

SE 3XA3: Module Guide

XA3: PDWTAK

Team 9, Just Because You're Correct Doesn't Mean You're Right

Connor Simpson, simpsc

Junhao Wang, wangjh2

Jim Wu, wuzz

November 13, 2016

Contents

| | | |
|----------|---|----------|
| 1 | Introduction | 1 |
| 1.1 | Overview | 1 |
| 1.2 | Context | 1 |
| 1.3 | Design Principles | 1 |
| 1.4 | Document Readers | 2 |
| 1.5 | Document Structure | 2 |
| 2 | Anticipated and Unlikely Changes | 2 |
| 2.1 | Anticipated Changes | 3 |
| 2.2 | Unlikely Changes | 3 |
| 3 | Module Hierarchy | 3 |
| 4 | Connection Between Requirements and Design | 4 |
| 5 | Module Decomposition | 5 |
| 5.1 | Hardware Hiding Modules (M1) | 5 |
| 5.2 | Behaviour-Hiding Module | 5 |
| 5.2.1 | Game Mode Module (M2) | 5 |
| 5.2.2 | Character Module (M3) | 6 |
| 5.2.3 | Map Module (M4) | 6 |
| 5.2.4 | User Input Module (M5) | 6 |
| 5.2.5 | User Interface Module (M6) | 6 |
| 5.2.6 | Menu Module (M7) | 6 |
| 5.2.7 | Camera Module (M8) | 6 |
| 5.3 | Software Decision Module | 7 |
| 5.3.1 | Artificial Intelligence Module (M9) | 7 |
| 5.3.2 | Grid Module (M10) | 7 |
| 5.3.3 | Pathfinding Module (M11) | 7 |
| 6 | Traceability Matrix | 7 |
| 7 | Use Hierarchy Between Modules | 9 |

List of Tables

| | | |
|---|---|----|
| 1 | Revision History | ii |
| 2 | Module Hierarchy | 4 |
| 3 | Trace Between Functional Requirements and Modules | 8 |
| 4 | Trace Between Anticipated Changes and Modules | 9 |

List of Figures

| | | |
|---|---------------------------------------|---|
| 1 | Use hierarchy among modules | 9 |
|---|---------------------------------------|---|

Table 1: **Revision History**

| Date | Version | Notes |
|--------|---------|------------|
| Nov 13 | 1.0 | Revision 1 |

1 Introduction

1.1 Overview

The XA3-PDWTAK project is a reimplementaion of an open-source recreation of the classic 1990s X-COM games. It runs off the Unreal engine and is meant to be an entertaining experience of a classic game with a more modern User Interface.

1.2 Context

The design of decomposing a system into modules is a common and accepted approach to the development of software. A module is a work assignment for a programmer for the purpose of information hiding. This style supports flexible designs that can change and evolve because the information each module hides represents likely future changes. Flexible designs for software development are valuable as modifications to design are frequent especially during initial development as solutions are explored.

The Module Guide uses the SRS to give context to some of the decisions and for a more detailed outline of modules, the information can be located in the MIS.

1.3 Design Principles

The project uses Design Principles to outline the reasoning for it's decomposition. These principles are as follows:

- Information Hiding
- Encapsulation
- Separation of Concerns
- Single Responsibility

The principle of Information Hiding is that each module hides information about itself from the rest of the system. The principle of Encapsulation is that changeable information is in the implementation of the module, but the module interface should not change when the implementation changes. The functionality of the module stays the same even if the implementation for this functionality changes. Separation of concerns is that each module has a feature with as little overlap in functionality as possible. The important factor of Separation of Concerns is minimization of interaction points to achieve high cohesion and low coupling. Single Responsibility is the principle that each module should be responsible for only a specific feature or functionality, or the merger for cohesive functionality.

1.4 Document Readers

The Module Guide (MG) outlines the module 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 the 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.

1.5 Document Structure

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: Game Balance

AC2: How the grid is defined

AC3: The user interface used to retrieve user input

AC4: How hits are calculated

AC5: Pathfinding improvements

AC6: Change art assets

AC7: Map Design changes

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/Output devices to the system (system assumes mouse, keyboard, and screen are available.)

