Blaze Brigade

- Module Guide -

SFWR ENG 3XA3 - Section L02
007 (Group 7)

Jeremy Klotz - klotzjj
Asad Mansoor - mansoa2
Thien Trandinh - trandit
Susan Yuen - yuens2

November 12, 2016

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
Nov 13, 2016	1.0	Completed Design Document

1 Introduction

1.1 Summary of Project

Blaze Brigade is a tactical simulation role-playing game that combines the strategic challenges as a form of interactive entertainment for its users. This turn-based game allows users to advance their units into enemy territory, participate in combat and strive to eliminate all of the opposing units. In adaption to the open-source freeware, Tactics Heroes, Blaze Brigade will incorporate new functional and design enhancements that may not be available on the existing open-source project. Such enhancements include the implementation of new features within the unit movement, combat and strategy aspects of the game as well as improved graphical representations of the menu menu and gameplay to improve the overall experience of the game.

1.2 Context of Module Guide

The system is fully devised and presents all of its functional and non-functional requirements in the Software Requirement Specification (SRS). As these requirements state the desirable properties of the system, the design documents will further evaluate on how these requirements are identified and achieved. The Module Guide (MG) will serve as a tool to decompose the following system into a modular structure adhering to the principle of information hiding. Upon reaching the finalized version of this document, the Module Guide can be distributed amongst various groups in order to learn and identify parts of the software that is being presented. These various groups are as follows:

- **Developers and maintainers:** Decomposing the system and documenting into the Module Guide will aid the developers and maintainers to understand the system-as-is and recognize what areas of the software are likely to be changed. In addition, a sense of the overall design will be structured and will be maintained in the following developments phases yet to come.
- **Designers:** In addition to the design pattern being documented, designers are able to determine whether the designs are constructed as initially specified. With the following anticipated changes to be happening, which areas of software is flexible and feasible to accommodate new design changes.
- **New recruits or outsourced resources:** The documentation will onboard the new recruits in familiarizing the overall structure of the implementation adhering to a specific design principle. This will reduce the downtime of debugging and have an advantage of multiple groups working on the system at once. Furthermore, if an external team were to implement the system or would like to carry out further improvements after the project timeline, this document will serve as an aid to determine the existing framework and how further implementation can take place.

In addition to the Module Guide, the Module Interface Specification (MIS) is also a product of the design documentation. The specification defines the syntax and semantics that are associated with the functions provided in the source code. Tools like Doxygen have been utilized to generate a set of documentation that will indicate the characteristics of the functions in terms of the corresponding inputs, outputs, assumptions, exceptions, state and environment variables. These characteristics will further aid in observing how the implementation is taken place and how the design constitutes from these functions.

1.3 Design Principle

The design principle taken into consideration revolves around the decomposition of the overall system into a modular structure of subsystems. These subsystems are observed in an abstract manner, hiding any details that may complicate the process. This act of information hiding and encapsulation ensures that each modules hides some design aspect from the rest of the system and analyzes which areas are expected to change. Hence, this document follows a design for change pattern and will be in the best interest throughout all of the subsystems presented in the system. For instance, the anticipated changes within the system would have been encapsulated in this process to ensure that any further changes to the design does not disrupt the main design interface of the system. As a principle for the decomposition into modular structure, the instance of low coupling is desired as the result is given as independent modules. In the same respect, high cohesion within the modules is highly desired since the elements of the module are strongly related to the module's characteristics. Therefore, this process is motivated around the concept of design for change as an exercise and validation to protect other modules of the system if any major changes occur in the overall design.

1.4 Outline of Module Guide

The Module Guide is organized in the given order. Section 2 lists all of the anticipated and unlikely changes that the software system might contain. Section 3 decomposes the system into a list of modules into the module hierarchy. Section 4 establishes the connection between the software requirements with the modules. Section 5 gives a detailed insight on how the modules have been decomposed with their corresponding descriptions. Section 6 includes three traceability matrices comparing the modules with the software requirements and anticipated changes as referenced earlier in Section 4. At last, section 7 pinpoints the use hierarchy between the modules initialized to establish connection between the independent modules.

1.5 Definitions, Acronyms, Abbreviations, Symbols

The following definitions and symbols are defined in Table 2 and will be referenced throughout the remainder of the Module Guide.

