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 13, 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 main 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 Requirements Specification (SRS), stating desirable properties of the system. Meanwhile, the Design Document 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 Maintenance:** System decomposition and the Module Guide will aid the developers and maintenance team in understanding the system-as-is and to recognize which 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. Along with the upcoming anticipated changes to be happening, designers can further determine which areas of software are flexible and feasible to accommodate new design changes.
- **New Recruits or Outsourced Resources:** The documentation will aid with the onboarding process of 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 simultaneously. 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 has 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 module 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 each 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, and further states 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 categorized as anticipated due to being hidden in modules. When these changes are made, they can be done easily and will 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 are likely to change. 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	Game State 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 non-functional requirements that were initially specified in the SRS. In this section, the system is decomposed into modules and connections are assessed 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, including 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 in such a manner that each module or sub-module will be independent and protected if there is design change in the other part of the system.

5 Module Decomposition

The goal of this section is to provide a detailed description of how the system operates. In order to do this, a module decomposition is necessary. The modules in this decomposition are not classes directly from the software code, but instead are a collection of sub-modules that complete an abstract concept. The sub-modules may be classes directly from the software, or be broken down until these classes are reached. This section will also include a pair of diagrams, one that explains the project's chosen architecture, and another that demonstrates how each module fits into said architecture.

5.1 System Architecture

The following definitions explain all important terms used to describe the system architecture.

MVC $\rightarrow$ The specific system architecture for this software. MVC stands for Model-View-Controller. The user interacts with the controller, which manipulates the model. The view then updates based on the model. From here, the user sees the result of their interaction through the view.

Model $\rightarrow$ The central point of the system architecture. The model contains all data, logic and rules of the software. Whatever the view displays to the user is based on the model.

View $\rightarrow$ The part of the system that displays information to the user. This is where the user sees all relevant information.

Controller $\rightarrow$ The part of the system that the user manipulates. The controller is what updates all information stored in the model.

