Deliverable #2 Tutorial 2 Group 5

SE 3A04: Software Design II – Large System Design

1 Introduction

The following document is a description of the desired interaction between the intended user and the systems, as well as the architectural details of the software.

1.1 Purpose

The purpose of this document is twofold. First, it is to provide a higher-level view of the system, using simplified graphical representation of what the system must do. Second, it is to provide a basic general guideline for architectural implementation of the systems within the software. The intended audience of this document is mainly focused for the developers of the software by generalizing the internal details of the software, however any stakeholders may find the document of interest as the functional requirements are conveyed here.

1.2 System Description

SpaceBase Ephemeris (SBE) is a real-time system that simulates a human settlement on a foreign planet, named Ephemeris. The simulation takes place in a distant future where technology has improved to sustain humankind on extraterrestrial planets which are otherwise uninhabitable. As a commander of the base, the user has the responsibility of manipulating the sub-systems to ensure all of the key operations are functional, and possibly ideal. The three subsystems are power control, atmospheric simulation, and the station personnel.

1.3 Overview

The first part of this document provides a graphical representation of the functional requirements of the software and how it must interact with the intended users using an use case diagram. The remainder of the document provides guidelines for internal design of the software, using an analysis class diagram, a detailed discussion on the architectural design of the system, and a series of class responsibility collaboration cards.

2 Use Case Diagram

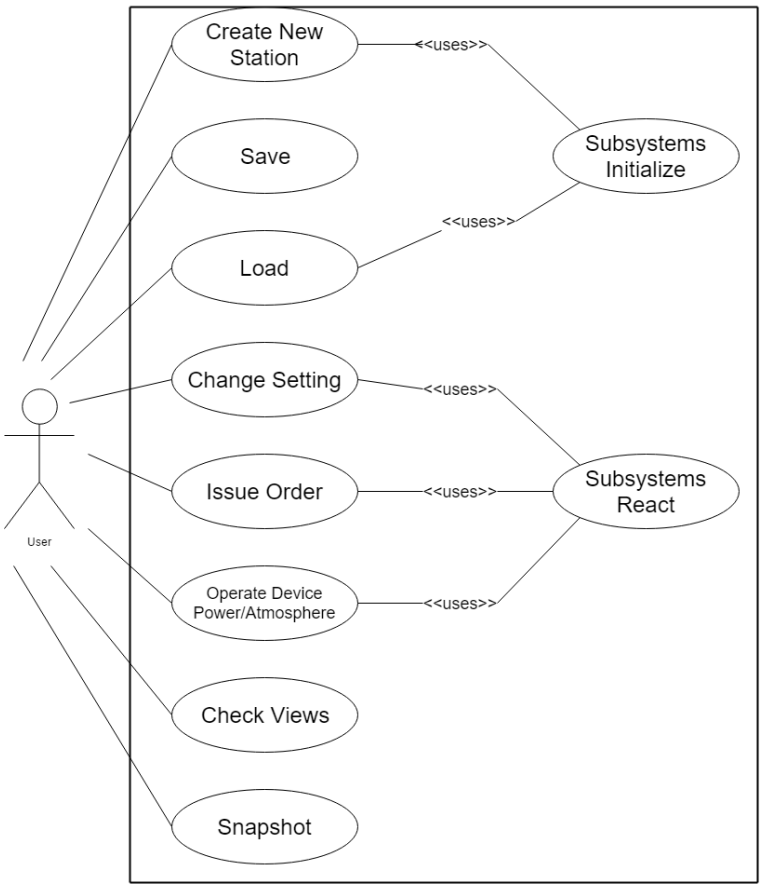

Figure 1: The use case diagram

1. Create New Station: The users opens the application to the main menu. They select the new-station button. They enter parameters for the station such as size, or accept the defaults, then confirm. The game generates a station with the selected parameters, initializes the subsystems, then opens and displays the new station.

2. Load: The user opens the application to the main menu. They select the load-previous-save button, then selects the appropriate file on their device. The game deserializes the game-state from the file, initializes the subsystems, then opens it and displays it.

3. Save: The user pauses while playing the game, then selects the save-game button. They enter an appropriate filename into the prompt given. The system serializes the game-state, and writes it to the file location given, then returns to the pause screen.

