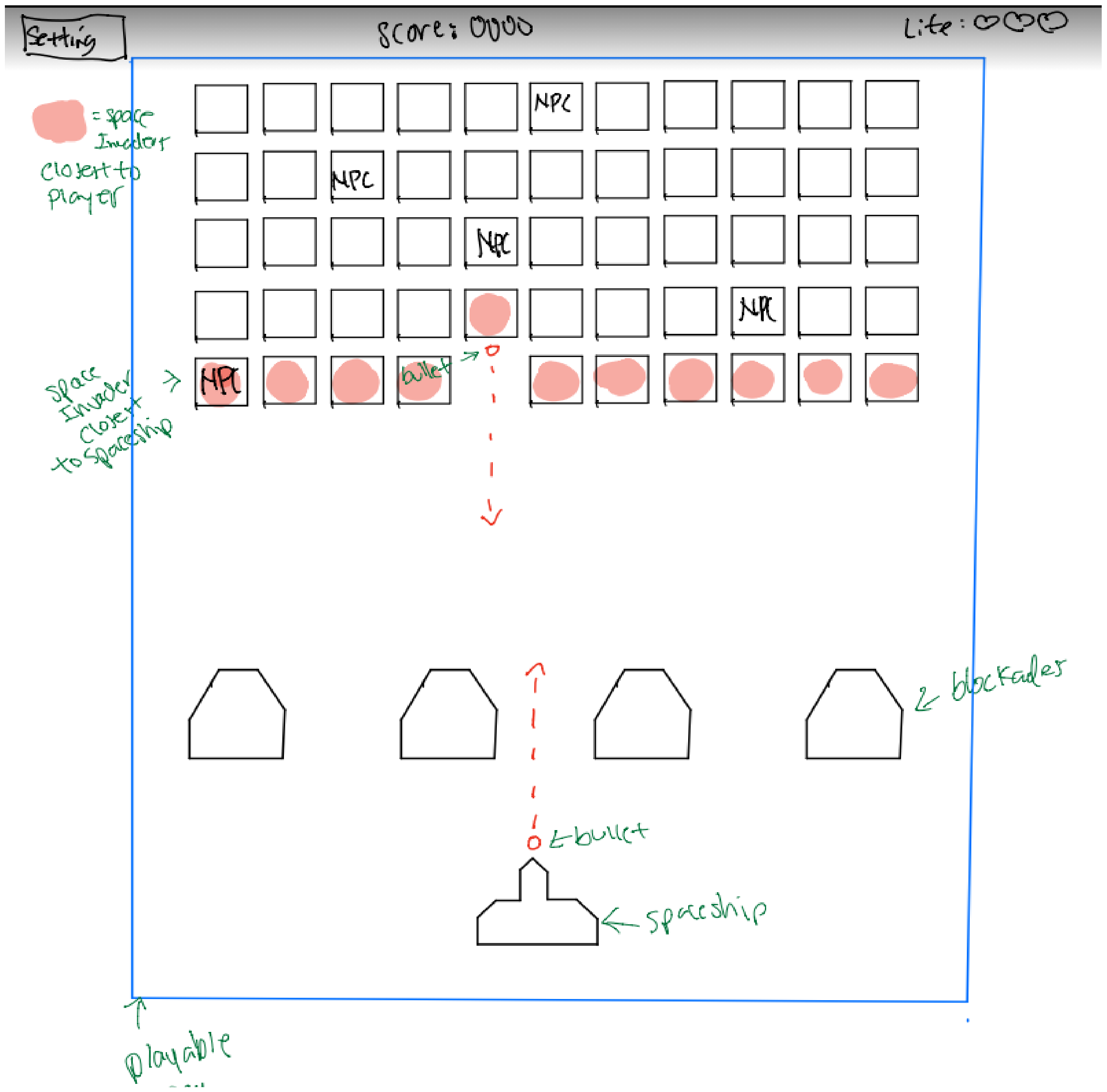
2.4 Design and Implementation

1. Screen Size/Playable area: 16:9 aspect ratio; 1920x1080 px
2. The PLAY button should be centered in the screen on the homepage
    a. Once clicked, the game page will be rendered
    b. There will be a 3 second timeout before the game starts
3. The SETTINGS Button should be positioned absolutely in the top left corner of the screen
4. The design of the webpage should be dark themed with no conflicting colors
5. There should be a playable area that doesn't take up the entire screen
6. There is 5 rows by 11 columns of space invaders with equal dimensions of <X> width and <X> height and spaced apart equally by <X> pixels
    a. There should be at least 5 NPC amongst the Space Invaders
        i. Should the NPC be shot by the player, an excruciating scream will be played

