SE 3XA3: Module Guide
Super Refactored Mario Python

203, Abstract Connoisseurs
David Jandric, jandricd
Daniel Noorduyn, noorduyd
Alexander Samaha, samahaa

April 7, 2020

Contents

1 Introduction **1**
- 1.1 Overview . . . 1
- 1.2 Scope . . . 1
- 1.3 Purpose . . . 1

2 Anticipated and Unlikely Changes **1**
- 2.1 Anticipated Changes . . . 1
- 2.2 Unlikely Changes . . . 2

3 Module Hierarchy **2**

4 Connection Between Requirements and Design **4**

5 Module Decomposition **5**
- 5.1 Hardware Hiding Modules (M1) . . . 5
  - 5.1.1 Game_Controller Module (M1) . . . 5
- 5.2 Behaviour-Hiding Module . . . 5
  - 5.2.1 Input Format Module (M2) . . . 6
  - 5.2.2 Menu Module (M3) . . . 6
  - 5.2.3 Dashboard Module (M4) . . . 6
  - 5.2.4 Pause Module (M5) . . . 6
  - 5.2.5 Mario Module (M6) . . . 6
  - 5.2.6 Camera Module (M7) . . . 7
  - 5.2.7 Goomba Module (M8) . . . 7
  - 5.2.8 Koopa Module (M9) . . . 7
  - 5.2.9 Coin Module (M20) . . . 7
  - 5.2.10 Item Module (M21) . . . 7
  - 5.2.11 RandomBox Module (M22) . . . 8
  - 5.2.12 MushroomItem Module (M23) . . . 8
  - 5.2.13 PowerUpBox Module (M24) . . . 8
- 5.3 Software Decision Module . . . 8
  - 5.3.1 Vector2D Module (M10) . . . 8
  - 5.3.2 Sound_Controller Module (M11) . . . 9
  - 5.3.3 SpriteSheet Module (M12) . . . 9
  - 5.3.4 Collider Module (M13) . . . 9
  - 5.3.5 Level Module (M14) . . . 9
  - 5.3.6 ~~Levels.json Module (M3)~~ . . . 9
  - 5.3.7 ~~Sprites Module (M3)~~ . . . 10
  - 5.3.8 ~~Sprite Module (M3)~~ . . . 10
  - 5.3.9 ~~Sprite~~ Entity Base Module (M18) . . . 10
  - 5.3.10 Animation Module (M19) . . . 10

Table 1: **Revision History**

Date	Version	Notes
13/03/20	1.0	Initial Write of Modular Guide

5.3.11 EntityCollider Module (M??) . . . 10
5.3.12 BounceTrait Module (M26) . . . 10
5.3.13 GoTrait Module (M27) . . . 11
5.3.14 JumpTrait Module (M28) . . . 11
5.3.15 LeftRightWalkTrait Module (M29) . . . 11
5.3.16 Tile Module (M30) . . . 11

6 Traceability Matrix **11**

7 Use Hierarchy Between Modules **12**

List of Tables

1 **Revision History** . . . ii
2 Module Hierarchy . . . 4
3 Trace Between Requirements and Modules . . . 12
4 Trace Between Anticipated Changes and Modules . . . 12

List of Figures

1 Use hierarchy among modules . . . 13
2 Use hierarchy among modules - Revision 1 . . . 14

1 Introduction

1.1 Overview

Super Refactored Mario Bros is modeled after the original Super Mario Bros game from Nintendo. The goal of the project is to create an almost perfect clone of the original game, but make it open source. This would allow others to learn how to use the python library pygame. Additionally, the creator of the original code that this project is based on hasn't completed the game. Further, they have created some unorganized and quite amateur code which our team hopes to refactor into readable and understandable code. The team will also finish the game and add several key lacking features. This document aims to translate the functional and non-functional requirements as seen in the SRS into a direct programming scenario. This document outlines how behaviours desired in the SRS will be implemented into the code of the game. Our team has decided on a Main-Subroutine architecture which is how the following outlined modules are designed to interact with each other.

1.2 Scope

The final version of Super Refactored Mario Bros will be a replica of the original Super Mario Bros game at its base functionality. It will be created solely using pygame. However, special features such as the multiplicity of power-ups or moving pipes or underground sections will not be implemented. This is because the goal of the team was to complete the base game and organize and/or rewrite the existing code.

