SE 3XA3: Module Guide
TankWar

Team #212, Genius
Di Wu, 400117248, wud43
Jiahao Zhou, 400082351, zhouj56
Xinyu Huang, 400120376, huangx65

April 6, 2020

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
13 March 2020	1.0	Creates the first version of MG
4 April 2020	2.0	fixed some errors following the TA's comments, merged positionType module and Map module into one and a new module called MapEditTank is added. The use relation graph is slightly changed due to the module changes.

1 Introduction

This project is to re-implement the open source project of BattleCity. The project is 2-D game allows two player to participate in and the game should work on Windows system(Win 7 and later versions) and Mac OS X systems with python and pygame.
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: How to determine the collide results between two objects.

AC2: The states of the double life tank.

AC3: The states of the high speed tank.

AC4: The states of the fast bullet tank.

AC5: The general data of a tank.

AC6: How the time is determined.

AC7: How to determine whether the tank can continue moving.

AC8: How to load and save the map as a text file.

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~~ decisions need to be changed later, 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 device. (Keyboard)

UC2: The rules of PVP gaming mode.

UC3: The rules of PVE gaming mode.

UC4: Storage of map, including the file style and the location.

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 Modules

M2: myTankControl Module

M3: Screen Module

M4: display Module

M5: PvsE Module

M6: PvsP Module

M7: mapEditing Module

M8: main Module

M9: bullet Module

M10: food Module

M11: decTime Module

M12: myTank Module

M13: highSpeedTank Module

M14: doubleLifeTank Module

M15: fastBulletTank Module

M16: enemyTank Module

M17: Map Module

~~M18: positionType Module~~

M18: MapEditTank Module

M19: wall Module

Level 1	Level 2	Module Label
Hardware-Hiding Module		M1
Behaviour-Hiding Module	myTankControl Module	M2
	Screen Module	M3
	display Module	M4
	PvsE Module	M5
	PvsP Module	M6
	mapEditing Module	M7
	~~Map Module~~	
	main Module	M8
	highSpeedTank Module	M13
	doubleLifeTank Module	M14
	fastBulletTank Module	M15
	MapEditTank Module	M18
	enemyTank Module	M16
Software Decision Module	bullet Module	M9
	food Module	M10
	decTime Module	M11
	myTank Module	M12
	~~highSpeedTank Module~~	
	~~doubleLifeTank Module~~	
	~~fastBulletTank Module~~	
	~~enemyTank Module~~	
	~~positionType Module~~	
	Map Module	M17
	wall Module	M19

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 ~~and Table~~.

The main module is designed to start the game and let the users choose the mode, then the PvsE, PvsP, mapEditing modules are used to implement the three different modes in the game. The highSpeedTank, doubleLifeTank, fastBulletTank modules with their super class myTank are designed to generate the players' tanks, and the myTankControl Module is designed for reading the keyboard input from the players to control the tank. Also, the enemyTank, food, and Map modules are designed to implement the enemy's tanks, the food can enhance the player's tanks, and the map, respectively. Finally, screen and display mod-

ules are designed to show the GUI output.
Besides the functions of the game, the non-functional requirements are also expected to be met by these modules. The look and feel requirements and style requirements are met by the screen and display modules which work for the GUI output, the usability and humanity requirements are also met by design the screen module, which shows very detailed instructions before the game starts, the performance requirements are basically met by designing all the modules to be stable and efficient, and the maintainability requirements are met by designing good comments in modules with the tool Doxygen.

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.

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. Whether or not this module is implemented depends on the programming language selected.

5.1 Hardware Hiding Modules (M1)

Secrets: ~~The data structure and algorithm used to implement the virtual hardware~~ How the virtual hardware is implemented.

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 myTankControl Module (M2)

Secrets: Key pressed by users.

Services: Responds to the corresponding key pressed.

Implemented By: function operatePlayer1 and operatePlayer2.

5.2.2 Screen module (M3)

Secrets: The condition of showing each kinds of screen in the game.

Services: Display the corresponding screen in the game in different situations.