   b. The group of space invaders will move <X> amount of pixels every <X> amount of seconds to the left or right.
      i. If the group of space invaders is at the left edge or right edge of the playable area, the whole group will move <X> amount of pixels down.
      ii. If space invaders are at the right edge, they will all move towards left edge and vice versa
         1. Program should check if the number of pixels moved will be out of bounds
         2. If it will be out of bounds, the position of the space invaders will be at the edge of the playable area instead of out of the playable area
   c. There should 3 at least 2 groups of space invaders, the NPC and the Invaders
      i. NPC contain no points and players should not hit them
         1. Once hit, a scream sound is played
         2. User loses a life
      ii. The invaders contain <X> amount of points
         1. Once hit by the bullet from the spaceship, the invader is removed from the screen
            a. An explosion animation would be prefered but removal upon contact of bullet is fine
         2. Once removed, <X> amount of points will be added to the Scorekeeper
            a. The amount of points should be constant throughout the space invaders
   d. At <X> second intervals, one space invader at a particular column closest to the spaceship will fire a bullet centered at the bottom center
7. There should be 1 row by 4 columns of objects called blockcades which blocks <X> number of hits from the space invaders.
   a. Bullets blocked from space invaders should decreases its dexterity by 1 and lower the count of hits needed until it is destroyed
   b. Bullets fired from spaceship will be blocked but decrease in dexterity should occur
8. Health gauge
   a. A health gauge will be fixed positioned on the top right of the screen containing at least 3 lives
      i. Once the spaceship is hit by the invaders:
         1. The game is paused for <X> seconds to allow spaceship to respawn
         2. In <X> seconds, the spaceship will be rendered at the bottom of the game an centered in the middle

9. Health gauge
    a. A health gauge will be fixed positioned on the top right of the screen containing at least 3 lives
        i. Once the spaceship is hit by the invaders:
            1. The game is paused for <X> seconds to allow spaceship to respawn
            2. In <X> seconds, the spaceship will be rendered at the bottom of the game an centered in the middle
10. ScoreKeeper
    a. A score keeper is placed center top to track points gained from destroyed space invaders
11. Should all space invaders be destroyed, a popup will be rendered stating:
    a. "YOU WIN"
    b. Points earned this session
    c. Continue
12. Should all lives be lost, a popup will be rendered stating:
    a. "YOU LOSE"
    b. Points earned this session
    c. Restart
13. Continue
    a. Continue will render a new level for the player
        i. In each new level, the number of NPCs will be increase by <X>
        ii. The number of space invaders firing bullets will increase by <X>
        iii. The amount of pixels the space invaders move towards the dges will be increased


= space Invader
closest to player
NPC
NPC
NPC
NPC
NPC
space Invader closest to spaceship
bullet
blockades
bullet
spaceship


Setting
Score: 0000
Life:
= space Invader closer to player
NPC
NPC
NPC
NPC
NPC
space Invader closer to spaceship
bullet
blockades
bullet
spaceship
playable

2.5 Operating Environment

The operating environment for this program requires an up to date browser and internet connection.

2.6 User Documentation

2.7 Assumptions and Dependencies

3. Specific Requirements

This section contains all the software requirements at a level of detail sufficient to enable designers to design a system to satisfy those requirements, and testers to test that the system satisfies those requirements. Throughout this section, every stated requirement should be externally perceivable by users, operators, or other external systems. These requirements should include at a minimum a description of every input (stimulus) into the system, every output (response) from the system and all functions performed by the system in response to an input or in support of an output. The following principles apply:

(1) Specific requirements should be stated with all the characteristics of a good SRS
- correct
- unambiguous
- complete
- consistent
- ranked for importance and/or stability
- verifiable
- modifiable
- traceable

(2) Specific requirements should be cross-referenced to earlier documents that relate
(3) All requirements should be uniquely identifiable (usually via numbering like 3.1.2.3)
(4) Careful attention should be given to organizing the requirements to maximize readability
(Several alternative organizations are given at end of document)

Before examining specific ways of organizing the requirements it is helpful to understand the various items that comprise requirements as described in the following subclasses. This section reiterates section 2, but is for developers not the customer. The customer buys in with section 2, the designers use section 3 to design and build the actual application.

