SE 3XA3: Module Guide
T-Rex Acceleration

Team 15, Dev$^{\text{enthusiasts}}$
Zihao Du (duz12)
Andrew Balmakund (balmakua)
Namit Chopra (choprn9)

April 9, 2021

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
03/18/2021	1.0	Initial Version
04/07/2021	2.0	Revision 1

1 Introduction

Overview

T-Rex Acceleration is a python based implementation of the Google Chrome game T-Rex Runner, which is implemented with JavaScript. The goal of the project is to create a new version of T-Rex Runner game based on the existing implementation with a fresh twist. New features like changing volume and powerups will be added to the project. This project also uses the MVC design pattern to design all modules.

Context

The module guide (MG) is based on the functional and non-functional requirements mentioned in the software requirements specification (SRS). The purpose of the document is to provide developers with a modular structure of the system, and how requirements can be met and traced in the structure. It is also helpful to check the interaction between modules and the consistency of the system. Along with MG, there will be a module interface specification (MIS) document in which these modules are specified in detail, including access routine, state variable, etc.

Design Principles

We advocate a decomposition based on the principle of information hiding (Parnas, 1972). This principle supports design for change because of the "secrets" that each module hides represent likely future changes. Design for change is valuable in software development, where modifications are frequent, especially during initial development as the solution space is explored.

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: The method to generate new background colours.

AC2: The number and types of powerups that can be acquired by the player.

AC3: The number and types of obstacles that the player has to avoid.

AC4: New images for the character, powerups, obstacles, and environment.

AC5: Sounds files associated with different game features.

AC6: The dimensions and position of the character, powerups, obstacles, and environment drawn on the screen.

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: The only way to interact with the system by using the keyboard (except the close button of the window and setting menu). The user must use the keyboard to control the character during the game. The keyboard is also needed to navigate through the main, pause, and end menu. This includes the settings menu as well, which is accessed through the main menu.

UC2: The number of players, currently one, that can play the game at once on the same machine is not expected to change. The game is designed to be played by one player and that is kept in mind when designing all the modules.

UC3: The environment and platforms where the game can be played are unlikely to be changed. The game is expected to be supported by Windows, macOS, and Linux.

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: Character

M2: Powerups

M3: Obstacle

M4: DetectCollision

M5: UpdateEnvironment

M6: Score

~~[M7:] MainMenu~~

M8: DisplayObstacle

M9: DisplayPowerups

M10: DisplayEnvironment

M11: DisplayWindow

M12: DisplayCharacter

M13: PlaySound

M14: DisplayMenu

M15: LoadAssets

M16: GameController

M17: MenuController

Level 1	Level 2
Hardware-Hiding Module	DisplayMenu DisplayCharacter DisplayPowerups DisplayObstacles DisplayEnvironment PlaySound DisplayWindow LoadAssets
Behaviour-Hiding Module	GameController MenuController
Software Decision Module	Character Powerups Obstacle DetectCollision UpdateEnvironment Score ~~MainMenu~~

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.

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

Secrets: The data structure and algorithm used to implement the virtual hardware.

Services: Serve 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

5.1.1 DisplayMenu (M14)

Secrets: ~~Handles all control flow to open different menus.~~User input handling for the settings menu and method to draw the menus.

Services: Display the different menus (pause menu, exit menu, instruction setting menu) onto the screen.

Implemented By: Dev$^{\text{enthusiasts}}$

5.1.2 DisplayCharacter (M12)

Secrets: Method to draw the character on the screen.

Services: Draw the character onto the screen.

Implemented By: Dev$^{\text{enthusiasts}}$

5.1.3 DisplayPowerups (M9)

Secrets: Method to randomly generate powerups at different times.

Services: Generate, draw, and update the positions and speeds of the powerups.

Implemented By: Dev$^{\text{enthusiasts}}$

5.1.4 DisplayObstacles (M8)

Secrets: The data structure storing all obstacles on the screen.

Services: Provide functionality to draw, remove, and update the positions and speeds of obstacles.

Implemented By: Dev$^{\text{enthusiasts}}$

5.1.5 DisplayEnvironment (M10)

Secrets: ~~Display different GUI environment elements when playing the game.~~ Method to draw different GUI environment elements in the gameplay.

Services: Provide functionality to draw the score, game instructions, floor, and background onto the screen.

Implemented By: Dev$^{\text{enthusiasts}}$

5.1.6 PlaySound (M13)

Secrets: ~~Play different sounds when specific actions are performed by the user at a set volume.~~Methods to change different sound volume settings

Services: Play background music and sound effects for the character jumping, ducking, colliding with obstacle, and collecting a powerup.

Implemented By: Dev$^{\text{enthusiasts}}$

5.1.7 DisplayWindow (M11)

Secrets: ~~Initialize the window of the application using several different properties.~~ The dimension of the game window and the method to initialize the window

Services: Generate an initial screen window.