UC2: There will always be a source of input data external to the software.

UC3: The purpose of the system to be game for entertainment.

UC4: The use of a 2D grid

UC5: The use of the Unreal Engine

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: Hardware-Hiding Module

M2: Game Mode Module

M3: Character Module

M4: Map Module

M5: User Input Module

M6: User Interface Module

M7: Menu Module

M8: Camera Module

M9: Artificial Intelligence Module

M10: Grid Module

M11: Pathfinding Module

| Level 1 | Level 2 |
|--------------------------|--------------------------------|
| Hardware-Hiding Module | |
| | Game Mode Module |
| | Character Module |
| | Map Module |
| Behaviour-Hiding Module | User Input Module |
| | User Interface Module |
| | Menu Module |
| | Camera Module |
| Software Decision Module | Artificial Intelligence Module |
| | Pathfinding Module |
| | Grid Module |

Table 2: Module Hierarchy

4 Connection Between Requirements and Design

The system is designed to satisfy the requirements that were developed in the SRS. At this stage the system is decomposed into modules and compared to the requirements/ A full breakdown of the functional requirements and modules used can be seen listed in Table 3.

The non functional requirements are addressed in the implementation of the modules. The Map module will use art assets that reflect the war like environment of the setting with

assets that are of similar size. This satisfies Non Functional Requirement (NFR) for Look and Feel. The User Interface and Menu modules will be made in such a way to meet the NFR for Usability and Humanity. The usage of the modules meets the Security NFR. Those are the NFR that can be directly linked to the system.

5 Module Decomposition

Modules are decomposed to David Parnas' principle of "information hide". They are then broken down in the following manner, *Secret*, *Service*, and *Implemented By*. The *Secrets* field will contain a brief statement to the design decision that is hidden by the module. The *Services* field will specify what the module does with no regard to how. And the *Implemented By* field states by what means the module is being implemented.

Only leaf modules in the hierarchy have to be implemented. So if a – is used in *Implemented By* then it means that the module is not a leaf and does not have to be implemented.

5.1 Hardware Hiding Modules (M1)

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 display outputs or to accept inputs.

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 Game Mode Module (M2)

Secrets: Game State

Services: Stores information about the current game state. Contains functions with regards to player turn control.

Implemented By: Unreal Engine & C++ Libraries

5.2.2 Character Module (M3)

Secrets: Character

Services: Stores information regarding character actors that are active.

Implemented By: C++ Libraries

5.2.3 Map Module (M4)

Secrets: Game Map

Services: Creates the game's playable map.

Implemented By: Unreal Engine & C++ Libraries

5.2.4 User Input Module (M5)

Secrets: Input.

Services: Gets the input from the user's mouse and keyboard. Intermediate between the Input, and then the module to perform the output.

Implemented By: Unreal Engine & C++ Libraries

5.2.5 User Interface Module (M6)

Secrets: User Interface

Services: Contains functions that take the user's input and performs actions.

Implemented By: Unreal Engine & C++ Libraries

5.2.6 Menu Module (M7)

Secrets: Menu

Services: Functions to perform the various actions for the menu.

Implemented By: Unreal Engine & C++ Libraries

5.2.7 Camera Module (M8)

Secrets: Camera

Services: Functions to control the movement of the viewable space by way of a Camera.

Implemented By: Unreal Engine & C++ Libraries

5.3 Software Decision Module

Secrets: The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the SRS.

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user. Changes in these modules are more likely to be motivated by a desire to improve performance than by externally imposed changes.

Implemented By: –

5.3.1 Artificial Intelligence Module (M9)

Secrets: Artificial Intelligence

Services: Contains functions for the actions and decisions made by the Artificial Intelligence.

Implemented By: Unreal Engine & C++ Libraries

5.3.2 Grid Module (M10)

Secrets: Grid

Services: Stores the grid. Contains functions to modify the grid.

