Julian Torres
Professor Zhang
CS 162
09 December 2019
Final Project

Design, Test Table, and Reflection

Design:

For this project, I have created a survival game. I initially struggled to come up with a game concept because I had difficulty in understanding the requirements. I did not fully grasp how the Space class was to be implemented. After reading posts on the discussion board, I understood the spaces represented actual locations within a game, that they were to be linked together. I instantly though of how the linked lists were implemented, and I could not help but think about this project in terms of a container. After further digesting the instructions, I came to the conclusion that I could simply include four pointers to the Space class, in which every derived class object could point to four other objects, or null values. Once this framework was developed, I instantly thought about a game in which the player must escape an island that has an active volcano on it, most likely influenced by the fact that I live on Oahu.

After developing the broad goal of the game, I began working on the Space classes. Particular to my thought process and workflow style, I like to work on certain, known aspects of a project at a given time, in which I can perform some work while details are introduced throughout this process. Therefore, I continued to work on the Space classes. I would create the Ocean, Beach, Forest, and Volcano derived classes. This gave me some direction to work toward. While working on these classes, I began to work out specific gameplay details. I would stop working on the derived classes occasionally to work on a gameplay component, such as introducing a time limit element or energy point system.

Initially, I thought I would have a total of three classes, excluding the derived classes. I thought I would use the Space classes, a Player class, and a Gameplay class. I ended up only using the Space and Gameplay class, as I introduced a poor programming method while working on the Player class, which was later resolved. I ended up including a menu function as well.

Ultimately, the beginning of this final project was the most difficult aspect. I spent a few days just trying to absorb the instructions, to understand what was required, how I would implement the linked structure. After I understood what was needed. the project was straightforward, yet it took a long time to complete the legwork and debugging. I was unable to include a map in the given amount of time, but it is an aspect that I will later include, time permitting.

Pseudocode:

Initial Menu:

```
Prompt the user to play or quit the program
If user chooses to play
    Display the game description
Else
    Terminate the program
```

Space Class:

```
Space* northSpace
Space* eastSpace
Space* southSpace
Space* westSpace
addSpace(Space*, Space*, Space*, Space*)
Link spaces together using addSpace function
spaceInteraction(vector&, capacity)
Prompt the player to perform a specific action based on the derived class. Add or
remove items from the player's backpack vector.
```

Gameplay:

```
Build the map by creating spaces
Link the spaces together
Assign values to pertinent gameplay variable, e.g. maxEnergy
Display the current day
Display the contents of the backpack
Prompt the player to move to a new space
If the player chooses the ocean
    Check if the player has a boat
    If the player has a boat
        The player wins
    Else
        Return the player back to the previous space
Else
    Move the player to the respective space
spaceInteraction(vector&, capacity)
Depending on the space
    The interaction will vary
    The player may perform an action to obtain a new item
    Or the player may lose an item
Check the backpack vector for food items
If there is food and the energy level is at a specified level or lower
    Consume food
Display the contents of the backpack
If there is at least one item in the backpack and three health points
    Prompt the player to attempt to build an item
    If yes
        Attempt to build and notify the player if an item was built
        Expend one energy point no regardless of outcome
Expend one energy point
Display end of day information
If there is food and the energy level is at a specified level or lower
    Consume food
Increment the day count
Repeat loop until the player wins, the volcano erupts, or the player runs out of
energy
```

Follow-up Menu:

```
Prompt the user to play or quit the program
If user chooses to play
    Play again
Else
    Terminate the program
```

Test Table:

Test Case	Input Value	Driver Function	Expected Outcome	Observed Outcomes
Input is not a valid entry	Input == “this is a test”	While(input != “1” && input != “2”)	Continue prompting user until a valid input is entered	Continue prompting user until a valid input is entered
The player attempts to enter the ocean without a boat	movePlayer(Ocean)	checkOcean();	If the player does not have a boat, the player is returned to the previous space.	If the player does not have a boat, the player is returned to the previous space.
The player has no items in the backpack	DisplayBackpack()	If(backpack.size() == 0)	Notify the user there are no contents in the backpack	Notify the user there are no contents in the backpack
The player’s backpack has reached the maximum capacity	checkBackpack()	If(backpack.size() == capacity)	Display the contents, and prompt the user to use or remove an item	Display the contents, and prompt the user to use or remove an item
The player has the necessary items to build an item	buildItems()	If(itemA != -1, itemB != -1, itemC != -1)	Check if there is a valid index value associated with the item. Create the item and modify the contents of the backpack	Check if there is a valid index value associated with the item. Create the item and modify the contents of the backpack
The player has three energy points and at least one item in the backpack	buildItems()	If(energy > 2 && backpack.size() >= 1)	The player will be prompted to attempt to build an item	The player will be prompted to attempt to build an item
The player’s energy points are less than or equal to two	checkStatus()	If(energy <= 2)	Notify the player to start searching for food	Notify the player to start searching for food
The player has zero energy points	checkStatus()	If(energy == 0)	Notify the player that she or he has lost the game	Notify the player that she or he has lost the game
The player reaches the time limit	checkStatus()	If(daysElapsed == eruption)	Notify the player that she or he has lost the game	Notify the player that she or he has lost the game
The player has created the boat and moved to the Ocean space	movePlayer(Ocean)	if(backpack.at (index) == “Boat”)	Notify the player that she or he has won the game	Notify the player that she or he has won the game

Reflection:

I typically do not like to create games – I often dread having to do so. However, after the initial struggle to understand the implementation of this project had passed, I enjoyed creating this game. There were only two real challenges throughout the entire project, and one small error in the end, which was just a minor design flaw, easily corrected.

The first challenge of this project, as previously noted, was with the linking of Space objects. I could not help but think about the linked list and queue assignments. I was under the impression that spaces had to be linked together in some dynamic manner; in which during gameplay, these spaces would change. I wrestled with this concept for quite some time, and I could not understand the instructions. However, I ultimately came to the conclusion that I could declare pointers in the Space class, and the objects would be hardcoded in some segment of my Gameplay class. This idea was further encouraged after seeing a post on the discussion board, in which a student asked if the spaces could be linked by hardcoding them at compile time. The answer was a resounding yes. I was thrilled, and I could actually begin focusing on the implementation.

I initially wanted to create a Player class, a Space class, and the Gameplay class. However, there was a huge flaw in this idea. My Player class would include the Space class, where the Player could be assigned to a starting space. This was not a problem yet. The Space class would then include the Player class, so the Space class could access functions belonging to the Player class. The Gameplay class would need information from both classes. This was a complete nightmare, as I never encountered this problem before, and it led to files not being read by one or the other class. I ended up separating the classes dependencies on one another, and I took it a step further. I ended up scratching the Player class because the only pertinent variables and functions that belonged to the Player were centered on the string vector. This would simply stored in the Gameplay class, as an entire class did not need to be created for this one aspect, as it would require extra getter and setter functions, which would ultimately be bulky and unnecessary. This new design allowed me to operate with the Gameplay class, in which it included the Space class and its derivatives. The Gameplay class would hold pointers to the necessary space class objects to build the map, and it would keep track of the days elapsed, the energy level of the player, and the player's backpack.

The only other problem I experienced was with my menu. I wanted to have the Gameplay class be fully self-contained, but this was poorly executed. The menu would run in the driver program until the player decided to quit the program after a game finished. However, because the Gameplay object was already created prior to the menu being ran in the while loop, specific game variables were not updated at the start of new games. This was fixed by removing the menu from the Gameplay class. The menu would be a standalone function, including the Gameplay header file.

Ultimately, this was a fun way to end the program. I learned a few very important lessons from this final project. Particularly, I learned that understanding the instructions is crucial. It goes without saying, but it is still worthwhile to mention. Second, I learned that classes should not depend on one another; this concept of nondualism can be saved for spirituality. Third, intricate games can be developed using basic concepts, conditional statements and loops. I think it absolutely paramount to understand the fundamentals of programming, as opposed to skipping past these building blocks to learn a complex operation.