Final Project Design & Reflection

Caleb Schmidt • CS162 • Fall 2017

Design

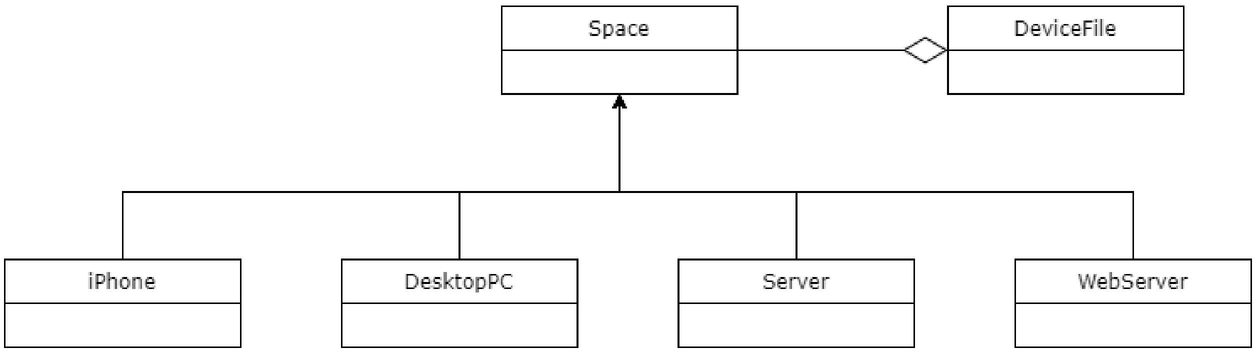
The problem to be solved

The core requirement of the final project is to implement a navigable series of interconnected, customized spaces, all of which inherit from a common Space class. Therefore, the core design challenges will be to: implement the spaces; link them in a proper topology; handle dynamic memory allocation and deallocation; coordinate player navigation among the spaces; and create an overall theme and goal for the game.

The theme I have chosen for my game is an abstract interpretation of the specification in which the 'spaces' will in fact be machines on a computer network. Though the façade of the game may be different than a traditional interpretation, the core mechanics of the game and the structure of the Space base class will be the same as a more concrete implementation of spaces. The aesthetic reason for choosing this interpretation and abstraction is to implement my theme. After considering various alternatives, I have settled on designing the game as a superficial simulation of a computer virus. Indeed, I plan on having the player play *as* a computer virus navigating the network of the University of Oregon with the goal of crashing the UO webserver. This strikes me as inspiring on several levels: aesthetic, technical, and school pride.

Functions, Classes & Interactions

Class Hierarchy



Main

As in several of the previous projects, the main function will only serve as the hook into the proper top-level class. Main will simply instantiate an instance of the Game class and then call the play method. This allows the memory management to be encapsulated according to the RAII principle within Game. The play method will contain the main game loop, and therefore handle the overall execution and flow of the game.

Game Class

The Game class will be the structural core of the overall program. Within it, the game flow will be handled by the main game loop, itself within the play method. Given that there will be nested series of menus for each type of player interaction sequence (as well as numerous helper functions for informing the player of their status, creating visual effects, etc.), the main game loop will consist of the top-level game menu implemented as a switch. It will implement the diverse game functionality by calling out to other methods specialized to each unique action. For instance, one top level menu option will be to navigate to another device. Selection of this action will call a specialized method that handles the logic necessary to then present the player with a list of devices to navigate to, as well as handling cases in which options do not exist (e.g., when no connection to a device exists in a given direction).

Space Class

The foundation of game navigation and interaction is the Space class. As all network devices will inherit from it, it will necessarily contain all common functionality for the network devices. This includes, but is not limited to: methods capable of linking spaces; addition, removal, and tracking of files (DeviceFiles) present on the device; state concerning whether the device has been exploited; and a name and description for use in presenting the space to the player. As the Space class must be an abstract base class, it will also include a pure virtual method dedicated to implementing custom exploitation behavior in subclasses. This method will be called when the device is successfully exploited, and will therefore be used to generate a unique visual effect based on the subclass implementation.

iPhone, DesktopPC, Server, and WebServer Classes

As the majority of core space functionality will be implemented in the Space parent class, the various subclasses will be relatively simple. They will essentially consist of a destructor (necessary for runtime polymorphism, even though no memory is dynamically allocated within the class, and therefore nothing needs to be done in the destructor), and the specialization of the exploitation virtual method. As mentioned above in the description of the Space class, the exploit implementations will likely be used to present the player with a distinctive visual effect upon exploitation of the network device.