Implemented By: Unreal Engine & C++ Libraries

5.3.3 Pathfinding Module (M11)

Secrets: Pathfinding

Services: Contains functions for the movement of characters using pathfinding algorithms.

Implemented By: Unreal Engine & C++ Libraries

6 Traceability Matrix

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

| Requirement | Modules |
|-------------|--------------------|
| FR1 | M7, M4 |
| FR2 | M7 |
| FR3 | M7 |
| FR4 | M7, M6, M4 |
| FR5 | M7, M5 |
| FR6 | M2 |
| FR7 | M2, M5, M6 |
| FR8 | M2, M7 |
| FR9 | M2, M7 |
| FR10 | M2, M3 |
| FR11 | M2, M3 |
| FR12 | M3 |
| FR13 | M3, M4 |
| FR14 | M3 |
| FR15 | M3 |
| FR16 | M3, M4 |
| FR17 | M2, M3 |
| FR18 | M3 |
| FR19 | M3, M4, M10 |
| FR20 | M3, M4, M10 |
| FR21 | M3 |
| FR22 | M3 |
| FR23 | M3 |
| FR24 | M3 |
| FR25 | M2, M3 |
| FR26 | M10 |
| FR27 | M10, M4 |
| FR28 | M10, M4, M3 |
| FR29 | M2, M5, M6, M9, M4 |
| FR30 | M3, M10, M4 |
| FR31 | M4, M3 |
| FR32 | M4, M3 |
| FR33 | M6, M7, M5, M4 |
| FR34 | M2, M3, M4, M6, M5 |
| FR35 | M5, M6 |
| FR36 | M5, M6, M3, M4 |
| FR37 | M6, M2 |
| FR38 | M5, M6, M3 |
| FR39 | M6, M3 |
| FR40 | M6, M5, M3 |
| FR41 | M6, M3 |

Table 3: Trace Between Functional Requirements and Modules

| AC | Modules |
|-----|------------|
| AC1 | M3 |
| AC2 | M10 |
| AC3 | M5, M6 |
| AC4 | M3 |
| AC5 | M11 |
| AC6 | M4, M6, M7 |
| AC7 | M4 |

Table 4: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1972) 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 1 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.

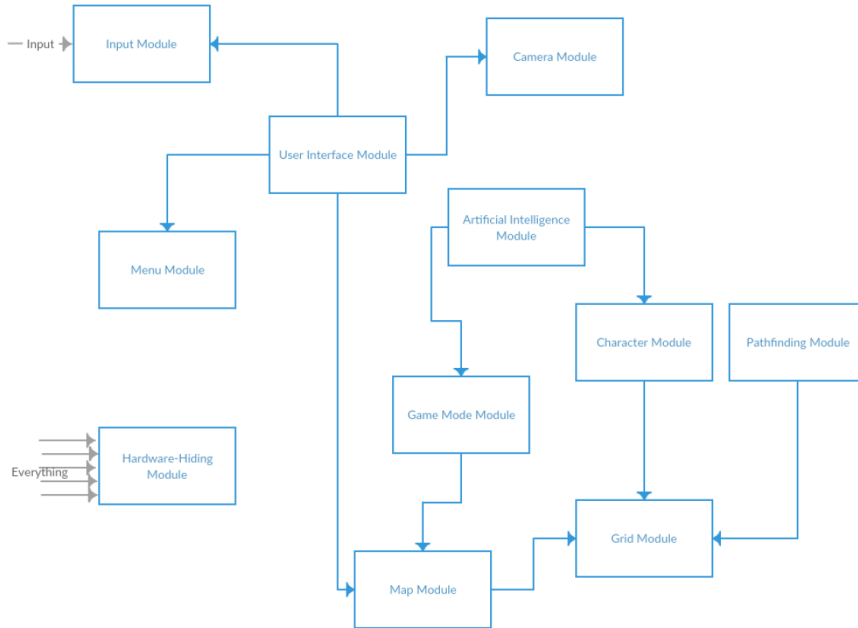


Figure 1: Use hierarchy among modules

References

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