SE 3A04: Software Design III: Large System Design

Group #5, Spaceship System Sabotage
Pareek Ravi 001407109
Pavle Arezina 001410366
David Hobson 001412317
Victoria Graff 001401451
Julian Cassano 001406891

March 9, 2017

Contents

List of Tables

List of Figures

1 Introduction

1.1 Purpose

The Large System Design document is responsible for the high level architecture design of the system and the results produced from this analysis will be utilized in the design phase. The document is derived from the requirements and specifications of the overall system, determined in the previous software requirement specification document. The intended audience of the document is mainly for the developers who will be utilizing the large system design document to understand the overall design of the program to help structure the responsibilities assigned to each class. Developers who are maintaining the program may also use the document to ensure new additions to the program follow the system design properly to ensure a smooth experience for the user.

1.2 System Description

Space system sabotage is a simulation of a fictional spaceship that is travelling to a pre-determined location in a fixed amount of time. The user will interact with the system through random events based on a function of the running time of the simulation. Random events will involve negative situations that occur, such as comets damaging the ship or part of the ship being sabotaged, where the user will have to respond to fix them in a timely manner with a selection of tools. If successful, the space ship continues toward its goal but if an incorrect solution was chosen then the problem is spread to other systems. If enough events happen with incorrect solutions to them, then the space ship will slowly spiral out of control and the user will fail the simulation. This system offers an entertaining environment to the user where they will be challenged to keep the spaceship intact until it dock import at the space station.

1.3 Overview

The rest of the Large System Design document is separated into four sections, the first being the use case diagram. This diagram models the functionality of the system using actors, the users of the system, and use cases of the system. The next section is the analysis class diagram which describes the key classes of a system and their interrelationship through boundary class, entity classes, and controller classes. These classes of the system are identified by extracting through the description of the use case descriptions. Section three should provide an overview of the overall architectural design of the application where the division of the system into subsystems follows high cohesion and low coupling. Last section is the Class Responsibility Collaborator (CRC) cards on the classes in the analysis class diagram where each card identifies and assigns responsibilities to classes required to build the system. After these four sections there is an appendix that includes the division of labour.