Implemented By: Screen module, pygame

5.2.3 Display module (M4)

Secrets: The process of determining the colliding results.

Services: Display any related actions based on the colliding results on the screen during the game.

Implemented By: Display module, pygame

5.2.4 PvsE Module (M5)

Secrets: All the data related to the PVE mode such as delay, enemy numbers.

Services: Generate a PVE mode for users to play.

Implemented By: PvsE module and pygame.

5.2.5 PvsP Module (M6)

Secrets: All the data related to the PVP mode such as tank groups, delay.

Services: Generate a PVP mode for users to play.

Implemented By: PvsP module and pygame.

5.2.6 mapEditing module (M7)

Secrets: The way of saving the map into a text file.

Services: Allowing users to create their own PVP or PVE map and save in the corresponding folder.

Implemented By: Screen module, pygame

5.2.7 ~~Map Module (M8)~~

~~**Secrets:** The Map module is hiding the arrangement of the bricks, irons, and homes in the PVE and PVP map. It also hides the methods to load and save a map from a local file and into a local file respectively.~~

~~**Services:** Generates a PVE map. Generates a PVP map. Loads and saves a PVE map. Loads and saves a PVP map.~~

~~**Implemented By:** Map module and pygame.~~

5.2.8 main module (M8)

Secrets: ~~Way of generating the menu screen and method for connecting PVP, PVE, and map editing mode together in the main module.~~
The implementation of selecting the mode

Services: Display the overall menu of the game and allow the user to select the mode.

Implemented By: main module, pygame

5.2.9 highSpeedTank Module (M13)

Secrets: The property of the High Speed Tank. The specific data about the High Speed Tank. The ultimate skill leap implementation is hided in the module

Services: Create a High Speed Tank object. The module provides leap skill to boost the speed of the tank.

Implemented By: myTank module and pygame.

5.2.10 doubleLifeTank Module (M14)

Secrets: The property of the Double Life Tank. The specific data about the Double Life Tank. The ultimate skill Bullet Proof implementation is hided in the module

Services: Create a Double Life Tank object. The module provides bullet proof skill to allow the tank block all the bullets in a certain period.

Implemented By: myTank module and pygame.

5.2.11 fastBulletTank Module (M15)

Secrets: The property of the Fast Bullet Tank. The specific data about the Fast Bullet Tank. The ultimate skill Double Bullet implementation is hided in the module

Services: Create a Fast Bullet Tank object. The module provides double bullet skill to allow the tank shoot two bullet at a time.

Implemented By: myTank module and pygame.

5.2.12 enemyTank Module (M16)

Secrets: The property of the Enemy Tank. The specific data about the Enemy Tank.

Services: Create a Enemy Tank object. The module provides how the Enemy Tank would act during the game.

Implemented By: enemyTank module and pygame.

5.2.13 MapEditTank Module (M18)

Secrets: The property of the Map Edit Tank. The specific data about the Map Edit Tank. The way to edit the map is hided in the module

Services: Create a Map Edit Tank object. The module provides how can the tank edit the map

Implemented By: myTank module and pygame.

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.

Implemented By: –

5.3.1 bullet Module (M9)

Secrets: The property of the bullet. The specific data about the bullet.

Services: Create a bullet object.

Implemented By: bullet module and pygame.

5.3.2 food Module (M10)

Secrets: The property of the food. The specific data about the food.

Services: Create a food object.

Implemented By: food module and pygame.

5.3.3 decTime Module (M11)

Secrets: The way to count the time in the game.

Services: Count the time.

Implemented By: decTime module.

5.3.4 myTank module (M12)

Secrets: The process of determining the colliding results between the users' tanks and the wall. The movement and shoot algorithm is hiding in this module.

Services: If the colliding result is true, the tanks should not be able to move. The movement and shooting function are provided in the module.

