

Blaze Brigade

- Module Guide -

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

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

December 8, 2016

Contents

1	Introduction	4
1.1	Summary of Project	4
1.2	Context of Module Guide	4
1.3	Design Principle	5
1.4	Outline of Module Guide	5
1.5	Definitions, Acronyms, Abbreviations, Symbols	5
2	Anticipated and Unlikely Changes	6
2.1	Anticipated Changes	6
2.2	Unlikely Changes	6
3	Module Hierarchy	7
4	Connection Between Requirements and Design	7
5	Module Decomposition	8
5.1	System Architecture	8
5.2	Underlying Architecture	9
5.3	Leaf-level Decomposition	10
5.3.1	Map Module	10
5.3.2	Unit Module	10
5.3.3	Weapon Module	10
5.3.4	Menu Module	11
5.4	Summary of Leaf Modules	11
5.4.1	Hardware Hiding Module	11
5.4.2	DamageCalculations (Controller)	11
5.4.3	Game (Controller)	11
5.4.4	GameFunction (Controller)	12
5.4.5	MouseHandler (Controller)	12
5.4.6	Node (Model)	12
5.4.7	Graph (Model)	12
5.4.8	GameState (Model)	12
5.4.9	Player (Model)	12
5.4.10	Unit Module (Model)	13
5.4.11	Buttons (Model)	13
5.4.12	Weapon Module (Model)	13
5.4.13	Main Menu (View)	13
5.4.14	HowToPlay (View)	13
5.4.15	Animation (View)	14
5.4.16	DrawClass (View)	14
5.4.17	Sound (View)	14
5.4.18	Camera (View)	14
6	Traceability Matrix	14

7	Use Hierarchy Between Modules	15
----------	--------------------------------------	-----------

List of Tables

1	Revision History	3
2	List of Definitions, Acronyms, Abbreviations and Symbols	6
3	Module Hierarchy	7
4	Trace Between Functional Requirements and Modules	14
5	Trace Between Non-Functional Requirements and Modules	15
6	Trace Between Anticipated Changes and Modules	15

List of Figures

1	The MVC Architecture of the Software System	9
2	Use Hierarchy among Modules	16

Table 1: **Revision History**

Date	Version	Notes
Nov 13, 2016	1.0	Completed Design Document
Dec 07, 2016	2.0	Revised section 3-7 with the modification of new modules

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: Model Module

M3: View Module

M4: Controller Module

Level 1	Level 2
Hardware-Hiding Module	C# XNA Game Studio
Model Module	Map Module, Unit Module, Weapon Module, Game State, Player
View Module	Menu Module, Animation, Camera, DrawClass, Sound
Controller Module	Game, GameFunction, MouseHandler, Damage Calculation

Table 3: Module Hierarchy

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.

All 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 encapsulates the model classes that represent the structure of the source code. The View module is primarily what users get to see as a final product, including the various parameters within the game. Finally, all game logic is handled inside the Controller module. One such example to prove the above would be appearance requirements. Appearance requirements specify the look and feel that the user should be expecting from the product and heavily focuses

on the view module. 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 → 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 → 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 → The part of the system that displays information to the user. This is where the user sees all relevant information.