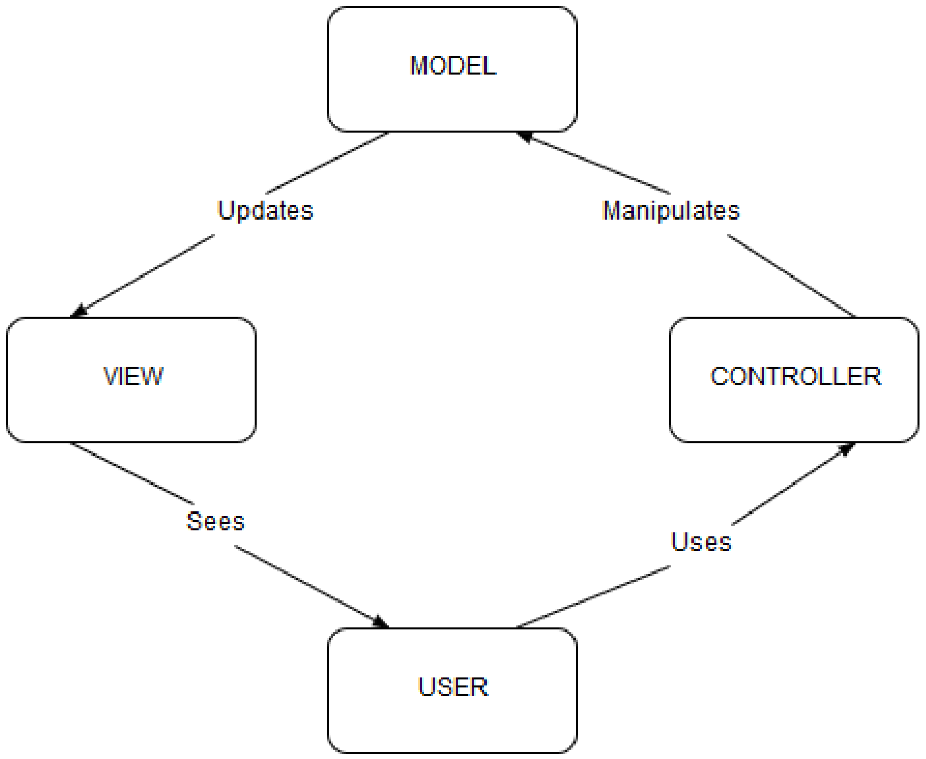

Figure 1: The MVC Architecture of the Software System

5.2 Underlaying Architecture

Now that the general idea of the system has been explained, this section will take the level of abstraction one step further. This level of abstraction demonstrates the module decomposition of the system. Arrows in the diagram represent sub-modules that help reach the next module in the system. Note that this is not a uses diagram, nor is it intended to be.

Hardware Hiding Module: This module (and its sub-modules) provide 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 the controller to update the current game state. The view will also be implemented through this, allowing for users to correctly interact with the software.

Behaviour-Hiding Module: 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.

Software Decision Module: This module extends the Model Module, 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).

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

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

Model Module: This module is the main Model in MVC, and contains most of 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 getters and setters).

The following is a diagram that depicts this level of abstraction of the software system. Note that red arrows indicate the path of the software, whereas the black arrows indicated extended modules.

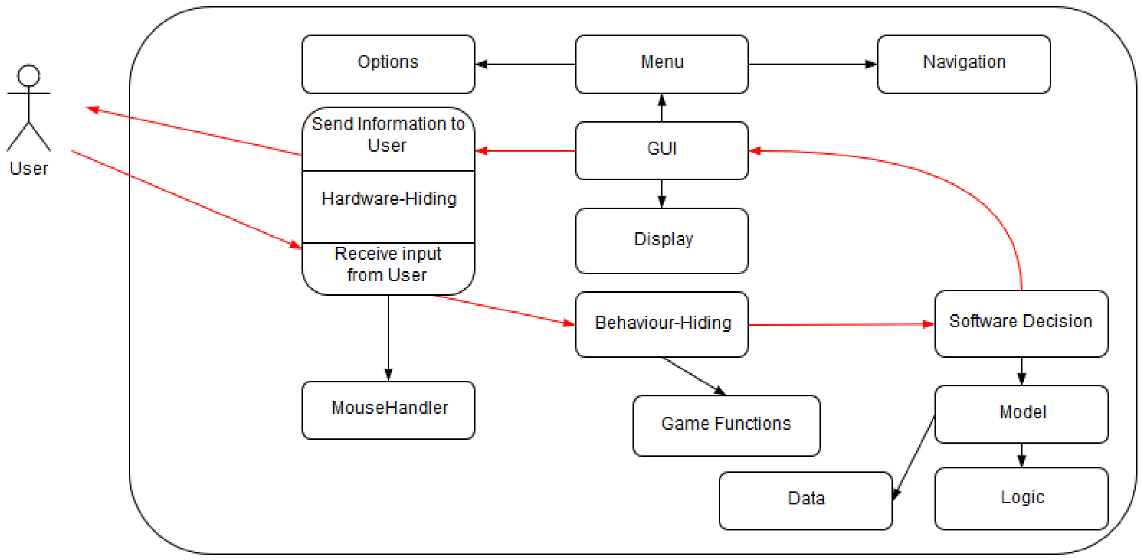

Figure 2: First Level Decomposition of Blaze Brigade

5.3 Leaf-level Decomposition

5.3.1 MouseHandler

This module is further broken down into Mouse Position, and Update Game.

- **Mouse Position:** Gathers the location of the last mouse click and prepares the information for the update game module.
- **Update Game:** Updates the GUI of the software based on the prepared information from the mouse position module.

5.3.2 Behaviour-Hiding Module

This module is further broken down into the Game Functions module.

- **Game Functions:** All of the algorithms and processes that run during the transition of one game state to another.

5.3.3 GUI Module

The GUI module is further broken down into Menu, and Display.

- **Menu:** The tool that appears on the GUI in which the user interacts with for navigation. This module is still to be broken down. See section 5.3.4 for this further breakdown.
- **Display:** The visual representation of all user-relevant data.