Remember this is not design. Do not require specific software packages, etc unless the customer specifically requires them. Avoid over-constraining your design. Use proper terminology:
The system shall… A required, must have feature
The system should… A desired feature, but may be deferred til later
The system may… An optional, nice-to-have feature that may never make it to implementation.

Each requirement should be uniquely identified for traceability. Usually, they are numbered 3.1, 3.1.1, 3.1.2.1 etc. Each requirement should also be testable. Avoid imprecise statements like, “The system shall be easy to use” Well no kidding, what does that mean? Avoid “motherhood and apple pie” type statements, “The system shall be developed using good software engineering practice”

Avoid examples, This is a specification, a designer should be able to read this spec and build the system without bothering the customer again. Don't say things like, "The system shall accept configuration information such as name and address." The designer doesn't know if that is the only two data elements or if there are 200. List every piece of information that is required so the designers can build the right UI and data tables.

3.1 External Interfaces

This contains a detailed description of all inputs into and outputs from the software system. It complements the interface descriptions in section 2 but does not repeat information there. Remember section 2 presents information oriented to the customer/user while section 3 is oriented to the developer.
It contains both content and format as follows:

- Name of item
- Description of purpose
- Source of input or destination of output
- Valid range, accuracy and/or tolerance
- Units of measure
- Timing
- Relationships to other inputs/outputs
- Screen formats/organization
- Window formats/organization
- Data formats
- Command formats
- End messages

Keyboard

Keyboards are used to control the spaceship by using the arrow keys or the more mainstream WASD key which are provided on the keyboard.

(Arrow key)

(WASD key)

- The input of each keystroke should not have any delay.

The SPACEBAR key will function as a fire key, whenever the key is pressed the SPACESHIP the player is controlling will shoot laser.

(SPACEBAR)

3.2 Functions

Functional requirements define the fundamental actions that must take place in the software in accepting and processing the inputs and in processing and generating the outputs. These are generally listed as “shall” statements starting with "The system shall…

These include:

- Validity checks on the inputs
- Exact sequence of operations
- Responses to abnormal situation, including
- Overflow
- Communication facilities
- Error handling and recovery
- Effect of parameters
- Relationship of outputs to inputs, including
- Input/Output sequences
- Formulas for input to output conversion

It may be appropriate to partition the functional requirements into sub-functions or sub-processes. This does not imply that the software design will also be partitioned that way.

3.3 Performance Requirements

This subsection specifies both the static and the dynamic numerical requirements placed on the software or on human interaction with the software, as a whole. Static numerical requirements may include:

(a) The number of terminals to be supported
(b) The number of simultaneous users to be supported
(c) Amount and type of information to be handled

Static numerical requirements are sometimes identified under a separate section entitled capacity.
Dynamic numerical requirements may include, for example, the numbers of transactions and tasks and the amount of data to be processed within certain time periods for both normal and peak workload conditions.
All of these requirements should be stated in measurable terms.
For example,
95% of the transactions shall be processed in less than 1 second
rather than,
An operator shall not have to wait for the transaction to complete.
(Note: Numerical limits applied to one specific function are normally specified as part of the processing subparagraph description of that function.)

3.4 Logical Database Requirements

This section specifies the logical requirements for any information that is to be placed into a database. This may include:

- Types of information used by various functions
- Frequency of use
- Accessing capabilities
- Data entities and their relationships
- Integrity constraints
- Data retention requirements

If the customer provided you with data models, those can be presented here. ER diagrams (or static class diagrams) can be useful here to show complex data relationships. Remember a diagram is worth a thousand words of confusing text.

3.5 Design Constraints

Hardware Limitations = There shouldn't be any hardware limitations useless the computer is literally powered by a potate. This is not a AAA game and thus we expect this game to run on any computer with a functional browser.

3.5.1 Standards Compliance

Specify the requirements derived from existing standards or regulations. They might include:
(1) Report format
(2) Data naming
(3) Accounting procedures
(4) Audit Tracing

For example, this could specify the requirement for software to trace processing activity. Such traces are needed for some applications to meet minimum regulatory or financial standards. An audit trace requirement may, for example, state that all changes to a payroll database must be recorded in a trace file with before and after values.

3.6 Software System Attributes

There are a number of attributes of software that can serve as requirements. It is important that required attributes are specified so that their achievement can be objectively verified. The following items provide a partial list of examples. These are also known as non-functional requirements or quality attributes.