Implemented By: myTank module, pygame

5.3.5 ~~highSpeedTank Module (M14)~~

~~**Secrets:** The property of the High Speed Tank. The specific data about the High Speed Tank. The ultimate skill leap implementation is hided in the module~~

~~**Services:** Create a High Speed Tank object. The module provides leap skill to boost the speed of the tank.~~

~~**Implemented By:** myTank module and pygame.~~

5.3.6 ~~doubleLifeTank Module (M15)~~

~~**Secrets:** The property of the Double Life Tank. The specific data about the Double Life Tank. The ultimate skill Bullet Proof implementation is hided in the module~~

~~**Services:** Create a Double Life Tank object. The module provides bullet proof skill to allow the tank block all the bullets in a certain period.~~

~~**Implemented By:** myTank module and pygame.~~

5.3.7 ~~fastBulletTank Module (M16)~~

~~**Secrets:** The property of the Fast Bullet Tank. The specific data about the Fast Bullet Tank. The ultimate skill Double Bullet implementation is hided in the module~~

~~**Services:** Create a Fast Bullet Tank object. The module provides double bullet skill to allow the tank shoot two bullet at a time.~~

~~**Implemented By:** myTank module and pygame.~~

5.3.8 ~~enemyTank Module (M17)~~

~~**Secrets:** The property of the Enemy Tank. The specific data about the Enemy Tank.~~

~~**Services:** Create a Enemy Tank object. The module provides how the Enemy Tank would act during the game.~~

~~**Implemented By:** enemyTank module and pygame.~~

5.3.9 ~~positionType Module (M18)~~

~~**Secrets:** The structure of a coordinate.~~

Services: ~~Generate a method of representing coordinates.~~

Implemented By: ~~positionType module.~~

5.3.10 wall Module (M19)

Secrets: The constructor of brick, iron wall and home base.

Services: Create brick wall, iron wall, and home base objects.

Implemented By: wall module and pygame.

5.3.11 Map Module (M8)

Secrets: The Map module is hiding the arrangement of the map.
~~bricks, irons, and homes in the PVE and PVP map. It also hides the methods to load and save a map from a local file and into a local file respectively.~~

Services: Generates a PVE map. Generates a PVP map. Loads and saves a PVE map. Loads and saves a PVP map.

Implemented By: Map module and pygame.

6 Traceability Matrix

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

Req.	Modules
FR1	M5, M6, M7, M8
FR2	M7, M17, M18, M19
FR3	M7, M17
FR4	M2, ~~M4~~, M12
FR5	M2, M13, M14, M15
FR6	M2, M9, M12
FR7	M4, M9
FR8	M4, M9
FR9	M2, M12
FR10	M3, ~~M5, M6, M12, M13, M14, M15~~
FR12	M10
FR13	M4
FR14	M4, M16
FR15	M4, M5
FR16	M4, M5
FR17	M5, M17
FR18	M6, M17
FR19	M5
FR20	M6
FR21	M4, M5, M6, ~~M12, M13, M14, M15~~
LF1	~~M3, M5, M6,~~ M8
LF2	M3
LF3	~~M7~~, M19, M17
LF4	M19
LF5	M19
LF6	~~M12~~, M13, M14, M15, M16
LF7	M19
LF8	M5, M6
UH1	M3
UH2	M3
UH4	M3
UH5	M3
PR1	~~M8~~, M4, M2
PR2	~~M8~~, M7, M17
PR4	M7
PR5	~~M8~~ All the modules
PR6	~~M8~~ All the modules
PR7	M7
PR9	~~M8, M5, M6~~ M2

Req.	Modules
OE1	~~M7~~ All the modules
OE3	~~M7~~ All the modules
OE4	~~M7~~ All the modules
OE5	~~M7~~ All the modules
MS2	~~M7~~ All the modules
MS3	~~M7~~ All the modules
MS4	~~M7~~ All the modules
SR2	~~M16, M6~~ M17
SR3	~~M7~~ M17
CR1	~~M2~~ M3
CR2	~~M2~~ M3

Table 3: Trace Between Requirements and Modules

AC	Modules
AC1	M4
AC2	M14
AC3	M13
AC4	M15
AC5	M12
AC6	M11
AC7	~~M12~~, M2
AC8	M7, M17

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 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.

