SE 3XA3: Module Guide
Ultimate Tic Tac Toe

Team 3, Ultimate Tic Tac Toe
Kunal Shah - shahk24
Pareek Ravi - ravip2

December 6, 2016

Contents

1 Introduction **3**

2 Anticipated and Unlikely Changes **4**
2.1 Anticipated Changes 4
2.2 Unlikely Changes 5

3 Module Hierarchy **5**

4 Connection Between Requirements and Design **6**

5 Module Decomposition **6**
5.1 Hardware Hiding Modules 7
5.1.1 User Interface Module (M1) 7
5.1.2 Click Listener (M2) 7
5.2 Behaviour-Hiding Module 7
5.2.1 Player Turn Module (M4) 7
5.2.2 Move Module (M3) 8
5.2.3 Active Region Module (M5) 8
5.2.4 Win Check Module (M6) 8
5.3 Software Decision Module 8

6 Traceability Matrix **8**

7 Use Hierarchy Between Modules **9**

List of Tables

1 **Revision History** 2
2 Module Hierarchy 6
3 Trace Between Requirements and Modules 9
4 Trace Between Anticipated Changes and Modules 9

List of Figures

1 Use hierarchy among modules 10

Table 1: **Revision History**

Date	**Version**	**Notes**
November 8	0.0	Initial set up
November 11	0.2	Introduction
November 11	0.4	Added Content to all sections
November 12	0.5	Anticipated and Unlikely Changes
November 12	0.6	Completed Module Decomposition and Traceability Matrix
November 13	1.0	Rev0 Submission
December 1	1.1	Added UseHierarchy_Diagram.pdf to repo
December 5	1.5	Updating document based on comments from Christopher
December 5	1.6	Updated use hierarchy between modules figure
December 7	2	Rev1 Submission

Abstract

Ultimate Tic Tac Toe is a variation on the classic game of Tic Tac Toe. It is simply multiple games of Tic Tac Toe running simultaneously to make a classic game that often ends in a draw have an exciting ending.

The primary purpose of the Module Guide (MG) is to describe, justify, and contextualize the module decomposition of the system. The document also provides a conceptual view of each module and its context in the broader system. Before the MG is written, the Software Requirements Specification (SRS) document should be completed. The SRS describes requirements for the software system that the MG should relate to the implemented design. By completing the SRS beforehand, it is more likely that the selected system design will comply with all compulsory requirements and the chosen module decomposition is appropriately structured.

1 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules layed out by Parnas et al. (1984), as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is used in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (Parnas et al.,

1984). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 2 lists the anticipated and unlikely changes of the software requirements. Section 3 summarizes the module decomposition that was constructed according to the likely changes. Section 4 specifies the connections between the software requirements and the modules. Section 5 gives a detailed description of the modules. Section 6 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 7 describes the use relation between modules.

2 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 2.1, and unlikely changes are listed in Section 2.2.

2.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will

only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: The hardware on which the software is running.

AC2: Draw / tie logic handling

AC3: Online muli-player support

AC4: Game user interface layout (responsiveness based on device screen size)

AC5: The language in which the game rules is presented.

2.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: Input devices (touch and mouse)

UC2: Win and loss logic

UC3: Data structures holding game state information

3 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 2. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: User Interface Module

M2 : Click Listener

M3 : Move module

M4: Player turn module

M5: Active region module

M6: Win check module

Level 1	**Level 2**
Hardware-Hiding Module	User Interface Module Click Listener
Behaviour-Hiding Module	Player turn module Move module Active region module Win check module
Software Decision Module	

Table 2: Module Hierarchy

4 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 3.

5 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984). 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.

5.1 Hardware Hiding Modules

5.1.1 User Interface Module (M1)

Secrets: No secrets, public game board

Services: This module provides the game board view. Publicly viewable using inspect element. Contains table scaffolding for for how the game is setup

Implemented By: HTML

5.1.2 Click Listener (M2)

Secrets: Location of click

Services: This module provides the interface between the model and view. So, when the user clicks on the view the script can accept inputs from User and display outputs and to screen.