These are characteristics the system must possess, but that pervade (or cross-cut) the design. These requirements have to be testable just like the functional requirements. It's easy to start philosophizing here, but keep it specific.

3.6.1 Reliability

Specify the factors required to establish the required reliability of the software system at time of delivery. If you have MTBF requirements, express them here. This doesn't refer to just having a program that does not crash. This has a specific engineering meaning.

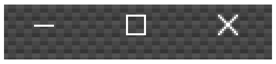
We expect some bugs even when the game is delivered. We do not accept any feedback or complaints once the game is deemed done by the developer.

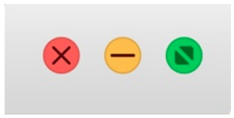
3.6.2 Availability

This game will only run like/as a demo. It will not be available 24/7. Only available when there's a showcase or if someone has a copy and would like to play it.
Checkpoint - There will be no checkpoint available in the game. There will be lives available for the player to lose, after player lose all their live the follow event will happen (Reference 2.4-14)

14. Should all lives be lost, a popup will be rendered stating:
    a. "YOU LOSE"
    b. Points earned this session
    c. Restart

Recovery - There will not be any recovery once the program is deemed unplayable and thus have to restart or close the program. All data of progress will be lost and players have to restart from the beginning.
Restart - The program should let the player restart or close out the program whenever an error had occurred, which it deemed unplayable. By using the top right X button (windows) or top left X button (MacOs)


(Window)


(MacOS)

3.6.3 Security

NO SECURITY AT ALL. This game would not include any security in it. That means please do not enter anything sensitive.

3.6.4 Maintainability

Specify attributes of software that relate to the ease of maintenance of the software itself. There may be some requirement for certain modularity, interfaces, complexity, etc. Requirements should not be placed here just because they are thought to be good design practices. If someone else will maintain the system

3.6.5 Portability

Specify attributes of software that relate to the ease of porting the software to other host machines and/or operating systems. This may include:

- Percentage of components with host-dependent code
- Percentage of code that is host dependent
- Use of a proven portable language
- Use of a particular compiler or language subset
- Use of a particular operating system

Once the relevant characteristics are selected, a subsection should be written for each, explaining the rationale for including this characteristic and how it will be tested and measured. A chart like this might be used to identify the key characteristics (rating them High or Medium), then identifying which are preferred when trading off design or implementation decisions (with the ID of the preferred one indicated in the chart to the right). The chart below is optional (it can be confusing) and is for demonstrating tradeoff analysis between different non-functional requirements. H/M/L is the relative priority of that non-functional requirement.

ID	Characteristic	H/M/L	1	2	3	4	5	6	7	8	9	10	11	12
1	Correctness													
2	Efficiency													
3	Flexibility													
4	Integrity/Security													
5	Interoperability													
6	Maintainability													
7	Portability													
8	Reliability													
9	Reusability													
10	Testability													
11	Usability													
12	Availability													

Definitions of the quality characteristics not defined in the paragraphs above follow.

- Correctness - extent to which program satisfies specifications, fulfills user's mission objectives

- Efficiency - amount of computing resources and code required to perform function
- Flexibility - effort needed to modify operational program
- Interoperability - effort needed to couple one system with another
- Reliability - extent to which program performs with required precision
- Reusability - extent to which it can be reused in another application
- Testability - effort needed to test to ensure performs as intended
- Usability - effort required to learn, operate, prepare input, and interpret output

THE FOLLOWING (3.7) is not really a section, it is talking about how to organize requirements you write in section 3.2. At the end of this template there are a bunch of alternative organizations for section 3.2. Choose the ONE best for the system you are writing the requirements for.

3.7 Organizing the Specific Requirements

For anything but trivial systems the detailed requirements tend to be extensive. For this reason, it is recommended that careful consideration be given to organizing these in a manner optimal for understanding. There is no one optimal organization for all systems. Different classes of systems lend themselves to different organizations of requirements in section 3. Some of these organizations are described in the following subclasses.

3.7.1 System Mode

Some systems behave quite differently depending on the mode of operation. When organizing by mode there are two possible outlines. The choice depends on whether interfaces and performance are dependent on mode.

3.7.2 User Class

Some systems provide different sets of functions to different classes of users.

3.7.3 Objects

Objects are real-world entities that have a counterpart within the system. Associated with each object is a set of attributes and functions. These functions are also called services, methods, or processes. Note that sets of objects may share attributes and services. These are grouped together as classes.