1.3 Purpose

The purpose of this document is to provide a written, organized representation between the code design and the requirements. In a large program such as Super Refactored Mario Bros, there are many working parts that all need to work together seamlessly to create the desired program. As a result, this document will help the development team keep track of how the modules must interact and function. It also provides a map to future programmers who desire to add or change features to the open source game.

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 specific hardware on which the software is running.

AC2: The amount and type of characters the user can choose from.

AC3: The amount and type of power-ups from the item boxes.

AC4: The frame-rate of specific computers.

AC5: The resolution of different monitors.

AC6: The new sounds corresponding to any new features.

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 current and only input comes from specific keys on the user's keyboard. The keyboard allows the user to navigate menus and play the game. If this source of input were the change, the entire game would have to be re-designed.

UC2: The current game locks the size of the game window which results in a simple way to create and design new levels. If window resizing were to be implemented, a large section of the program would be heavily effected.

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: ~~Controller~~ Game_Controller

M2: Input

M3: Menu

M4: Dashboard

M5: Pause

M6: Mario

M7: Camera

M8: Goomba

M9: Koopa

M10: Vector2D

M11: Sound_Controller

M12: Spritesheet

M13: Collider

M14: Level

~~**M15:** Levels.json~~

~~**M16:** Sprites~~

~~**M17:** Sprite~~

M18: Entity_Base

M19: Animation

M20: Coin

M21: Item

M22: RandomBox

M23: MushroomItem

M24: PowerUpBox

M25: EntityCollider

M26: BounceTrait

M27: GoTrait

M28: JumpTrait

M29: LeftRightWalkTrait

M30: Tile

Level 1	Level 2
Hardware-Hiding Module	~~Controller~~ Game_Controller
Behaviour-Hiding Module	Input Menu Dashboard Pause Mario Camera Goomba Koopa Coin Item RandomBox MushroomItem PowerUpBox
Software Decision Module	Vector2D Sound_Controller SpriteSheet Collider Level ~~Levels.json~~ ~~Sprites~~ ~~Sprite~~ EntityBase Animation EntityCollider BounceTrait GoTrait JumpTrait LeftRightWalkTrait Tile

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". 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 structures and algorithms used to implement the virtual input and output hardware from the user's keyboard.

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

5.1.1 Game_Controller Module (M1)

Secrets: Process of starting the game, passing objects to others, and running the game.

Services: The main controller that initializes subroutines and organizes the functionality of the entire architecture.

Implemented By: Super Refactored Mario Bros

5.2 Behaviour-Hiding Module

Secrets: Required behaviours of the game elements.

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: Super Refactored Mario Bros

5.2.1 Input Format Module (M2)

Secrets: The format and structure of the input data.

Services: Converts the input data into the data structure used by the input parameters module.

Implemented By: Super Refactored Mario Bros

5.2.2 Menu Module (M3)

Secrets: The format and structure of the game menu's items and graphics as well as links to other modules based on user selection.

Services: Displays the options on the game window and transfers the user to where they want to go in the game.

Implemented By: Super Refactored Mario Bros

5.2.3 Dashboard Module (M4)

Secrets: The formatting and gathering of the user's score, total collected coins, time left, and number of lives left.

Services: Displays the user's score, total collected coins, time left, and number of lives left in game.

Implemented By: Super Refactored Mario Bros

5.2.4 Pause Module (M5)

Secrets: The pausing of game play and creating a pause menu over top of the blurred current game window. Additionally it creates a link back to the main menu and manipulates game settings.

Services: Displays a menu that allows user to alter settings and provides a link back to the main menu.

Implemented By: Super Refactored Mario Bros

5.2.5 Mario Module (M6)

Secrets: How the character interacts with other objects in the game, such as enemies, blocks and coins.

Services: Provides the functionalities of the user playable character through the game.

Implemented By: Super Refactored Mario Bros

5.2.6 Camera Module (M7)

Secrets: How the game camera pans and follows the user's character.

Services: Provides the driver to move the camera of the game as the user's character moves through the world.

Implemented By: Super Refactored Mario Bros

5.2.7 Goomba Module (M8)

Secrets: The function of the Goomba enemies such as their movement and how they can be killed.

Services: Allows the user to interact with an enemy, and gain points from their death.

Implemented By: Super Refactored Mario Bros

5.2.8 Koopa Module (M9)

Secrets: The function of the Koopa enemies such as their movement, how they can be killed and their shell movement.

Services: Allows the user to interact with an enemy, and gain points from their death.