Implemented By: HTML user interface and Javascript

5.2 Behaviour-Hiding Module

5.2.1 Player Turn Module (M4)

Secrets: current player state

Services: Module randomly picks symbol for first move and provides externally visible player symbol to the display to the system as specified in the software requirements specification (SRS) documents.

Implemented By: Player Turn Module

5.2.2 Move Module (M3)

Secrets: The format and structure of the input data.

Services: This module is responsible for translating user input and sending to the game logic data structure.

Implemented By: Move Module

5.2.3 Active Region Module (M5)

Secrets: Next Active region

Services: This module is responsible for updating the board to the new active region based on previous player interactions after receiving information from the move module

Implemented By: Active Region Module

5.2.4 Win Check Module (M6)

Secrets: Current game state

Services: This module is responsible for taking care of checking if the player has won the game, and updating the board after receiving information from the move module. This means if player has won or drawn an inner or game board.

Implemented By: Win Check Module

5.3 Software Decision Module

6 Traceability Matrix

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

Req.	Modules
FR1	M1
FR2	M1, M2,M3, M5
FR3	M5, M6
FR4	M1, M6
FR5	M6
FR6	M1, M4

Table 3: Trace Between Requirements and Modules

AC	Modules
AC1	M1,M2
AC2	M6
AC3	M4
AC4	M1
AC5	M1

Table 4: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1978) said of two modules 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 1 illustrates the use relation between the modules. It can be seen that the each level of the hierarchy offers a testable and usable subset of the system.

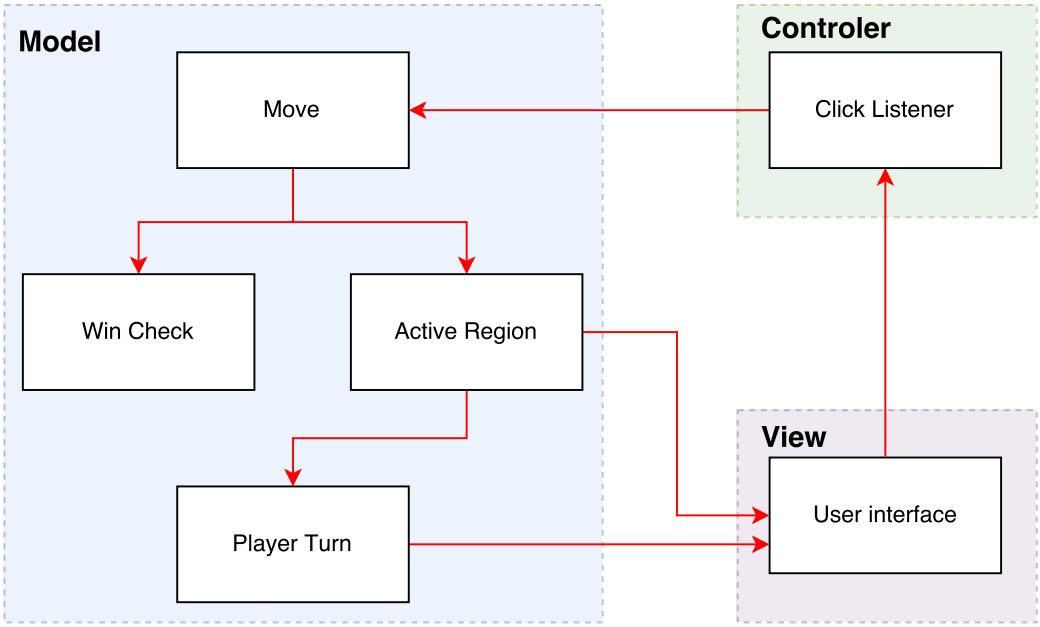

Figure 1: Use hierarchy among modules

References

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.