3.7.4 Feature

A feature is an externally desired service by the system that may require a sequence of inputs to effect the desired result. Each feature is generally described in a sequence of stimulus-response pairs.

3.7.5 Stimulus

Some systems can be best organized by describing their functions in terms of stimuli.

3.7.6 Response

Some systems can be best organized by describing their functions in support of the generation of a response.

3.7.7 Functional Hierarchy

When none of the above organizational schemes prove helpful, the overall functionality can be organized into a hierarchy of functions organized by either common inputs, common outputs, or common internal data access. Data flow diagrams and data dictionaries can be use dot show the relationships between and among the functions and data.

3.8 Additional Comments

Whenever a new SRS is contemplated, more than one of the organizational techniques given in 3.7 may be appropriate. In such cases, organize the specific requirements for multiple hierarchies tailored to the specific needs of the system under specification.

Three are many notations, methods, and automated support tools available to aid in the documentation of requirements. For the most part, their usefulness is a function of organization. For example, when organizing by mode, finite state machines or state charts may prove helpful; when organizing by object, object-oriented analysis may prove helpful; when organizing by feature, stimulus-response sequences may prove helpful; when organizing by functional hierarchy, data flow diagrams and data dictionaries may prove helpful.

In any of the outlines below, those sections called “Functional Requirement i” may be described in native language, in pseudocode, in a system definition language, or in four subsections titled: Introduction, Inputs, Processing, Outputs.

System Features

4.1 Graphics Visualization

- **Purpose**

  This document holds simplified explanations of functions through pseudo-code used for the graphic implementation for the game < SPACE INVADERS >. Each function usage and how it should be implemented will be included in this document in regards to graphics. This document is intended for software developers and not for users.

- **Scope**

  This document will only contain key phrases regarding how the executable code is intended to be used. The graphics library will be limited and generalized for any language, additionally it is only intended to be used for the game <SPACE INVADERS> and if invoked for another project will cause unintended errors.

- **Intended Implementation**

  Backend should call graphicsLoop() every time they want a new frame to be displayed. If Backend wants to display the game at 120 fps (frames per second) they should call the function 120 times per second. If they want 80 fps call it 80 times per second.

- **Initialization**

  When the game is started Backend must create all the entities to be displayed using the addEntity(entityData) function. They call it for each individual Entity which needs to be created. Entities include all forms of enemy, bricks/blocks, the player character, the lasers, and other objects which interact with one of the previous.

- ## Creating Entities

addEntity will return a reference each time it is called which backend will have to save in some sort of container. When an entity becomes dead, Backend will use the reference for that entityData to update its isAlive variable to dead and delete their reference. Graphics will handle the display of the entity, its deathAnimation, and deletion of their own reference the next time Graphics Loop() is called.

- ## Updating Entities

When an entity needs to be updated its variables should be changed to the new values through its reference. Eg changing the x and/or y coordinates of an entity, the isAlive as mentioned, and any of the special types specified below for each kind of entity.

- ## Pseudo Code Definitions

a. function graphicsLoop()
   Draws a single frame for the entire game.
   Backend should call this whenever they want the screen to update.

b. function addEntity(entityData)
   entityData format is documented below.
   - Each entity that the backend wants to draw on the screen should have an entityData object.
   - This function adds it to the graphics loop.
   - When entityData.isAlive is set to false, then it will be removed from the graphics loop.
   - The only entity that will be displayed will be the ones that are in the graphics loop.

c. function clearParticles()
   Clears the screen of particles

d. function clearEntities()
   Remove all entities without spawning the death animation

e. entityData format
An entityData object is like this
```
{
        type : <string>,
        isAlive : <boolean>,
        [special attributes …]
}
```

Backend should create this object then put it into the graphics loop via addEntity(entityData).

To edit the entity's attributes (like xy coordinates, or whatever special attributes that entity may
have), keep a reference to the entityData object and edit that.

● Global Variables (entityData)

Variables depend on the type. The ones listed here are shared among all types.

type:

"type" tells us which function to use for drawing the entity.
ex. type : "invader", type : "player"
Each "type" will have its own special attributes.
ex. type player will have "x" and "y", type bullet may have trail : true or trail : false.

isAlive:

"isAlive" should be set to true, if it is set to false then entityData will be removed from the graphics loop and a death animation will spawn.

Nonfunctional Requirements

5.1 Performance Requirements

5.2 Safety Requirements

5.3 Security Requirements

5.4 Software Quality Assurance

INDEX