Symbol	Description
SRS	Software Requirements Specification document
MG	Module Guide document
MIS	Module Interface Specification document
Module	A decomposed subsystem of the overall software system
AC	Anticipated Changes
UC	Unlikely Changes
MVC	Model-View-Controller
FR	Functional Requirements
NFR	Non-Functional Requirements

Table 2: List of Definitions, Acronyms, Abbreviations and Symbols

2 Anticipated and Unlikely Changes

2.1 Anticipated Changes

The design decisions in this section are likely to change because they are hidden in modules. When these changes are made, they can be done easily and not affect other modules of the project.

AC1: All classes that implement the Class interface are likely to have their stats change for balancing reasons.

AC2: All weapons that implement the Weapon interface are likely to have their stats change for balancing reasons.

AC3: The getHitRate() and getCritRate() methods inside the DamageCalculations class are likely to change for balancing reasons.

AC4: All sprites from outside sources. It has been determined that the project should contain all original content.

2.2 Unlikely Changes

The following design decisions are unlikely to change because they affect many modules. Since they affect multiple modules, changing these decisions may result in multiple changes in the overall design of the project. Unless these changes are necessary, they will not occur.

UC1: Input/Output devices (Input: Mouse, Output: Updated Model and Screen).

UC2: The software implements the MVC (Model-View-Controller) architecture.

UC3: The Graph of nodes that represents the playable grid.

UC4: Nodes are identified by their x and y coordinates.

UC5: The path finding algorithm.

3 Module Hierarchy

M1: Hardware-Hiding Module

M2: Behaviour-Hiding Module

M3: Software Decision Module

M4: Menu Module

M5: Model Module

M6: GUI Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Menu Module, GUI Module
Software Decision Module	Model Module

Table 3: Module Hierarchy

Since Blaze-Brigade consists of purely software, M1 does not apply to the system. The software never interfaces with the hardware itself. The lowest level of interfacing with the software is the OS.

4 Connection Between Requirements and Design

The system is intended to satisfy all of the functional and nonfunctional requirements that were initially specified in the SRS. In this section, the system is decomposed into modules and assess the connections between the decomposed requirements with the corresponding requirements. These are shown in the Table 4 and Table 5 under Section 6.

Most of the requirements can be categorized as one of the modules provided in the Module Hierarchy. For instance, the Menu Model extends through all of the requirements that initiate the menu option in one way or another. The model module encapsulate the model classes that represent the structure of the source code. The GUI module is primarily what users get to see as a final product which includes various parameters within the game. The design decisions that are needed to accommodate these requirements heavily rely on the main function criteria that the system holds. For instance, the appearance requirements specify the look and feel that the user should be expecting from the product and heavily focuses on the menu and GUI aspect. These modules initialized will cover those aspects and model this case scenario is such a manner that each module or submodules will be independent and protected if there is design change in the other part of the system.

5 Module Decomposition

Module Decomposition summary TODO

5.1 System Architecture

5.2 Underlaying Architecture

5.3 Leaf-level Decomposition

5.4 Summary of Leaf Modules

5.4.1 Hardware Hiding Modules (M1)

Secrets: The algorithms and format structures used to provide an interface between hardware and software.

Services: This module provides an interface for users to interact with the software. The module will convert the raw input data from the mouse into data that can be used by controller to update the current game state. The view will also be implemented through this, allowing for users to correctly interact with the software.

Implemented By: Mouse, MouseHandler, M2

5.4.2 Behaviour-Hiding Module(M2)

Secrets: The behavioural process of the software.

Services: This module functions as the controller in MVC, and handles all the software decision making of Blaze Brigade. This includes all visible

behavior of the system specified in the SRS. Hence any changes to the SRS will hereby result in modifications to this module.

Implemented By: Computer, game.cs

5.4.3 Software Decision Module(M3)

Secrets: The design decisions that determine *how* the software updates.

Services: This module extends the Model Module, and stores the state of the overall game, and contains the state of how everything in the game should currently behave. These results determine what is displayed in the GUI Module (view).

Implemented By: M2

5.4.4 GUI Module (M6)

Secrets: How and when what is displayed.

Services: This module is the main View in MVC, and displays data to users in the form of graphics according to what the current game state is.