Implemented By: Super Refactored Mario Bros

5.2.9 Coin Module (M20)

Secrets: Contains the coin animation (revolving) for static in level coins and how they can disappear.

Services: The user can interact with these coins to gain points.

Implemented By: Super Refactored Mario Bros

5.2.10 Item Module (M21)

Secrets: Contains the coin animation for appearing out of the random box and when/how the coins disappear.

Services: The user can interact with these coins to gain points.

Implemented By: Super Refactored Mario Bros

5.2.11 RandomBox Module (M22)

Secrets: Contains the random box animation, and when the box is functional/non-functional.

Services: The user can interact with the box and gain points from getting coins from it.

Implemented By: Super Refactored Mario Bros

5.2.12 MushroomItem Module (M23)

Secrets: Contains the functions to make the mushrooms appear, move the mushroom, and disappear.

Services: The user can interact with these mushrooms to transform Mario from 'big Mario' to 'small Mario' and gain points.

Implemented By: Super Refactored Mario Bros

5.2.13 PowerUpBox Module (M24)

Secrets: Contains the power-up box animation and when the box is functional/non-functional.

Services: The user can interact with the power-up box to get a mushroom to appear.

Implemented By: Super Refactored Mario Bros

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: Super Refactored Mario Bros

5.3.1 Vector2D Module (M10)

Secrets: The method of doing the vector addition, as well as calculation the magnitude of a vector.

Services: Uses vector addition in order to create new vectors for doing math with positions, velocities and accelerations.

Implemented By: Super Refactored Mario Bros

5.3.2 Sound_Controller Module (M11)

Secrets: The method of playing audio in the game for the user, which sounds are used and the channels over which he music and sounds will be played

Services: Provides functionality to play, stop and mute sounds or music.

Implemented By: Super Refactored Mario Bros

5.3.3 SpriteSheet Module (M12)

Secrets: Where and how the game accesses the various files to pull images from and create sprites

Services: Provides functionality to isolate an image to be used to create a sprite.

Implemented By: Super Refactored Mario Bros

5.3.4 Collider Module (M13)

Secrets: How the game detects that two objects have overlapped and are in a collision state.

Services: Provides methods and functions to check positions to determine if a collision has occurred.

Implemented By: Super Refactored Mario Bros

5.3.5 Level Module (M14)

Secrets: The method of which objects and entities in the game are spawned and placed in a level.

Services: Loads objects and entities from sprites and places them within the context of a level.

Implemented By: Super Refactored Mario Bros

5.3.6 ~~Levels.json Module (M3)~~

~~**Secrets:** Contains the structure of every level as a combination of sprites.~~

~~**Services:** Used to create unique levels that the user can play.~~

~~**Implemented By:** Super Refactored Mario Bros~~

5.3.7 ~~Sprites Module (M3)~~

~~**Secrets:** The method of which collections of sprites are loaded to be used in the game.~~

~~**Services:** Loads a collection of sprites to be used by the different entities and level building modules.~~

~~**Implemented By:** Super Refactored Mario Bros~~

5.3.8 ~~Sprite Module (M3)~~

~~**Secrets:** The method of which the game draws a sprite.~~

~~**Services:** Provides functionalities to draw a sprite from an image to be used in the game.~~

~~**Implemented By:** Super Refactored Mario Bros~~

5.3.9 ~~Sprite~~ Entity Base Module (M18)

Secrets: The various fields that are general to all attributes in the game.

Services: Provides functionalities to update and access fields required of all attributes in the game.

Implemented By: Super Refactored Mario Bros

5.3.10 Animation Module (M19)

Secrets: The images to be displayed and how they change over time.

Services: Provides animation functionality to any entities required by the game.

Implemented By: Super Refactored Mario Bros

5.3.11 EntityCollider Module (M??)

Secrets: The methods for checking if two entities are colliding.

Services: Provides functionality to entities that can interact with other entities in the game.

Implemented By: Super Refactored Mario Bros

5.3.12 BounceTrait Module (M26)

Secrets: How this module updates an entities state.

Services: Gives entities a method for bouncing up.

Implemented By: Super Refactored Mario Bros

5.3.13 GoTrait Module (M27)

Secrets: How this module updates an entities state and how it interacts with the user.

Services: Provides advanced movement functionality for an entity.

Implemented By: Super Refactored Mario Bros

5.3.14 JumpTrait Module (M28)

Secrets: How this module updates an entities state and how it interacts with the user.

