SE 3XA3: Design Document
PROJECT TETRIS

Team #11, Team Tetris
Daniel Agostinho agostd
Anthony Chang changa7
Divya Sridhar sridhad

November 13, 2016

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Developer(s)**	**Notes**
11/02/2016	0.0	Anthony Chang	Initial Draft
11/02/2016	0.0	Daniel Agostinho	Initial Draft
11/02/2016	0.0	Divya Sridhar	Initial Draft

1 Introduction

This is the Module Guide (MG) for PROJECT TETRIS. PROJECT TETRIS is a reimplementation of the classic arcade game Tetris. The user moves pieces called Tetraminos into a game board called a "well" and stacks them. When a row of the well is completely filled, that row is cleared and the user gets points. The purpose of the game is to obtain a high score by clearing as many rows as possible without letting the stack of pieces reach the top.

This project was designed with key programming principles in mind. These principles are: information hiding, modularity, and design for change. We divided the project into modules which are independent sections of code that have their own unique purpose. Modules support good programming principles such as information hiding. The modules abstract their information from other modules such that the other modules need not worry about that information. It does not concern them and allows for the modules to be modified independently and changes can be made without affecting the whole system.

The rest of the document is organized as follows. Section **??** lists the anticipated and unlikely changes of the software requirements. Section **??** summarizes the module decomposition that was constructed according to the likely changes. Section **??** specifies the connections between the software requirements and the modules. Section **??** gives a detailed description of the modules. Section **??** includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section **??** describes the use relation between modules.

2 Anticipated and Unlikely Changes

This section lists any possible changes to the system. According to the likeliness of the change, they are classified into two categories: anticipated changes are listed in Section **??**, and unlikely changes are listed in Section **??**.

2.1 Anticipated Changes

A design for change paradigm was utilized during our design process for PROJECT TETRIS. The following anticipated changes are design choices that are made to elements hidden within modules and thus are easy to alter without affecting other components of the project.

AC1: The specific hardware on which the software is running.

AC2: The specific Operating System on which the software interfaces with.

AC3: The language of Java (future versions/releases).

2.2 Unlikely Changes

The following design choices are incorporated in multiple components of the system. If a decision should later need to be changed, many different modules will need to be modified. Thus, these design decisions are unlikely to change.

UC1: Input/Output devices (Input: Mouse and/or Keyboard, Output: File and/or Screen).

UC2: The Algorithm for tetromino interaction (setting to well and/or clearing a row).

UC3: Goal of the software: Playable Tetris game.

3 Module Hierarchy

This section of the documents outlines the design of the modular decomposition. Table 2 shows how the modular hierarchy is decomposed by secrets.

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table **??**. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Hardware-Hiding Module

M2: Output Module

M3: Input Module

M4: Game Logic Module

M5: Game Graphics Module

...

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Output Module Input Module Game Graphics Module Input Control Module
Software Decision Module	Game Logic

Table 2: Module Hierarchy

4 Connection Between Requirements and Design

The system is designed such that the requirements specified in the Software Requirements Specification are satisfied. In the Design Document, the system is broken down into modules. The traceability matrix in Table **??** displays the connection between the requirements and the modules. Each requirements has a set of modules that are created to satisfy said requirement.

5 Module Decomposition

As this system was designed, one of the programming principles that was followed was information hiding. The modules are decomposed as to correctly abstract changes from other modules. This is called information hiding. In module decomposition, the Secrets are the

Modules are decomposed according to the principle of "information hiding" proposed by **?**. The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. Also indicate if the module will be implemented specifically for the software. Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented. Whether or not this module is implemented depends on the programming language selected.

5.1 Hardware Hiding Modules (M??)

Secrets: The data structure and algorithm used to implement the virtual hardware.

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 accept inputs and display outputs.

Implemented By: OS

5.2 Behaviour-Hiding Module

Secrets: The contents of the required behaviours.

Services: Includes programs that provide externally visible behaviour of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

Implemented By: –

5.2.1 Input Format Module (M??)

Secrets: The format and structure of the input data.

Services: Converts the input data into the data structure used by the input parameters module.

Implemented By: [Your Program Name Here]

5.2.2 Etc.

5.3 Software Decision Module

5.3.1 Exception Handling: File Not Found

Secrets: Algorithm used to find required files.

Services: Prompt user if the file cannot be found and to utilize a default.

Implemented By: PROJECT TETRIS

5.3.2 Event Listener

Secrets: Algorithm for accepting user inputs

Services: Listens for events triggered by the user and performs the desired action accordingly.

Implemented By: PROJECT TETRIS

5.3.3 Graphical User Interface

Secrets: Algorithms and rules pertaining to the display

Services: Interfaces with the OS to display the game

Implemented By: PROJECT TETRIS

6 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	Modules
R1	M??, M??, M??, M??
R2	M??, M??
R3	M??
R4	M??, M??
R5	M??, M??, M??, M??, M??, M??
R6	M??, M??, M??, M??, M??, M??
R7	M??, M??, M??, M??, M??
R8	M??, M??, M??, M??, M??
R9	M??
R10	M??, M??, M??
R11	M??, M??, M??, M??

Table 3: Trace Between Requirements and Modules

AC	Modules
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??

Table 4: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. **?** said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure **??** illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

Figure 1: Use hierarchy among modules