2 Use Case Diagram

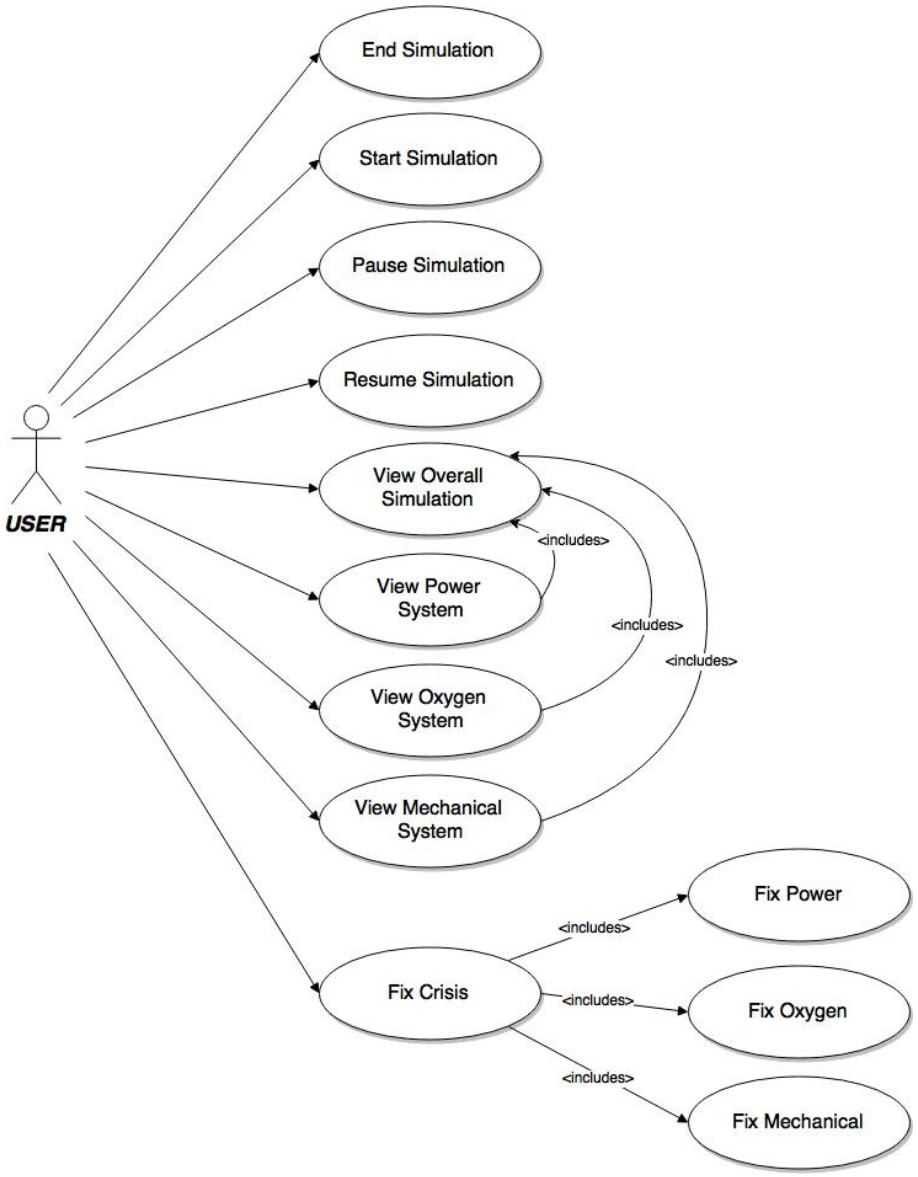

Figure 1: Use Case Diagram

Use Case Definitions

a) End Simulation: The user is in a play session and intends to indefinitely stop the simulation and return to the main menu.

b) Start Simulation: The user is in the main menu and intends to begin a play session.

c) Pause Simulation: The user is in a play session and intends to pause the simulation but intends to resume at a later time.

d) Resume Simulation: The user is in a paused play session and intends to resume and continue the simulation.

e) View Overall System: The user is in a play session and intends to display the status of all the subsystems at once.

f) View Power System: The user is in a play session and intends to include the status of the Power System in the displayed view.

g) View Oxygen System: The user is in a play session and intends to include the status of the Oxygen System in the displayed view.

h) View Mechanical System: The user is in a play session and intends to include the status of the Mechanical System in the displayed view.

i) Fix Crisis: The user is in a play session and intends to resolve an event that is negatively affecting one of the subsystems.

j) Fix Power: The user is in a fix crisis event and intends to resolve an event affecting the power system.

k) Fix Oxygen: The user is in a fix crisis event and intends to resolve an event affecting the oxygen system.

l) Fix Mechanical: The user is in a fix crisis event and intends to resolve an event affecting the mechanical system.

3 Analysis Class Diagram

Figure 2: Analysis Class Diagram of System

4 Architectural Design

4.1 System Architecture

The overall architecture utilized in this project is the presentation-abstraction-control (PAC) architecture, essentially a hierarchical structure made out of model-view-controller (MVC) blocks. PAC architecture is an interaction orientated software architecture that was developed from MVC to support the application requirement of multiple agents in addition to the interactive application requirements. Each agent has a control module that plays an active role, where the model and view module are separated which entails the sub-system architecture. The multiple agents are ordered in a hierarchy where the controller components communicated in a well-structured method. It is very suitable for any distributed system where each remote agent has its own functionalities with data and interactive interface.

With the analysis class diagram, it was determined that the architecture needed to

follow high coupling and low cohesion design principle with a requirement of sub-systems to be able to be easily detached from the overall system. By striving to implement the first design principle mentioned coupled with exploring the ways the user will be able to utilize the system, it became clear that structure similar to MVC would be utilized to create the architecture of the program. With self-contained modules that can be easily detached from the system needed to fulfill user requirements, it became apparent that some sort of hierarchical system would be utilized where small units of MVC sub-systems can be added and removed when needed. Clearly the PAC architecture is well suited for this system where it allows multi-viewing of agents, easy to plug in new agent or replace existing agent.