Services: Provides jumping functionality to an entity.

Implemented By: Super Refactored Mario Bros

5.3.15 LeftRightWalkTrait Module (M29)

Secrets: How this module updates an entities state.

Services: Provides simple horizontal movement to an entity.

Implemented By: Super Refactored Mario Bros

5.3.16 Tile Module (M30)

Secrets: How the module stores data.

Services: Provides a simple data class for representing a static object with an image.

Implemented By: Super Refactored Mario Bros

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, M2, M6, M13, ~~M3, M3, M3~~, M19, M25, M28
FR2	M1, M2, M6, M7, M13, ~~M3, M3, M3~~, M19, M25
~~FR3~~	~~M1, M2, M6, M13, M3, M3, M3~~
FR4	M1, M2, M6, M7, M13, ~~M3, M3, M3~~, M19, M25
~~FR5~~	~~M1, M2, M6, M7, M13, M3, M3, M3~~
FR6	M1, M2, M6, M7, M8, M13, ~~M3, M3, M3~~, M19, M25, M28, M29
FR7	M1, M2, M6, M7, M9, M13, ~~M3, M3, M3~~, M18, M19, M25, M27, M28, M29
FR8	M1, M2, M6, M7, M9, M13, ~~M3, M3, M3~~, M18, M19, M25, M28, M26, M29
FR9	M1, M6, M7, M8, M9, M13, ~~M3, M3, M3~~, M18, M19, M25, M27
FR10	M1, M6, M4
FR11	M1, M4, M6
FR12	M1, M3, M4, M6
FR13	M1, M14, ~~M3~~
FR14	M1, M4, M6
FR15	M1, M6, M23, M24, M25, M28, M29
FR16	M1, M6, M23, M25, M29
FR17	M1, M6, M25, M29
FR18	M1, M4, M6, M21, M22, M25, M28
FR19	M1, M4, M6, M20, M25

Table 3: Trace Between Requirements and Modules

AC	Modules
AC1	M1, M2
AC2	M6
AC3	M6, M24
AC4	M1
AC5	M14
AC6	M11

Table 4: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. If 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.