Implemented By: Dev$^{\text{enthusiasts}}$

5.1.8 LoadAssets (M15)

Secrets: The data structures of all images and sounds used in the game.

Services: Load all image and audio files into the game system.

Implemented By: Dev$^{\text{enthusiasts}}$

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 GameController (M16)

Secrets: ~~Handle all internal game logic and user input.~~ User input handling and game logic.

Services: Responsible for calling the appropriate subroutines throughout the game session.

Implemented By: Dev$^{\text{enthusiasts}}$

5.2.2 MenuController (M17)

Secrets: ~~Handles most of the menu related user input.~~ User input handling during menus.

Services: Interact with the user on the setting, exit, instruction, resume and pause menu.

Implemented By: Dev$^{\text{enthusiasts}}$

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 Character (M1)

Secrets: The data structure of the character.

Services: Provide functionality to control movement of character (jump or duck) and apply different behaviours when powerups are collected.

Implemented By: Dev$^{\text{enthusiasts}}$

5.3.2 Powerups (M2)

Secrets: The data structure of the powerup.

Services: Provide properties for the type and speed related to these powerups.

Implemented By: Dev$^{\text{enthusiasts}}$

5.3.3 Obstacle (M3)

Secrets: The data structure of the obstacle.

Services: Provides functionality to set and get different properties of an obstacle such as changing the dimensions, speed, image display, and position.

Implemented By: Dev$^{\text{enthusiasts}}$

5.3.4 DetectCollision (M4)

Secrets: The algorithm to calculate collision between two objects.

Services: Detect whether or not there is a collision between two objects.

Implemented By: Dev$^{\text{enthusiasts}}$

5.3.5 UpdateEnvironment (M5)

Secrets: ~~Changing~~ Method to change the background color during the gameplay.

Services: Updating the background color and ground floor position.

Implemented By: Dev$^{\text{enthusiasts}}$

5.3.6 Score (M6)

Secrets: The data structure of the score.

Services: Responsible for tracking , boost and returning the current and high score.

Implemented By: Dev$^{\text{enthusiasts}}$

5.3.7 ~~MainMenu (M7)~~

~~[Secrets:] Maintains different volume settings for different sounds.~~

~~[Services:] Changing the volume according to user input and return the volume setting.~~

~~[Implemented By:] Dev$^{\text{enthusiasts}}$~~

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	M1, M12, M13, M16
FR2	M1, M12, M13, M16
FR3	M1, M3, M4, M13, M16
FR4	M3, M8, M13
FR5	M17
FR6	M14, M17
FR7	M16, M17
FR8	M16, M17
FR9	M6, M10, M16
FR10	M6, M10, M17
FR11	M2, M9
FR12	M1, M2, M4, M9, M12, M13, M16
FR13	M1, M12
FR14	M13, M15, M16
FR15	M14, M17
FR16	~~M7,~~ M14, M16, M17
FR17	M10, M16

Table 3: Trace Between Requirements and Modules

AC	Modules
AC1	M5
AC2	M1, M9
AC3	M8
AC4	M15
AC5	M13, M15
AC6	M8, M9, M10, M12

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 2 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.