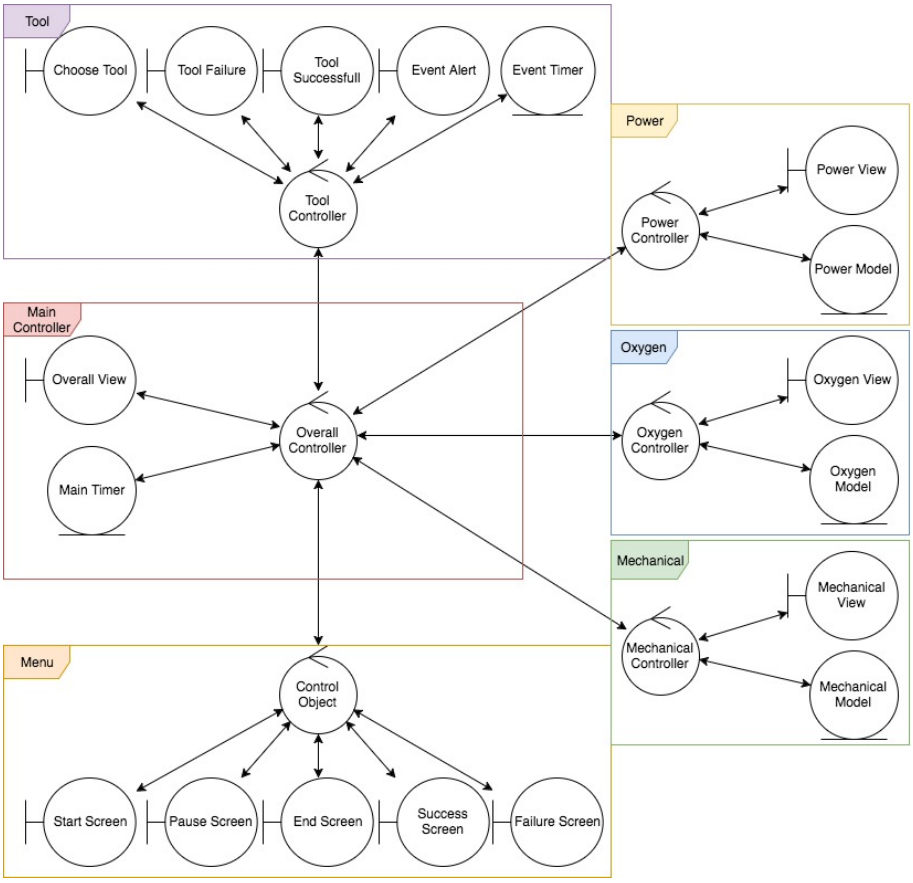

Figure 3: Analysis Class Diagram of System Grouped

4.2 Subsystems

- The power subsystem controls all of the all of the interactions that relate to the simulated amount of electrical power in the system and is dependent on the main controller. This subsystem can "malfunction" and must be stimulated to return to base state.
- The oxygen subsystem controls all of the interactions that relate to the simulated amount of air in the system and is dependent on the main controller. This subsystem can "malfunction" and must be stimulated to return to base state.

- The mechanical subsystem controls all of the interactions that relate to the simulated physical moving parts in the system and is dependent on the main controller. This subsystem can "malfunction" and must be stimulated to return to base state.
- The tool subsystem controls all of the interactions that will "fix" the other simulated subsystems and return them to normal state.
- The menu subsystem controls all interactions outside of the play session and will allow the user to perform actions such as pausing, resuming, or quitting the play session.
- The main controller subsystem is the main hub of interaction between all other subsystems and will appropriately stimulate each subsystem at the correct time. This subsystem receives input from all other subsystems and returns the output back to each system to determine what will happen in the simulation.

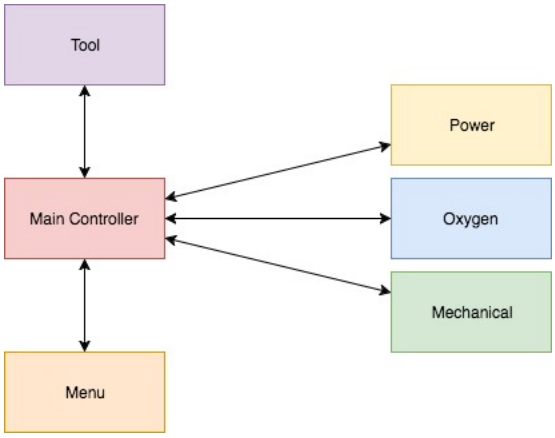

Figure 4: Structural Architecture Diagram for the PAC system.

5 Class Responsibility Collaboration (CRC) Cards

Class Name: Start Screen	
Responsibility:	**Collaborators:**
Receive request to display a prompt to start game	Menu Controller
Display prompt to user	
Respond to prompt being pressed by user	
Send request to Menu Controller to start game	Menu Controller

Class Name: End Screen	
Responsibility:	**Collaborators:**
Receive request to display a prompt to end game	Menu Controller
Display prompt to user	
Respond to prompt being pressed by user	
Send request to Menu Controller to end game	Menu Controller

Class Name: Pause Screen	
Responsibility:	**Collaborators:**
Receive request to display a prompt to pause game	Menu Controller
Display screen prompt to user	
Respond to prompt being pressed by user	
Send request to Menu Controller to pause and unpause game	Menu Controller

Class Name: Success Screen	
Responsibility:	**Collaborators:**
Receive request to display a screen message when game has been won	Menu Controller
Display prompt to user	

Class Name: Failure Screen	
Responsibility:	**Collaborators:**
Receive request to display a screen message when game has been lost	Menu Controller
Display prompt to user	