4. Change Setting: The user pauses while playing the game, then selects the settings button. The user enters a new value for one of the settings displayed (for example changing the speed at which personnel move and act). The user confirms the new setting, causing the subsystems to implement it, and react appropriately to any changes in the world it may cause.

5. Issue Order: The user selects a location within the station. They select that a generator should be built at that location. The personnel subsystem determines which personnel will carry out the order, and assigns them to do so. As time passes, the assigned personnel install the generator.

6. Operate Device: The user selects an airlock door in the station to be opened. The power subsystem then checks that the door is powered, and operates it. The atmospheric subsystem circulates gases between the two sides to correct any imbalances. The personnel system reroutes personnel through the door if it creates a shorter path to their destination.

7. Check Views: The users pauses while playing the game. They toggle on or off any number of subsystem views, then return to gameplay. The game now displays the selected views.

8. Snapshot: The user pauses while playing the game. They select the snapshot button and are prompted to enter their user credentials for Twitter. They enter them and confirm. If the system determines the credentials are valid, it creates a picture out of the current game state and views, then posts the picture with the user's credentials, and returns the user to the game.

3 Analysis Class Diagram

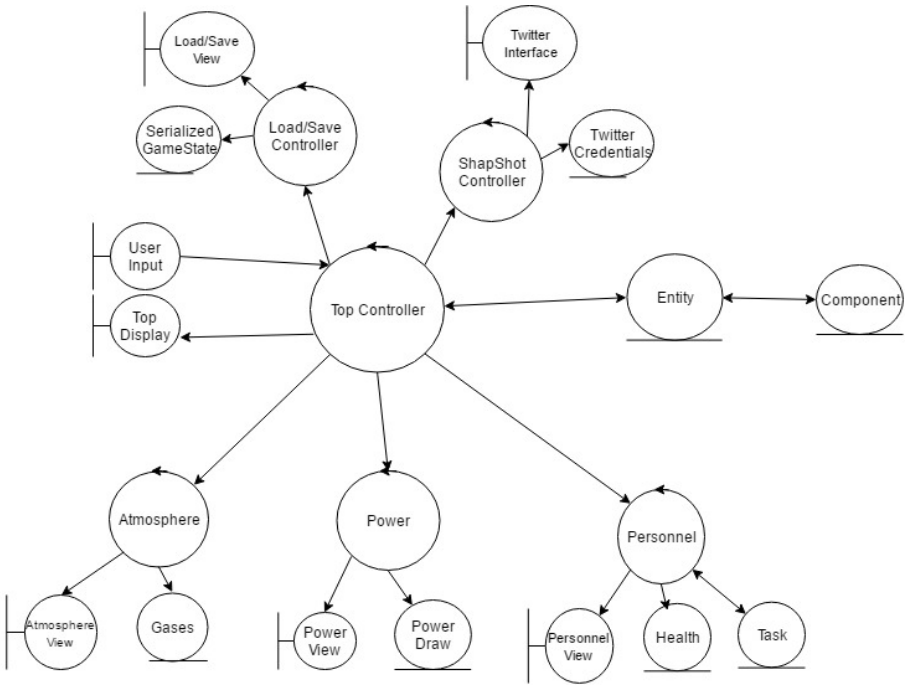

Figure 2: The analysis class diagram

4 Architectural Design

This section should provide an overview of the overall architectural design of your application. You overall architecture should show the division of the system into subsystems with high cohesion and low coupling.

4.1 System Architecture

The overall architecture of the system is a **Presentation, Abstraction, and Control (PAC)** architecture. The three subsystems, as well as a few other major components are all PAC agents, with a single master PAC agent which orchestrates them. The PAC architecture was chosen first because it is an effective architecture for interaction-orientated software. It was chosen over other such architectures because it is an effective model for systems with a set of agents, each with their own model and interface, such as this. It is useful for situations where agents may need to be easily removed, replaced, or added later in development, a

requirement for this system. Since each subsystem must have a separate, toggle-able view, an architecture that supports multiple Presentations is a good choice. The hierarchical nature of PAC allows for low coupling as subsystems can have an overarching controller which can orchestrate communication, rather than tying each subsystem directly to each other subsystem. Lastly, MVC is a poor choice as it would lead to low cohesion when all the subsystems are grouped into a single controller.