OLD VERSION

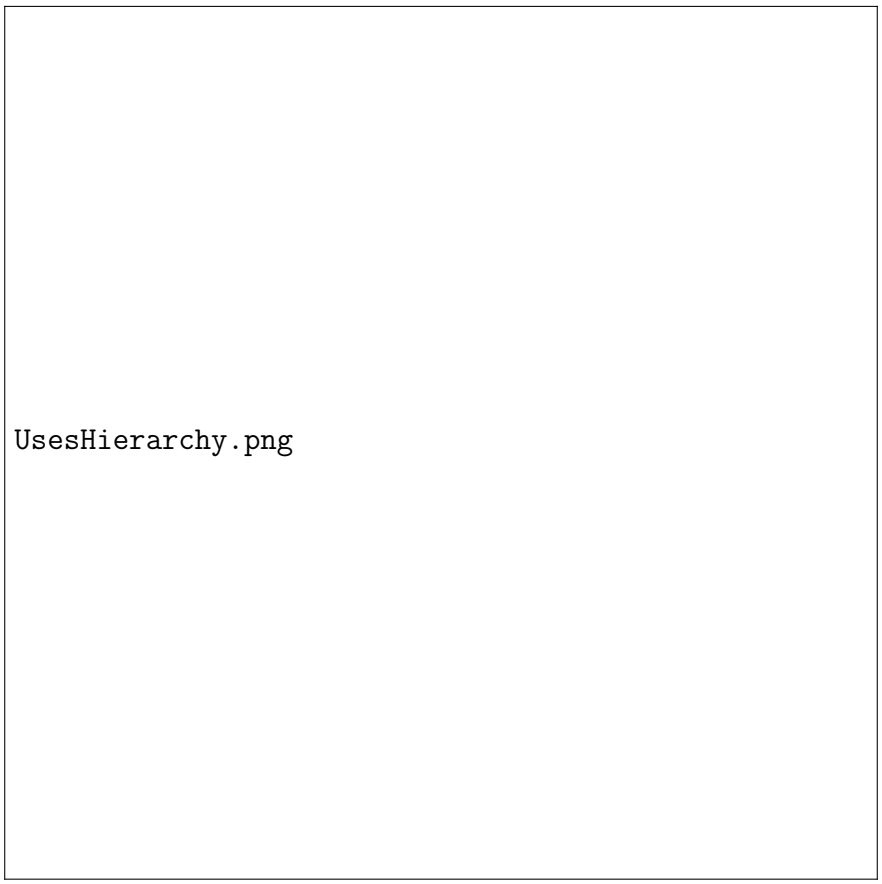

Figure 1: Use hierarchy among modules

NEW VERSION

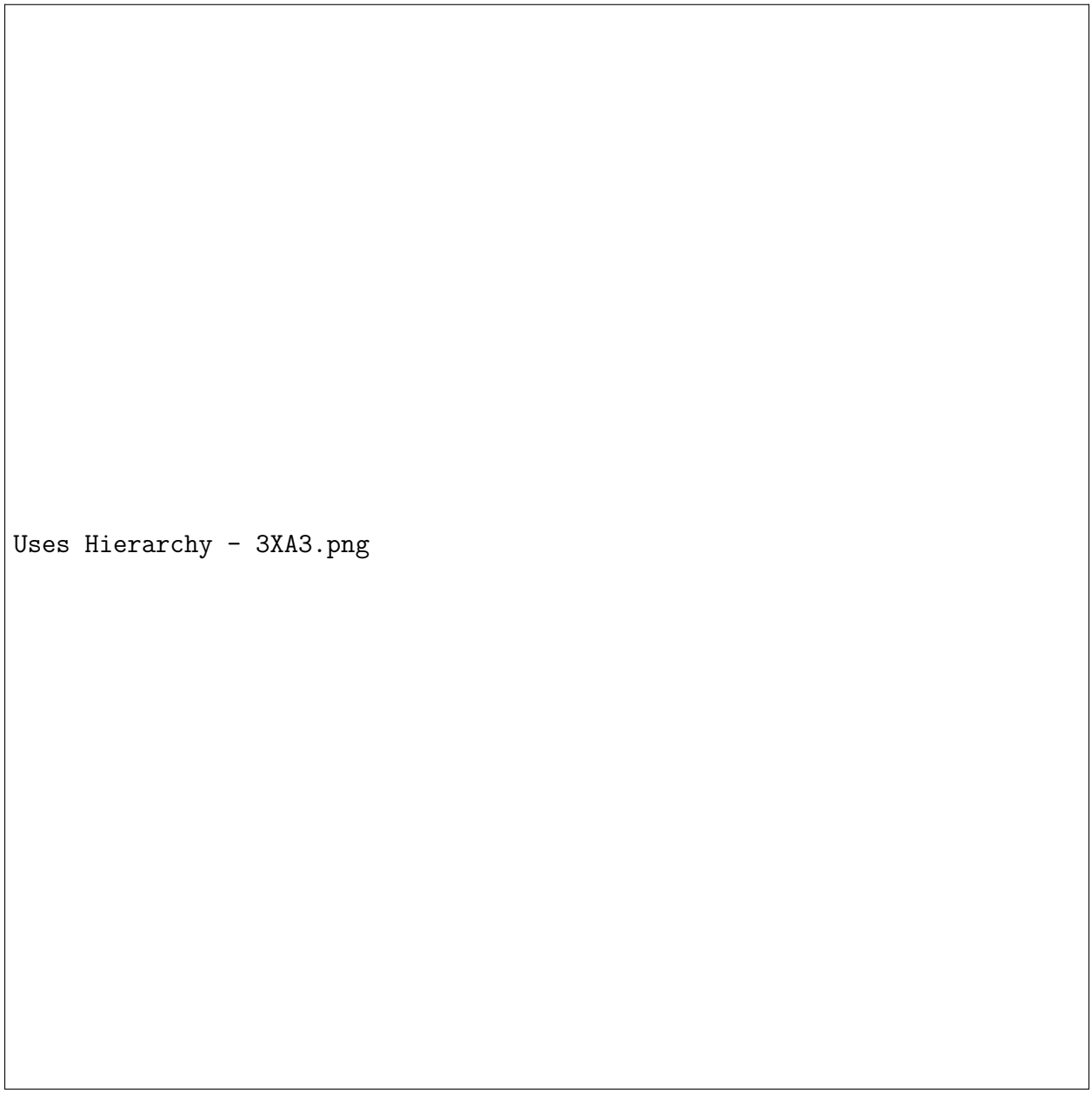

Figure 2: Use hierarchy among modules - Revision 1