Module Guide for SE 4G06
An AI-based Approach to Designing Board Games

Team #6, Board Gamers
Ilao Michael, ilaom
Bedi Hargun, bedih
Dang Jeffrey, dangj12
Ada Jonah, karaatan
Mai Tianzheng, mait6

April 6, 2023

1 Revision History

Date	Version	Notes
April 3rd	1.0	Split design doc into 3 documents from Rev0 feedback for Rev1
April 5th	2.0	Add Module Decomposition Game Engine and Data Visualizer by [Tianzheng Mai]

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
Board Gamers	Development Team
UC	Unlikely Change
AI	Artificial Intelligence
A	Assumption
LC	Likely Change
FR	Functional Requirement
NFR	Non Functional Requirement
FSM	Finite State Machine
TA	Teaching Assistant

Contents

List of Tables

List of Figures

3 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 implemented 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 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 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 10 describes the use relation between modules.

4 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 4.1, and unlikely changes are listed in Section 4.2.

4.1 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 specific AI agent learning method that will be used to train the AI Agents.

AC2: The number of AI agents playing a given game engine.

AC3: The specific game engine that will be integrated and developed alongside the framework

AC4: The graphs and charts to be generated by the data visualizer.

4.2 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: The entire AI library used to implement the AI Agents.

UC2: The algorithm used to accept and execute actions on the game engine.

UC3: The format of the output log from the Game Engine to the Data Visualizer.

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: AI Agent Module

M2: Game Environment Module

M3: Actions (Commands) Module

M4: Game Loop Module

M5: JSON Module

M6: Graph Module

M7: JSON Data Parser Module

5.1 Module Hierarchy

Module Type	Module Name	Module Description
AI	AI Agent Module	Trains AI Agents on the game and generates a policy
AI	Game Environment Module	Receives input from AI Agents to take action on the game
GE	Actions (Commands) Module	Describes the possible game moves that the AI Agents are able to take
GE	Game Loop Module	Continues the game loop for the game and checks if the game-over condition has been fulfilled or not.
DV	JSON Module	Responsible for recording each AI Agents moves and observation space and putting them into a JSON file.
DV	Graph Module	Produce a graph selected by the user.
DV	JSON Data Parser Module	Parses JSON files with AI Agents' move history

Table 1: Module Hierarchy

6 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 2.

7 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. *SE 4G06* means the module will be implemented by the SE 4G06 software. Implemented by AI libraries means the module will be provided by the AI Library of choice in the system.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented.

7.1 AI Modules

Secrets: The algorithms and data structures that facilitate the learning and training of the AI.

Services: Exposes methods for our code to learn and train, so the system does not need to implement from scratch.

Implemented By: SE 4G06

7.1.1 AI Agent Module (M1)

Secrets: The data structures and algorithms used to train the AI's

Services: Exposes methods for training.

Implemented By: AI Library

7.1.2 Game Environment Module (M2)

Secrets: None

Services: Interfaces with the AI Libraries to connect the Game Engine and the AI Agents

Implemented By: SE 4G06

7.2 Game Engine Module

Secrets: The model provides information on the current state of the game and possible actions to the AI Agents, and processes their response to drive the game.

Services: Continues the game loop for the game and check if the game-over condition has been fulfilled.

Implemented By: SE 4G06

7.2.1 Action (Command) Module (M3)

Secrets: The model refers to the moves that an AI Agent can take.

Services: List all valid game moves of the AI Agent.

Implemented By: SE 4G06

7.2.2 Game Loop Module (M4)

Secrets: The engine manages the overall game logic and states.

Services: keeps track of the current state of the game and respond to AI Agent's input in a continuous loop while the game is running.

Implemented By: SE 4G06

7.3 Data Visualizer Module

Secrets: The user interface is used to convey complicated game data in various graphical models.

Services: Provides visual representations of data to help users understand complex information from the game output.

Implemented By: SE 4G06

7.3.1 JSON Module (M5)

Secrets: The method used to record the JSON data of each AI Agent's moves and observation space.

Services: Gain the information from the simulations and store them in a JSON file for further use in other modules.

Implemented By: SE 4G06

7.3.2 Graph Module (M6)

Secrets: The model is used to define the statistical information, display the distribution, trends, and common winning paths of data points and make comparisons in various graphical models.

Services: Utilize the output data from the game environment and display them in diverse charts.

Implemented By: SE 4G06

7.3.3 JSON Data Parser Module (M7)

Secrets: The bridge used to connect the game Engine and data visualization model.