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

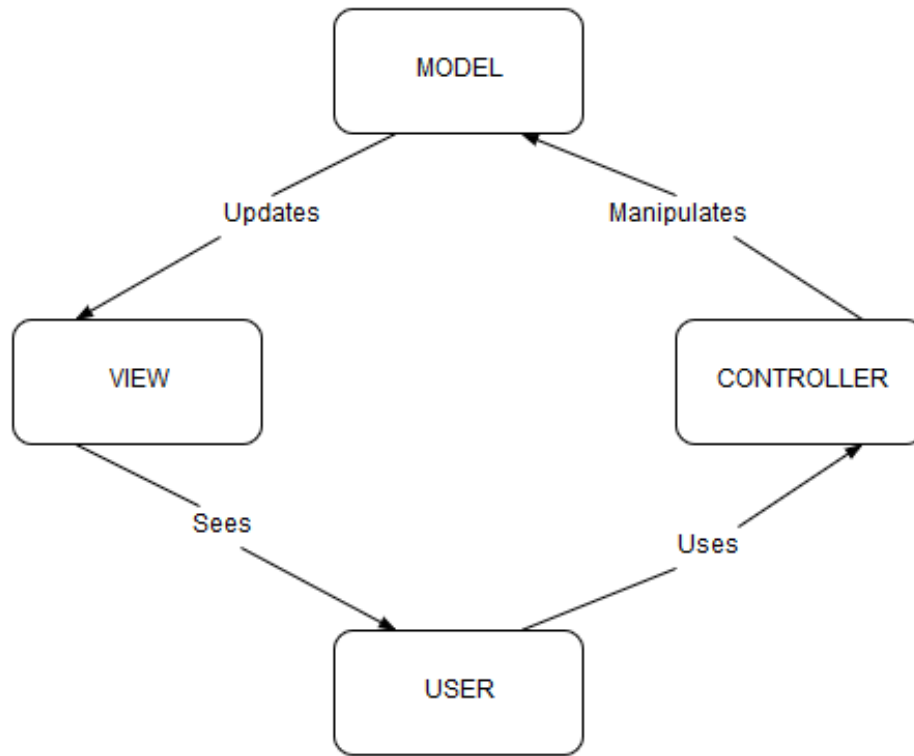


Figure 1: The MVC Architecture of the Software System

5.2 Underlying 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 consists of C# XNA Game Studio, and provides an interface for users to interact with the software. This 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.

Model Module: This model contains the structure of the game. Most of the elements in this module are simply data such as unit stats and position. The majority of the methods in this module are simple C# properties (special C# method that functions as both a getter and setter).

View Module: This module contains all view related features. This module displays data to the users in the form of graphics representing the current game state.

Controller Module: This module handles all the software decision making of Blaze Brigade, including how the entirety of the game functions as a whole. 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.

5.3 Leaf-level Decomposition

Another level of decomposition is yet required for Blaze Brigade. This will include the decomposition of the smaller modules listed in section 3 that are a grouped under model, view and controller modules.

5.3.1 Map Module

This module is further broken down into Graph and Node.

- **Graph:** This is the graph structure used to represent the game map grid and is comprised of Nodes.
- **Node:** The structure that represents each tile on the game map grid.

5.3.2 Unit Module

This module is further broken down into Warrior, Mage and Archer.

- **Warrior:** One of the 3 playable unit classes with high physical attack and defense.
- **Archer:** One of the 3 playable unit classes with high skill and speed.
- **Mage:** One of the 3 playable unit classes with high magical attack and defense.

5.3.3 Weapon Module

This module is further broken down into Bronze Sword, Iron Sword, Short Bow, Long Bow, Fireball, and Fireblast.

- **Bronze Sword:** A warrior's weapon.
- **Iron Sword:** A warrior's weapon.
- **Short Bow:** An archer's weapon.
- **Long Bow:** An archer's weapon.
- **Fireball:** A mage's weapon.
- **Fireblast:** A mage's weapon.

5.3.4 Menu Module

The GUI module is further broken down into Main Menu, HowToPlay, HowToPlay2, and HowToPlay3.

- **Main Menu:** The Main menu upon start-up that allows for navigation to the rest of the game.
- **HowToPlay:** The first page of instructions on how to play.
- **HowToPlay2:** The second page of instructions on how to play.
- **HowToPlay3:** The third page of instructions on how to play.

5.4 Summary of Leaf Modules

Unit Module and Weapon Module are treated as the smallest decomposition to a leaf module for convenience as every unit in Unit Module and every weapon in Weapon module are almost identical outside of game parameters such as movement and damage dealt. HowToPlay, HowToPlay2 and HowToPlay3 are also treated as 1 module as they are simply different pages on how to play.

5.4.1 Hardware Hiding Module

Secrets: The data structure and processes required to implement virtual hardware.

Services: This module provides an interface between the hardware and software.

Implemented by: OS

5.4.2 DamageCalculations (Controller)

Secrets: The math behind how damage is calculated.

Services: The damage dealt from one unit to another.

Implemented by: GameFunction

5.4.3 Game (Controller)

Secrets: How the game functions together.

Services: The game provides the "brain" for how the Blaze Brigade functions as a whole.

Implemented by: Hardware Hiding Module

5.4.4 GameFunction (Controller)

Secrets: The decision making for how the game functions.

Services: GameFunction provides the main decision making for updating the current game.

Implemented by: MouseHandler

5.4.5 MouseHandler (Controller)

Secrets: How player input is converted into information used to update the game.

Services: This handles all user input, then directly calls different parts of GameFunction depending on input to process it.

Implemented by: Game.cs

5.4.6 Node (Model)

Secrets: The unit stored on the node.

Services: Node represents each tile in the Game Map.

Implemented by: Graph, GameFunction

5.4.7 Graph (Model)

Secrets: Where each node is stored in graph.

Services: The graph representing the Game Map.

Implemented by: Game, GameFunction

5.4.8 GameState (Model)

Secrets: The current state of all aspects of the game.

Services: GameState stores the current state of all the different aspects of the game.

Implemented by: GameFunction

5.4.9 Player (Model)

Secrets: The units stored in player.

Services: Player represents each player of the game.

Implemented by: Game, GameFunction

5.4.10 Unit Module (Model)

Secrets: The parameters and information of each unit.

Services: The unit stores all relevant information of the unit including stats, position, and textures (Only the texture itself is stored. All visuals are drawn in View Module). Warrior, Mage, and Archer all fall under Unit Module.

