Name: Brendan Corazzin

Assignment: Final Project

Date: 6/11/2017

Reflection

For this project, I took a slightly different approach than all past projects for this course. I was really struggling with understanding how I would implement the requirement that each room/space contain four pointer variables (or an array of pointers) to other rooms/spaces. I had a tough time understanding how the rooms would connect and how that would work in the code. So, started working on the project in the same way that I typically do: I wrote the program design and the class diagram. I then wrote the pseudocode for the functions found at the bottom (the play() and spaceInfo() functions). After writing this material, I still found that I was totally stumped on how to meet the requirements regarding the use of pointers to other space/room objects. So, I began making files for each of the classes. Once that was done, I wrote out the play() function and then the spaceInfo() function. As I wrote the spaceInfo() function, I started to see the need for other functions inside of the Player and Pack classes, so I started writing those as well. This is different than what I normally do, as I usually write out all of the pseudocode for all the of functions I think I need before I even start writing code.

Throughout the process of writing and testing code for the Player, Pack, and Game class, it started to become clear to me that me that the Space objects would need be instantiated within the game class so that they would persist throughout the entirety of game play. The Player object would also need to be instantiated in the Game class as well. Then the necessary pointers for each space could be passed into other spaces in a structured way that provided context for the game (up, down, left, and right). The initSpaces() and initPtrs() functions were written to accomplish this. Once I wrote these functions, I saw that the Player object could then be passed into different spaces as determined by the user. As this became clear, I wrote code without writing pseudocode. Instead, I wrote basic notes at the top of each class as I worked through each problem I encountered. I know this practice isn’t ideal, but it honestly was the only way I could figure out how to tie all the classes together in a way that worked. In the class diagram below, the functions written in orange text are the functions I wrote during the development of the game.

At the end of development, I ran through the testing table below and only had to make a few minor changes to the game so that all the expected results matched the actual results. At the time of writing this paper, I am satisfied with the code I wrote, although I see room for improvement. For example, the Player and Pack classes interact in kind of an odd way. Most of the functions in Player that access the Items vector simply call a function in Pack. I did this because the compiler was giving me errors about accessing the Items vector directly from the Player class since it was a private data member of Pack. I tried a few work arounds but due to time pressures, I ended up writing the functions as the appear now. I’m not totally sure how I could improve this code (it’s probably and easy and obvious fix), but I still feel like there is a way to clean things up. I also considered adding functionality to some of the items. For example, I wanted to make it possible for the player to deposit items in various locations and then come back to those items and pick them up again later. I’m totally sure how I would have accomplished this, but I ran out of time and decided to just leave the game as it was. I also considered writing an additional class called Edge. This class would have derived from the Space class and contained the code for the edges of the spaces (boulders, spring, cliff, etc.), but I had everything up and working as I intended without doing this, so I decided against it. If there were more time, I likely would have made some of these changes, but I was satisfied just getting everything working properly while also conforming to the assignment specifications. All things considered, this was definitely my favorite assignment for this class and I really enjoyed working on it.

Program Design

* The problem to be solved: develop a text based game where the player travels through spaces to win the game. The game must have the following elements:
  + The game must have three different types of spaces and at least six total spaces
  + Each space must have four pointers that link it to other spaces
  + The player will be able to collect items as they move through the rooms
  + The player must have a container to hold items and the container must have a limit on how many items it can hold
  + The player must interact with the spaces, not just move through them and collect things
  + There must be some sort of time limit on the game
  + Game requires a menu and the menu must explain the game and the goal
* Game description: The player wakes up stranded on a tropical island. When they wake up, they are laying on the island’s south beach. The only thing they have with them is a backpack. The object of the game is to find a way off the island. There are 6 different spaces on the island: the south beach, a forest, a river, the north beach, a temple, and the west beach. All of the spaces are completely unique with the exception of the beaches. The north beach and the south beach are almost the same. On the west beach, there is a fishing boat with a small crew that the player can use to leave the island, but the temple serves as a barrier to this beach. The player can collect objects from different spaces. The objects must be used to make it off the island. The player also will face different scenarios where they die, including running out of food and/ or water.
* Identify the inputs:
  + User will input ‘y’ or ‘n’ to make decisions
  + User will input ‘u’, ‘d’, ‘l’, or ‘r’ to move between space objects and within space objects
* Desired output:
  + Main menu will prompt user to continue or quit
  + User will be given option to look in backpack throughout game (this will make it easy for myself and the grader to test if items are indeed being added or removed from the player’s items vector and to verify that the vector persists throughout the game)
    - Contents will be printed
  + User will be given option to move throughout the game
    - Info about the space objects will be printed
  + User will be given options to interact with the spaces
    - Info regarding the results of the user’s actions will be printed
  + Print message if player dies or if player wins the game