5.3.4 Menu Module

This module is broken down into Options, and Navigation.

- **Options:** The set of all decisions the user can make while in the menu.
- **Navigation:** How the user is navigated through the software based on what option they select from the menu.

5.3.5 Model Module

This module is further broken down into Data, and Logic.

- **Data:** All information that is relevant to the units, players, and map of the game.
- **Logic:** All rules, and logic that apply to the game. All of the rules and logic are checked based on the current state of the game.

5.4 Summary of Leaf Modules

5.4.1 Mouse Position

Secrets: The algorithm used to determine where the mouse last clicked.

Services: This module collects the information needed to update the view.

Implemented by: MouseHandler.cs, Mouse

5.4.2 Update Game

Secrets: Behavioural process of how the view is updated.

Services: This module updates the GUI so that the user can see how their decisions changed the state of the game.

Implemented by: MouseHandler.cs, game.cs

5.4.3 Game Functions

Secrets: The algorithms that execute during the transition of one game state to another.

Services: This module moves the software from its current state to the next state if possible.

Implemented by: GameFunction.cs

5.4.4 Display

Secrets: How and when what is displayed.

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

Implemented by: Draw methods, Buttons.cs

5.4.5 Options

Secrets: The structure for different menu options.

Services: This module handles the main menu layout, based on the current game state.

Implemented by: Button.cs, MainMenu.cs, howToPlay.cs, howToPlay2.cs howToPlay.cs

5.4.6 Navigation

Secrets: The design decisions that implement the software navigation.

Services: This module guides the user throughout the application. All decisions are made through the navigation system.

Implemented by: Game.cs, Button.cs, mouseHandler.cs

5.4.7 Data

Secrets: The structure of how unit, map, weapon and player information is stored.

Services: Holds all of the unit, map, weapon and player values. These values are updated when notified to.

Implemented by: Graph.cs, Node.cs, Unit.cs, Weapon.cs, Player.cs, gameState.cs

5.4.8 Logic

Secrets: The algorithms that confirm game rules and logic.

Services: Provides structure to the game. The logic of the system helps determine which state the software will move to next.

Implemented by: Game.cs, damageCalculation.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.

Requirement	**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

Requirement	**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 modules 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, and 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 3. 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 ensures 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.

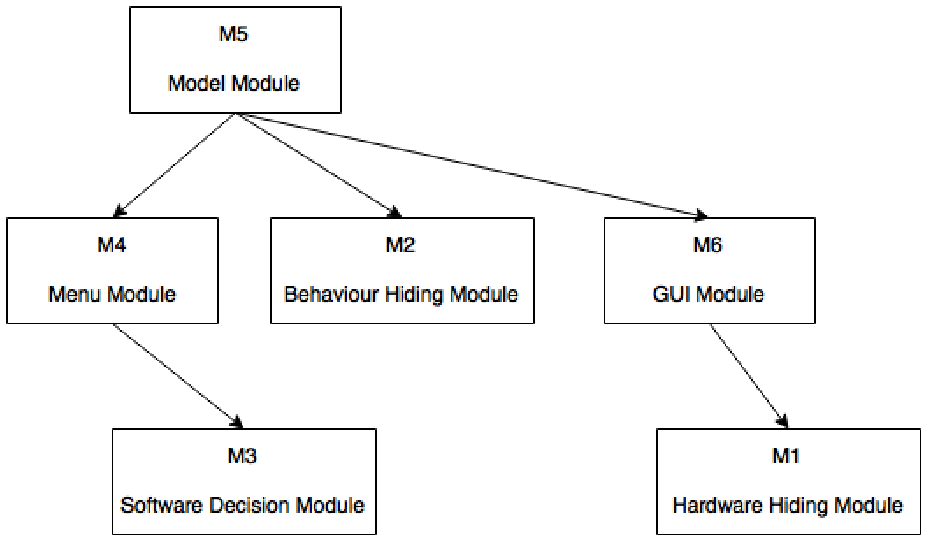

Figure 3: Use Hierarchy Among Modules