Implemented by: Player

5.4.11 Buttons (Model)

Secrets: The button information for each unit.

Services: This stores all the information of the buttons for the unit. This includes button location, whether or not the button has a weapon stored to it, whether or not it is active, and the button texture (Only the texture itself is stored. All visuals are drawn in View Module).

Implemented by: Unit

5.4.12 Weapon Module (Model)

Secrets: The parameters and information of each Weapon.

Services: The weapon stores the stats of weapon. Bronze Sword, Iron Sword, Short Bow, Long Bow, Fireball, and Fireblast all fall under Weapon Module.

Implemented by: Button

5.4.13 Main Menu (View)

Secrets: The controls to navigate to the actual game.

Services: The Main Menu allows the player to navigate to instructions on how to play, exit game, or start the game.

Implemented by: Game

5.4.14 HowToPlay (View)

Secrets: Navigation through instructions and back.

Services: How To Play provides player instructions on how to play the game, as well as provides navigation to and from Main Menu.

Implemented by: Game, Main Menu

5.4.15 Animation (View)

Secrets: How units are animated.

Services: Animation provides the unit animation during moving and attacking.

Implemented by: GameFunction

5.4.16 DrawClass (View)

Secrets: How all visuals are drawn to screen.

Services: DrawClass provides the conversion of data into appropriate visuals that the player can see. This includes Game Map, Units, and GUI.

Implemented by: Game

5.4.17 Sound (View)

Secrets: How sounds are played.

Services: Sounds provides the music and sounds that play during the game.

Implemented by: GameFunction

5.4.18 Camera (View)

Secrets: Determining the current location of the game screen that the player sees.

Services: Camera provides the currently viewable section of the game map that the player sees.

Implemented by: GameFunction

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, M2, M3
FR2	M2, M3
FR3	M2, M4
FR4	M2, M4
FR5	M4

Table 4: Trace Between Functional Requirements and Modules

Requirement	Modules
NFR1	M1, M2, M3
NFR2	M2, M3
NFR3	M1, M2
NFR4	M1, M2
NFR5	M2, M3, M4
NFR6	M3
NFR7	M3
NFR8	M1, M3

Table 5: Trace Between Non-Functional Requirements and Modules

AC	Modules
AC1	M2, M4
AC2	M2, M4
AC3	M4
AC4	M3

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 2. 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.

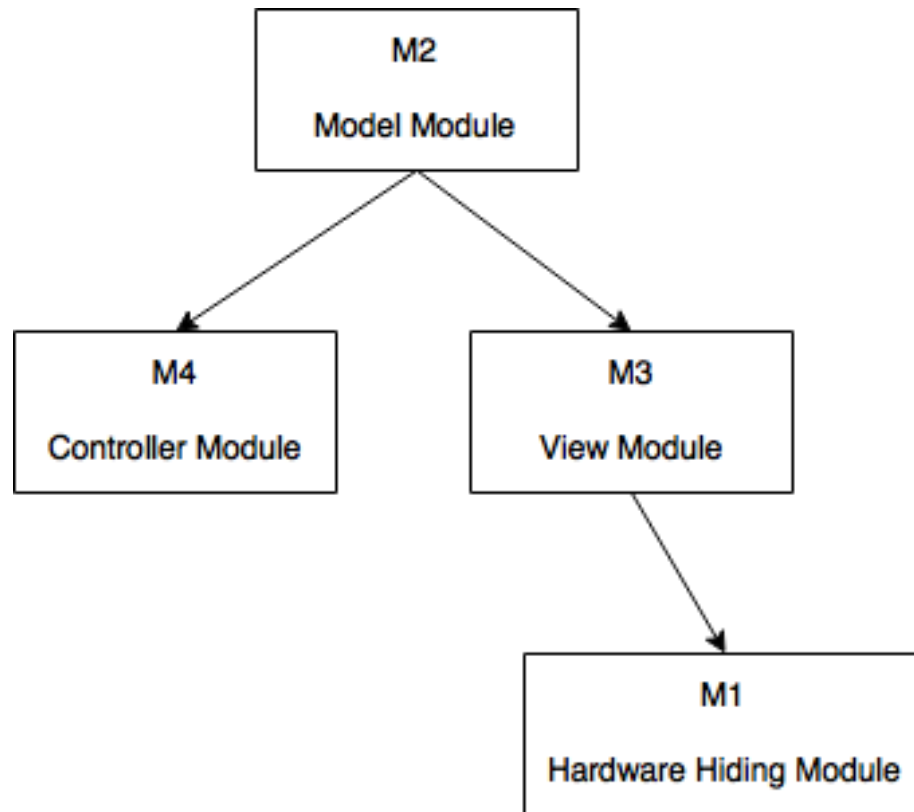


Figure 2: Use Hierarchy among Modules