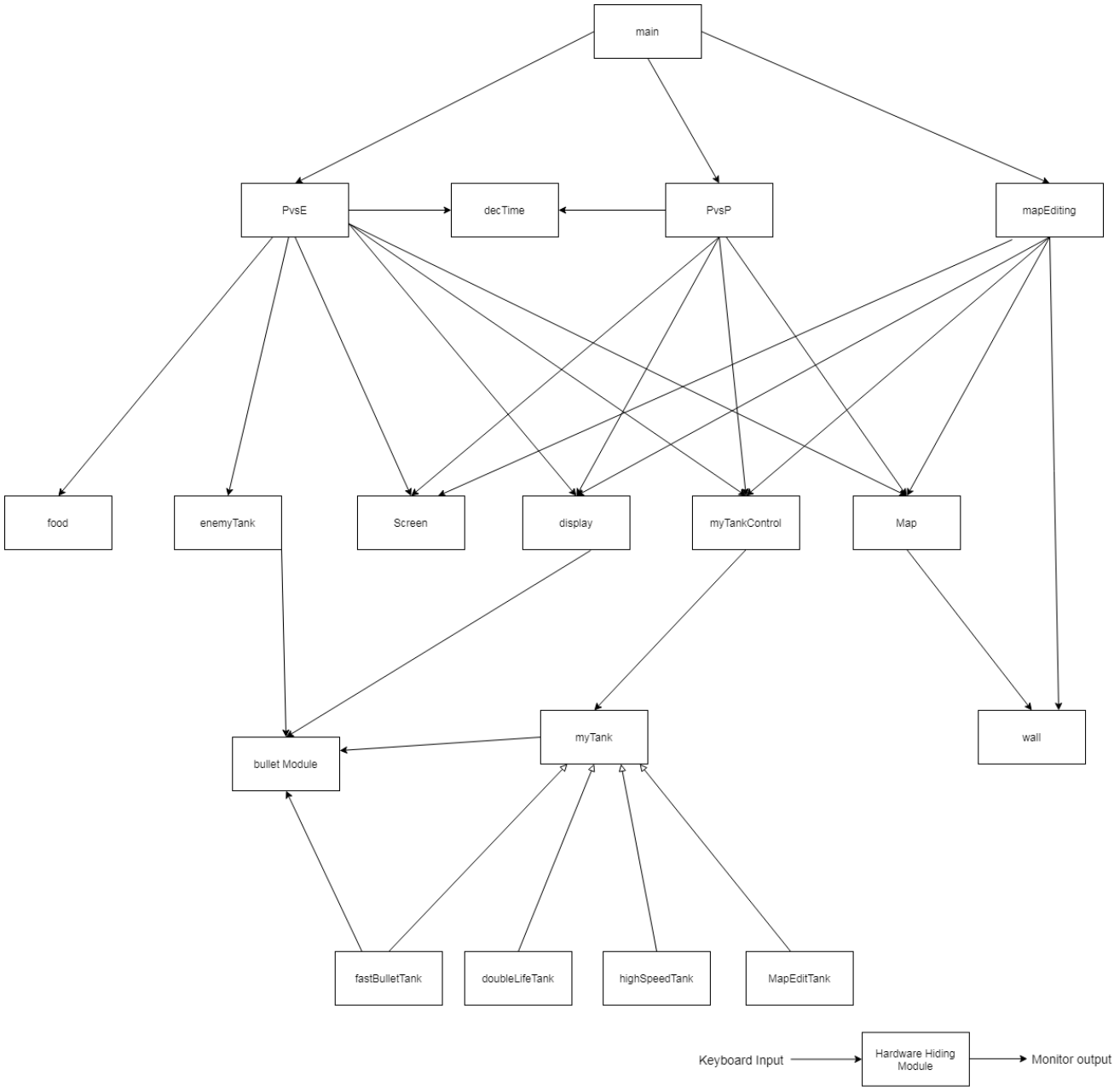

Figure 1: Use hierarchy among modules

References

David L. Parnas. 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.