Class Name: Control Object	
Responsibility:	**Collaborators:**
Receive request to pause or unpause game	Pause Screen
Receive request to end game	End Screen
Receive request to start game	Start Screen
Send request to start game, end game or pause game	Overall Controller
Receive request to start, end or pause game	Overall Controller
Send request to start game	Start Screen
Send request to end game	End Screen
Send request to pause game	Pause Screen

Class Name: Choose Tool	
Responsibility:	**Collaborators:**
Display tool selection to user	
Recive tool selection from user	
Send request to use tool	Tool Controller
Receive request to display tool selection	Tool Controller

Class Name: Tool Fail	
Responsibility:	**Collaborators:**
Display to user that they chose the wrong tool	
Receive request to display failure message	Tool Controller

Class Name: Tool Success	
Responsibility:	**Collaborators:**
Display to user that they chose the right tool	
Receive request to display success message	Tool Controller

Class Name: Event Alert	
Responsibility:	**Collaborators:**
Receive request to show that there is an issue with the spaceship	Tool Controller
Show issue in spaceship	

Class Name: Event Timer	
Responsibility:	**Collaborators:**
Hold information about when events occur and sends to tool controller	Tool Controller

Class Name: Tool Controller	
Responsibility:	**Collaborators:**
Receive user's tool choice	Choose Tool
Determine if it was the correct tool to use and inform user	Tool Success, Tool Fail
Inform the user of an issue	Event Alert
Receive event from overall controller	Overall Controller
Receive time to determine when event should occur	Event Timer

Class Name: Power Model	
Responsibility:	**Collaborators:**
Hold and update information of the power system	
Receive updates and from the overall controller	Overall controller

Class Name: Power View	
Responsibility:	**Collaborators:**
Receive request if view is hidden or shown	Power Controller

Class Name: Power Controller	
Responsibility:	**Collaborators:**
Updates subsystem information after tool usage	Power Model, Overall Controller
Informs overall controller of subsystem information	Power Model, Overall Controller
Informs the overall controller what is being displayed	Power View, Power Model, Overall Controller
Generate the stimulation based on a time given	Overall Controller

Class Name: Oxygen Model	
Responsibility:	**Collaborators:**
Hold and update information of the oxygen system	
Receive updates and from the overall controller	Overall controller

Class Name: Oxygen View	
Responsibility:	**Collaborators:**
Receive request if view is hidden or shown	Oxygen Controller

Class Name: Oxygen Controller	
Responsibility:	**Collaborators:**
Updates subsystem information after tool usage	Oxygen Model, Overall Controller
Informs overall controller of subsystem information	Oxygen Model, Overall Controller
Informs the overall controller what is being displayed	Oxygen View, Oxygen Model, Overall Controller
Generate the stimulation based on a time given	Overall Controller

Class Name: Mechanical Model	
Responsibility:	**Collaborators:**
Hold and update information of the mechanical system	
Receive updates and from the overall controller	Overall controller

Class Name: Mechanical View	
Responsibility:	**Collaborators:**
Receive request if view is hidden or shown	Mechanical Controller

Class Name: Mechanical Controller	
Responsibility:	**Collaborators:**
Updates subsystem information after tool usage	Mechanical Model, Overall Controller
Informs overall controller of subsystem information	Mechanical Model, Overall Controller
Informs the overall controller what is being displayed	Mechanical View, Mechanical Model, Overall Controller
Generate the stimulation based on a time given	Overall Controller

Class Name: Overall View	
Responsibility:	**Collaborators:**
Display spaceship system	
Display one or multiple subsystems	Overall Controller
Get user input on which subsystem issue to handle	Overall Controller

Class Name: Main Timer	
Responsibility:	**Collaborators:**
Send game time to Overall Controller	Overall Controller

Class Name: Overall Controller	
Responsibility:	**Collaborators:**
Requests to start, pause and end game	Control Object
Handle tool choices and relay information to subsystem	Tool Controller, Power Controller, Oxygen Controller, Mechanical Controller
Handle input from user for which view to display	Power Controller, Oxygen Controller, Mechanical Controller, Overall View
Giving time tics to subsystem to generate stimuli	Main Timer, Power Controller, Oxygen Controller, Mechanical Controller

A Division of Labour

Member	**Duties**	**Signature**
David Hobson	Analysis Class Diagram System Architecture Editing	
Pavle Arezina	Introduction System Architecture Editing	
Pareek Ravi	CRC Cards	
Victoria Graff	CRC Cards Use Case Diagram	
Julian Cassano	System Architecture Use Case Diagram	