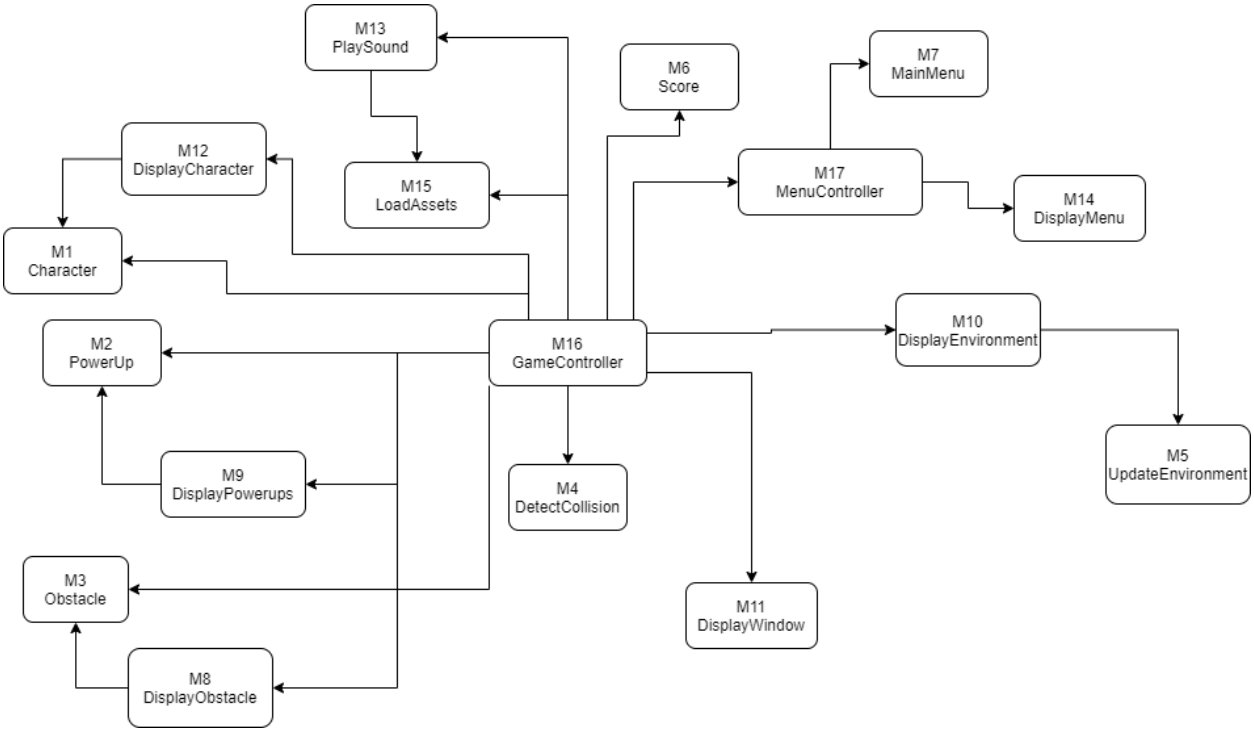

Figure 1: Use hierarchy among modules (Rev0)

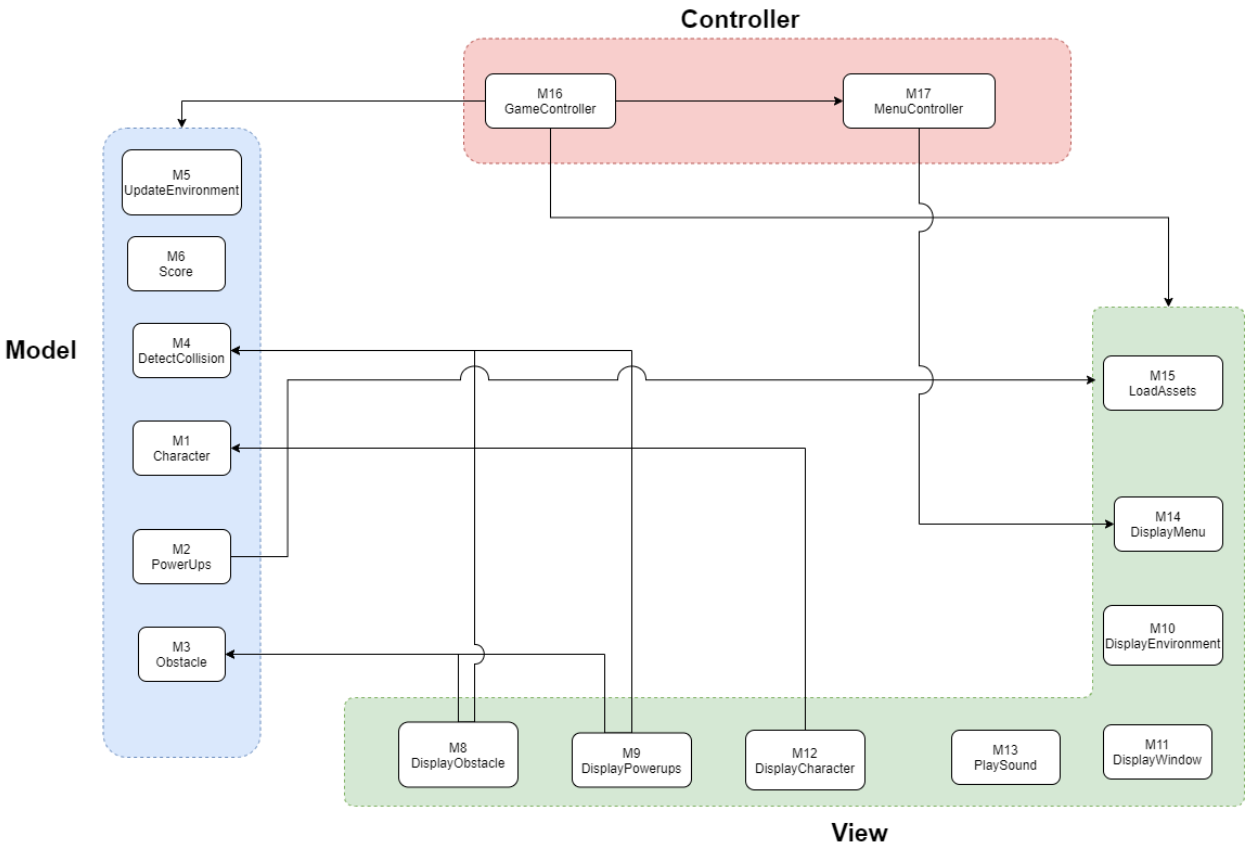

Figure 2: Use hierarchy among modules (Rev1)

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.