There is also an Entity Component System (ECS) architecture, which is common in games to simplify inheritance hierarchies and reduce coupling in cases where objects may have relevant properties to multiple subsystems (for eg, a tile may have properties relevant to both atmospheric and power subsystems). The contents of the station, such as tiles, people, and devices, are entities, which contain a set of components, each relevant to a specific subsystem.

4.2 Subsystems

The following is a description of the subsystems included in the system.

a) Atmospheric: Simulates the flow of various gases through the station, and their effects. Various devices powered by the Power subsystem can affect these flows, and the gases (or lack thereof) can affect the Personnel subsystem by damaging the personnel.

b) Power: Simulates the power flow through station wires, the draw of various devices, and the output of station generators. These devices can affect the Atmospheric subsystem by moving, adding, or removing gases in the station. The Power subsystem can affect the Personnel subsystem with these devices, for example a depowered airlock cannot be opened, preventing the passage of personnel.

c) Personnel: Simulates the personnel aboard the station, and controls their movements and actions. The station personnel can directly affect the Atmospheric subsystem by slowly changing the gas composition by breathing. They can directly affect the Power subsystem by installing, removing, or using various devices in the station.

5 Class Responsibility Collaboration (CRC) Cards

Class Name: MainMenuGUI	
Responsibility:	**Collaborators:**
Display buttons for user input	
Direct view from main menu to a new station	StationGenerator
Show user saved games upon load game request	MainMenuAbstract
Open chosen game from saved games list	StationGenerator
On user request for settings, open settings menu	SettingsGUI

Class Name: MainMenuController	
Responsibility:	**Collaborators:**
Create new station	StationGenerator
Load values from saved game file into game to allow game to continue from saved state	StationGenerator

Class Name: MainMenuAbstraction	
Responsibility:	**Collaborators:**
Store saved games and list on request	MainMenuGUI
Load values from saved game file to continue game	MainMenuController

Class Name: SettingsGUI	
Responsibility:	**Collaborators:**
Display settings window	
Get current settings attributes	SettingsAbstract
Display text for settings description and current settings attribute	
Display the buttons for user input	
Send new settings to abstract	SettingsController

A Division of Labour

1. Ian: Designed use case diagram, wrote subsystem descriptions, use case scenarios, and architecture description, wrote CRC cards, designed analysis class diagram
2. Nishanth: Designed use case diagram, designed analysis class diagram
3. Arfa: Designed use case diagram, designed analysis class diagram
4. Steven: Wrote Introduction section, wrote CRC cards
5. Areeb: Drew use case diagram from the paper reference, designed use case diagram, wrote CRC cards

Class Name: SettingsController	
Responsibility:	**Collaborators:**
Get updated settings values	SettingsGUI
Apply the new changes	SettingsAbstract

Class Name: SettingsAbstraction	
Responsibility:	**Collaborators:**
Store settings values	
Return requested attributes	SettingsGUI
Change requested attributes	SettingsController

Class Name: StationGenerator	
Responsibility:	**Collaborators:**
Load default station values to create station base	
Load existing station values from file chosen in main menu and recreate station	MainMenuController

Class Name: PersonnelController	
Responsibility:	**Collaborators:**
Accept orders from user	TopController
Instruct personnel how to perform orders	Task
Keep personnel away from dangers	Health
Simulate personnel's condition in the presence of danger	Health

Class Name: PersonnelView	
Responsibility:	**Collaborators:**
Display locations of personnel	
Display tasks of personnel	

Class Name: Task	
Responsibility:	**Collaborators:**
Store current task of a person	Component

Class Name: Health	
Responsibility:	**Collaborators:**
Store current liveliness status of a person	Component

Class Name: PowerController	
Responsibility:	**Collaborators:**
Simulate powerflow through the station	PowerDraw
Operate powered devices	Powerdraw, Device

Class Name: PowerView	
Responsibility:	**Collaborators:**
Display flow of power through the station	PowerController
Display overall power status of station	PowerController

Class Name: PowerDraw	
Responsibility:	**Collaborators:**
Store the amount of power an entity adds or draw from the grid	Component

Class Name: PowerDraw	
Responsibility:	**Collaborators:**
Store the amount of power an entity adds or draw from the grid	Component