SE 3XA3: Software Requirements Specification
Tetrileet

Team 15, AAA Solutions
Student 1 Abdallah Taha, tahaa8
Student 2 Andrew Carvalino, carvalia
Student 3 Ali Tabar, sahraeia

April 12, 2021

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
March 16 2021	1.0	Added sections 1, 2, 3, 5
March 17 2021	1.1	Added sections 4, 6, 7
April 12 2021	1.2	Added Anticipated changes AC5,AC6,AC7 and updated the traceability accordingly. Changed the Grid module to Behaviour-Hiding Module. Changed the secret of the Start Game Module. All changes were done in red.

1 Introduction

Tetrileet is a redevelopment of LoveDaisy's Tetris found on GitHub, which is itself an adaptation to the classic video game known as *Tetris*. The original project was made in Python, but our remake is in JavaScript. Our main goal with this project is to improve upon the preexisting implementation, by adding or adjusting program aspects - all of these changes have been touched upon and explained in the Software Requirements Specification (SRS). The intentions behind these changes are to create a more accessible, user-friendly, and customized experience, enticing longer play times from users.
One of the most commonplace and widely accepted methods to software development is modular design. This entails the division of a software system into multiple different components called modules. We selected this method to ensure better manageability and maintainability for our project, while following the principle of information hiding. This will support maintainability in the future, especially if any changes are to be made to the software.
From this MG, we created the Module Interface Specification (MIS) to showcase the details regarding the implementation of every module.
The Module Guide's (MG) main purpose is to describe the design of the software from a modular perspective to readers. Readers can include:

- New members of AAA solutions who are joining the Tetrileet development team:
  The module guide is a valuable resource for a member of the development team to conceptually grasp the design of the overall project, and also to find information for specific modules of interest.

- Current maintainers of the project: Before making changes to the system, a maintainer should have a concise understanding of the software system's hierarchy. The module guide can help ensure this. Additionally, if changes are made to the system, other maintainers can become aware of this through the MG, as the maintainer who made the changes should update the module guide to reflect them.

- Current designers of the project: Designers can use the MG to ensure consistency, feasibility and flexibility in the system - modules must be consistent with one another to work, the deconstruction of the entire system should be convenient and intuitive, and the design as a whole should be flexible - allowing designers to tweak various aspects without difficulty.

The rest of the document is organized as follows:

- Section 2: a list of changes in the software requirements, both anticipated and unlikely

- Section 3: an overview of the module decomposition created from the likely changes

- Section 4: the connections between the software requirements and modules of the design

- Section 5: detailed descriptions of modules
- Section 6: traceability matrices which check completeness of the design against SRS requirements and show the relation between anticipated changes and modules
- Section 7: a diagram showcasing the use relations between modules

2 Anticipated and Unlikely Changes

This section lists possible changes to the system that may or may not be implemented at a later time. Changes which are anticipated to be implemented are listed in Section **??**, and unlikely changes to the project are listed in Section **??**.

2.1 Anticipated Changes

AC1: The web browser the player uses to run the application and how the scripting and styling modules account for those differences.

AC2: The size of the browser window the player uses and how the game fits inside it.

AC3: The internal implementation of existing modules.

AC4: The User Interface of the game and how objects are displayed.

AC5: Added different shapes to the game.

AC6: Some Modules may be combined if found to be more efficient in development.

AC7: Testing methods may be changed if new ones are found that are more efficient.

2.2 Unlikely Changes

UC1: Alternate input devices, such as a video game console controller or a touch pad.

UC2: Adding sound effects to the game, such as menu music or block placement sounds.

UC3: The user running the game on an alternate device, such as a phone or through a web browser on a video game console.

UC4: Implementing a multiplayer feature in the game.

3 Module Hierarchy

This section lists all existing modules and showcases the module design. Modules are displayed in their hierarchy in Table **??**.

M1: Game Window

M2: Start Game

M3: Pause Game

M4: End Game

M5: Grid

M6: Game Controller

M7: Shape

M8: BlockStack

M9: RowCheck

M10: BlockFall

Level 1	**Level 2**
Hardware-Hiding Module	Game Controller
Behaviour-Hiding Module	Game Window
Software Decision Module	Grid Shape Start Game End Game BlockStack Pause Game RowCheck BlockFall

Table 2: Module Hierarchy

4 Connection Between Requirements and Design

The design of Tetrileet is intended to satisfy the system requirements outlined in the SRS document. In line with the goal of developing a modular program, this section decomposes the functionalities of the game into smaller contained pieces and how they related to the requirements. These connections to the program requirements are listed in Table **??**.

5 Module Decomposition

The purpose of designing the system with modules is that it allows us to decompose the software into smaller, more manageable pieces of code. By doing this, our program will have increased maintainability, and will allow us to develop it more easily. According to the principle of information hiding, each of our modules will have one secret, which is hidden from the remaining modules. Having these secrets for the modules will allow us to change those sections without disrupting the rest of the system. Furthermore, each of the modules will also have a service which will describe their functionality without revealing their implementation.