DeviceFile Class

The DeviceFile class is an addition to the requirements that will be necessary to implement the game as I envision it. Specifically, I intend to require the player to collect some number of special password files to complete the goal of taking down the OU webserver. Not all files will contain passwords; in fact, I plan to have most of the game’s tone and mood come from the names and contents of the various nonpassword files. To that end, the DeviceFile class will largely be a container to hold the filename and contents, as well as some Boolean flag to indicate the presence or absence of passwords in a specific instance. It will naturally have the appropriate getters and setter associated with these values.

Testing Plan

The game has several mechanics that present unique design and implementation challenges. Foremost among these is the construction and destruction of an interlinked series of space instances. The testing will involve both specific testing of memory handling procedures, as well as general analysis of memory management via the application of Valgrind during all testing runs, regardless of whether memory use is the specific test case being addressed. This will ensure that memory issues are squashed as soon as they crop up, and not only later as part of a dedicated memory-use testing cycle.

A key mechanic to be tested is player movement. Players should not be allowed to move past a device on the network without exploiting it, so the game must ensure there are no edge cases where the player may be able to circumvent this central restriction within the game. Testing will also cover handling the exploitation mechanic wherein the player attempts to exploit a device to progress through the network and steal the device's contents. This will be implemented as a dice roll, with various modifiers, compared against the relative difficulty of the device (an instance-specific member variable).

Testing of the file transfer mechanic will also be key. The player can steal a limited number of files from the network devices they have exploited. Specifically, they are incentivized to steal password-containing files as they boost the player's ability to exploit devices (by modifying their exploitation dice roll), and are in fact necessary for the player to win the game: a certain, minimum number will be required to be in the player's inventory to attempt an exploit of the UO webserver.

Testing Plan Table

Test Case	Values	Function Tested	Expected Outcomes	Observed Outcomes
Memory Management	Game network map configuration	Memory management -- allocation	All spaces allocated -- program completes with no memory leaks or errors detected by Valgrind	Program completes with no memory leaks or errors detected by Valgrind
	Game network map configuration	Memory management -- deallocation	All spaces deleted -- program completes with no memory leaks or errors detected by Valgrind	Program completes with no memory leaks or errors detected by Valgrind
	Game network map configuration	Memory management – navigation with no invalid reads	Navigation among all spaces -- program completes with no memory leaks or errors detected by Valgrind	Program completes with no memory leaks or errors detected by Valgrind
Player Navigation	Navigation from exploited device	Game.move()	Player can navigate to the target device	Player can navigate to the target device
	Navigation from unexploited device	Game.move()	Player cannot navigate to the target device	Player cannot navigate to the target device
File Transfer	Empty Inventory	Game.stealFile()	File is removed from the Space and placed in the player inventory	File is removed from the Space and placed in the player inventory
	Full Inventory	Game.stealFile()	File is not removed from space, not placed in player inventory, and error message presented	File is not removed from space, not placed in player inventory, and error message presented

Reflection

Technically, this project was unique from its predecessors only in its combination and extension of elements. The use of linked structures has already been dealt with in the course, and we have had the chance to create several implementations of these kinds of data structures. Similarly, the memory management requirements of this project are close to those of the various linked-list and queue based programs we have completed. However, while the elements were familiar, this project did represent a challenge of complexity. Whereas our previous implementations were 1-dimentional, the requirement of four directional connections (or at the very least, the *potential* for such connections) necessitated extending the learned concepts to a 2-dimentional space. However, by consistently extending the 1-dimetional analogs that I had already created, it was possible to retain fidelity in terms of memory management, modularity, and flexibility of implementation.

The testing phase for this project was rather smooth. By implementing the spaces first, linking them, and then testing them, I was able to iron out the kinks in the game's spatial abstraction up front. Since this aspect of the game was the most complex, it gave me plenty of time to focus on other aspects of gameplay. Specifically, it allowed me to ensure that the aesthetics of the rudimentary visual effects (i.e., text 'animations') were to my liking and that the various bits of flavor text were fleshed out.

I found this final project to be both the most enjoyable and the most challenging of the projects in this class. Interestingly, it was both fun and difficult for precisely the same reason: freedom of design. The open specification left much room for creativity and complexity. I experienced the highs and lows of both during this project. Once I settled on my theme, I was immediately energized and inspired to begin designing and implementing the game. Since I could choose the subject matter, I was much more inclined to take ownership of the project. Indeed, personal pride was a much larger component in this project than any other I have done in this class. Given the freedom of this project, it to a large degree reflects my personal tastes, style, humor, and interests. It's also just fun to be so meta.