Services: Fetch data from the JSON log file in the game for data visualization.

Implemented By: SE 4G06

8 Traceability Matrix

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

9 Module Traceability

Module Name	Requirements
AI Agent Module	FR1 FR2 FR4 FR6 NFR1 NFR3 NFR4 NFR5
Game Environment Module	FR3 FR5 NFR1 NFR3 NFR5
Action (Command) Module	FR4 FR7 FR11 NFR3 NFR6
Game Loop Module	FR7 FR8 FR9 FR10 FR11 NFR1 NFR3 NFR4 NFR5 NFR6
JSON Module	FR3 FR13 NFR3
Graph Module	FR12 NFR2 NFR7
JSONDataParser Module	FR13 FR14 FR15

Table 2: Module Traceability

AC	Modules
AC1	M1
AC2	M1 M2
AC3	M3 M4
AC4	M6

Table 3: Trace Between Anticipated Changes and Modules

10 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1978) 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.

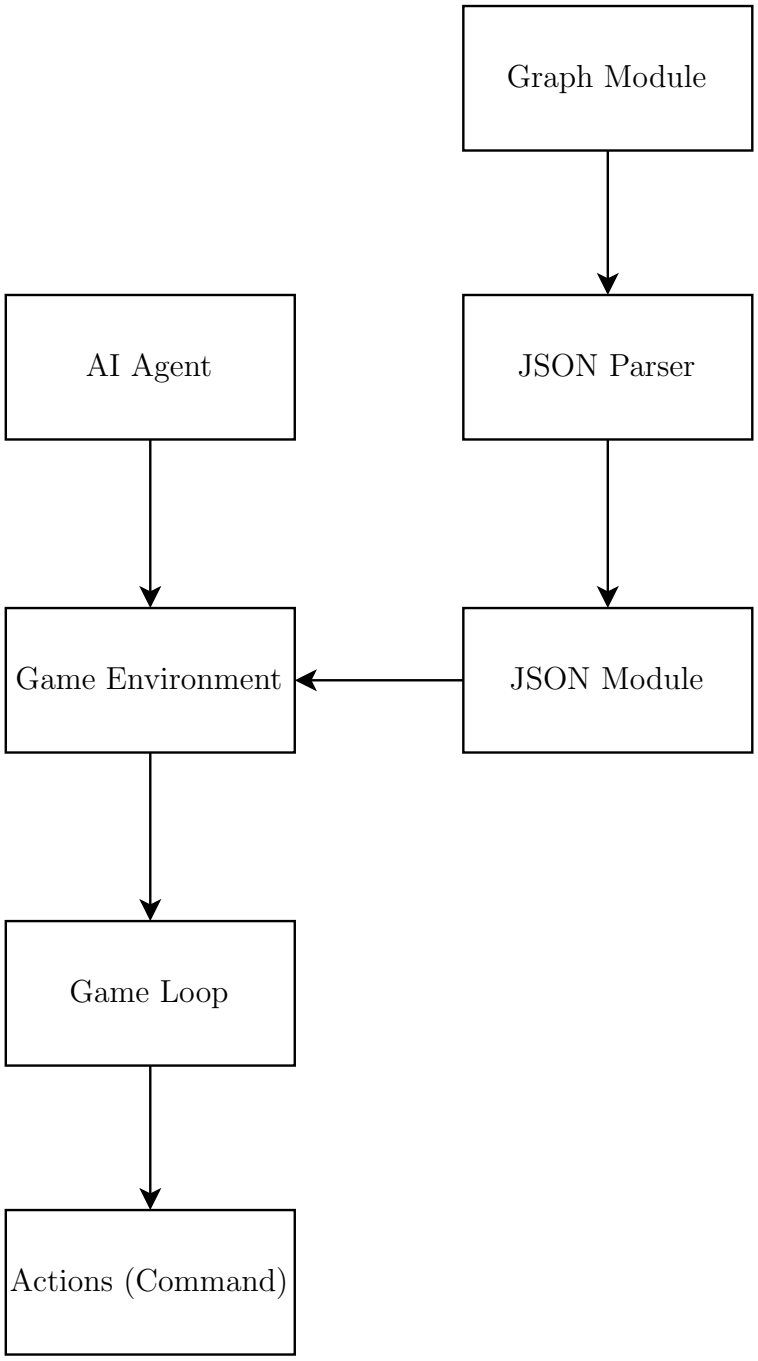

Figure 1: Use hierarchy among modules

References

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

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.