Implemented By: Draw methods, M**??**, Buttons.cs

5.4.5 Menu Module (M4)

Secrets: The navigational structure for different menu options.

Services: This module handles the main menu layout, navigation and controls.

Implemented By: M5

5.4.6 Model Module (M5)

Secrets: The design decisions that implement the structure of the software.

Services: This module is the main Model in MVC, and contains most the structure of the game. Most of the elements in this module are simply data, with most methods simply being a C# property (combination of getter and setter).

Implemented By: Game.cs, Graph.cs, Node.cs, Unit.cs, Weapon.cs, Player.cs

6 Traceability Matrix

This section show three traceability matrices outlining the comparison between the modules with either the functional requirements, non-functional requirements and anticipated changes.

Req.	**Modules**
FR1	M1, M4, M6
FR2	M2, M5
FR3	M5, M6
FR4	M3, M5, M6
FR5	M5

Table 4: Trace Between Functional Requirements and Modules

Req.	**Modules**
NFR1	M1, M2, M6
NFR2	M3, M5
NFR3	M3, M6
NFR4	M1, M3
NFR5	M3, M5, M6
NFR6	M1, M2
NFR7	M6
NFR8	M1, M3

Table 5: Trace Between Non-Functional Requirements and Modules

AC	**Modules**
AC1	M2, M3, M5
AC2	M4, M5
AC3	M5
AC4	M6

Table 6: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

In this section of the Module Guide, the system has already been decomposed into the desired modular structure and characterized by determining the connection between the initial requirements and anticipated changes to those modules. The uses hierarchy presented in this section compares the independent modules

with each other to find the common grounds on which modules uses the instance of another. This practice ensures the correctness of the program when it comes to testing as well as a reference to the integration procedure if the design of the system experiences a major change. For instance, if we model a scenario where Module A uses Module B, then all of the parameters that rely on both module have to be specified to ensure that the valid output of Module B can be efficiently used in Module A to proceed with the execution of the design. This hierarchy also represents the testing environment, as Module A and Module B would first be tested independently then the correlation between the shared parameters of both modules. In theory, Module A would present a similar module entity as it is categorized in the higher level of the hierarchy and relies on Module B on the lower level to provide some set of specified work assignment. The following user hierarchy of the current system is shown in Figure 1. Notice how the representation is described as a Directed Acyclic Graph, a finite set of modules in phase from the higher levels to the lower levels of the hierarchy with no apparent recurring cycles. Hence the design pattern ensure that the decomposition has been done correctly and the definition of the design can now be distributed amongst the various groups that relate to the context of the Module Guide.

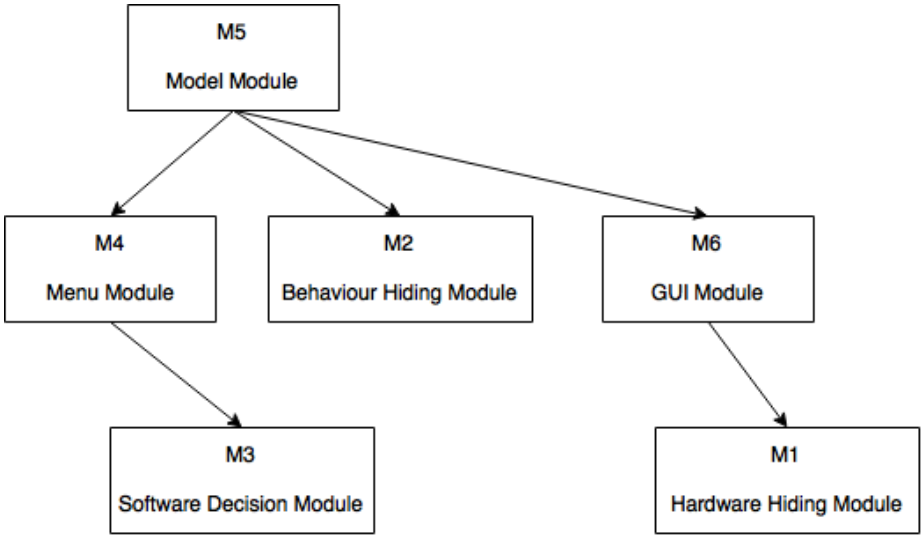

Figure 1: Use hierarchy among modules