5.1 Hardware Hiding Modules

M6 Game Controller

Secrets: The data structure and algorithm used to implement input/output of the virtual hardware - for example, a keyboard or screen.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: OS

5.2 Behaviour-Hiding Module

5.2.1 M1: Game Window

Secrets: The interface of the game that the player sees and interacts with.

Services: Displays the game board, score display, and buttons necessary for the player to interact with the game.

Implemented By: Tetrileet

5.2.2 M5: Grid

Secrets: Algorithm for designing and dealing with Shape movement and placement on the grid.

Services: Provides a game board that is associated with programming functionality to shape mechanics within the game.

Implemented By: Tetrileet

5.3 Software Decision Module

5.3.1 M7: Shape

Secrets: Methods for creating the different shapes of the game and their different orientations.

Services: Provides the shapes and rotation functionalities that are necessary for the game.

Implemented By: Tetrileet

5.3.2 M2: Start Game

Secrets: Algorithm that initializes game conditions.

Services: Resets all game conditions, and then grants the permission for shapes to start falling.

Implemented By: Tetrileet

5.3.3 M4: End Game

Secrets: Algorithm for detecting that a shape is outside of the Grid.

Services: Notifies certain other modules when the game ends.

Implemented By: Tetrileet

5.3.4 M8: BlockStack

Secrets: Algorithm for detecting when a shape has been placed on top of another shape.

Services: Provides detection for when a falling shape has landed and disables its movement.

Implemented By: Tetrileet

5.3.5 M3: Pause Game

Secrets: Algorithm that ties a user input to pausing game conditions including restricting movement of blocks.

Services: Gives functionality to a pause button by communicating with other modules to restrict game functionality based on input.

Implemented By: Tetrileet

5.3.6 M9: RowCheck

Secrets: Algorithm for detecting when a full row on the game grid has been occupied by shapes.

Services: Provides the implementation to detect when a player has filled in a row in the game. It will also erase the row and update the score board.

Implemented By: Tetrileet

5.3.7 M10: BlockFall

Secrets: Algorithm for Block movement down the game grid.

Services: Provides the implementation to allow for the shapes to fall down the game grid.

Implemented By: Tetrileet

6 Traceability Matrix

Two traceability matrices are shown: one displays the trace between the modules and the requirements, and the other displays the trace between the modules and the anticipated changes.

Req.	Modules
FR1	M1
FR2	M1
FR3	M2, M6
FR4	M2, M5
FR5	M4, M5, M10
FR6	M4, M5

Table 3: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M1
AC3	M2, M3, M4, M5, M6, M7, M8, M9, M10
AC4	M1, M2, M3, M4, M5, M7
AC5	M7, M5
AC6	M1, M2, M3, M4, M5, M7
AC7	M1, M2, M3, M4, M5, M7

Table 4: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

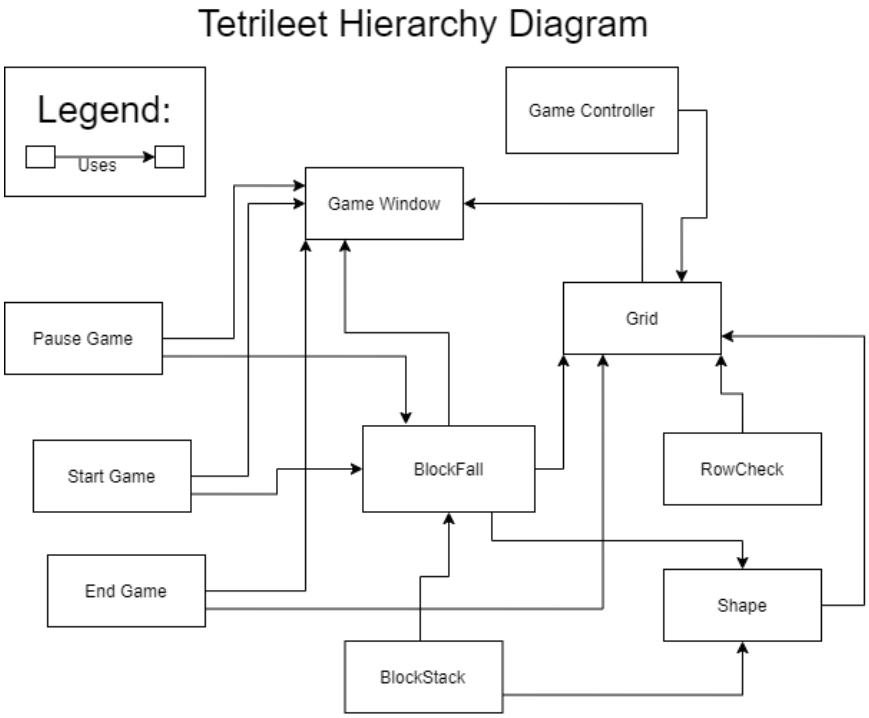

Figure 1: Use hierarchy among modules

8 Schedule

The Gantt chart of our project schedule is located in this repo, at 3xa3-L01-group-15/ProjectSchedule/group115.gan.