Class Diagram

**Player Class**

Pack obj

getStrength()

getWater()

setStrength(int)

refillWater()

printInfo()

checkItems(string check)

packIsFull()

removeItem()

deleteItem(string)

addItem(string)

update()

useItems()

**Pack Class**

Items vector

printContents()

getCapacity()

getItem(int)

isFull()

checkItems(string)

discard(int)

add(string)

deleteItem(string)

**Space Class**

printInfo()

getSpaceName()

spaceInfo()

initPtrs(Space\*, Space\* Space\*, Space\*)

**Game Class**

Player obj

play()

initSpaces()

~Game()

has-a

has-a

has-a

is-a

is-a

is-a

**Temple Class**

spaceInfo(Player\*)

**River Class**

spaceInfo(Player\*)

**Forest Class**

spaceInfo(Player\*)

**Wbeach Class**

spaceInfo(Player\*)

winGameMssg()

**Nbeach Class**

spaceInfo(Player\*)

**Sbeach Class**

spaceInfo(Player\*)

is-a

is-a

is-a

Testing plan:

The testing plan for this project is broken down into multiple tables that deal with independent functions and classes. The function or class the testing table corresponds to is at the top of each table.

|  |  |  |  |
| --- | --- | --- | --- |
| Table 1: main function | | | |
| Input | Expected Output | Actual Output |  |
| NA | Function loops so long as menu() doesn’t return ‘q’ | Function loops until menu() returns ‘q’ |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Table 2: menu function | | | |
| Input | Expected Output | Actual Output |  |
| ‘c’  ‘q’  Enter  Spacebar  100  “hello” | Returns ‘c’  Return ‘q’  Prints error message  Prints error message  Prints error message  Prints error message | Returns ‘c’  Returns ‘q’  Prints error message  Returns blank char  Prints error message  Prints error message |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Table 3: validateInput function | | | |
| Input | Expected Output | Actual Output |  |
| “hello”  ‘c’  enter  1  100  -1 | Error message, return 1  Error message, return 1  Error message, return 1  Change value, return 0  Change value, return 0  Change value, return 0 | Error message, return 1  Error message, return 1  Error message, return 1  Change value, return 0  Change value, return 0  Change value, return 0 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Table 4: getChar function | | | |
| Input | Expected Output | Actual Output |  |
| ‘c’  ‘q’  Enter  Spacebar  100  “hello” | Returns ‘c’  Returns ‘q’  Prints error message, prompts again  Return space  Prints error message, prompts again  Prints error message, prompts again | Returns ‘c’  Returns ‘q’  Prints error message, prompts again  Return space  Prints error message, prompts again |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Table 5: getYorN function | | | |
| Input | Expected Output | Actual Output |  |
| ‘y’  ‘n’  Enter  Spacebar  100  “hello” | Returns ‘y’  Returns ‘n’  Print error, prompts again  Print error, prompts again  Print error, prompts again  Print error, prompts again | Returns ‘y’  Returns ‘n’  Print error, prompts again  Print error, prompts again  Print error, prompts again  Print error, prompts again |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Table 6: Game class | | | |
| Function | Input | Expected Output | Actual Output |
| play() | NA | Print welcome message  Loops until strength or water is 0 or currSpace is nullptr  Print appropriate message if strength or water = 0 | Performance matches expected output |
| initSpaces() | NA | Calls initPtrs() and passes params | Calls initPtrs() and passes params |
| Game() | NA | Creates space object pointers | Creates space object pointers |

|  |  |  |  |
| --- | --- | --- | --- |
| Table 7: Player Class | | | |
| Function | Input | Expected Output | Actual Output |
| update() | NA | Decrements water and strength by one | Decrements water and strength by one |
| printInfo() | NA | Prints correct info for player object | Prints correct info for player object |
| useItem() | String | Returns correct item  Deletes item from pack | Returns correct item  Deletes item from pack |

|  |  |  |  |
| --- | --- | --- | --- |
| Table 8: Pack Class | | | |
| Function | Input | Expected Output | Actual Output |
| checkItems() | string | Compares string to strings in items vector and returns true if match is found | Performs as expected |
| isFull() | NA | Returns true if 4 items are in items vector | Performs as expected |
| getItem() | Int | Returns element at the items index value | Performs as expected |
| discard() | int | Deletes item at items index value | Performs as expected |
| deleteItem() | string | Deletes item from items that matches string name | Performs as expected |

|  |  |  |  |
| --- | --- | --- | --- |
| Table 9: Space Class | | | |
| Function | Input | Expected Output | Actual Output |
| initPtrs() | 4 pointer to space objects | Load pointers into spaces array | Performs as expected |

|  |  |  |  |
| --- | --- | --- | --- |
| Table 10: Sbeach Class | | | |
| Function | Input | Expected Output | Actual Output |
| Sbeach() | NA | Loads correct data into data members | Performs as expected |
| spaceInfo() | Pointer to player object | This function advances the player through the space and into other spaces. Test if:  correct if statements execute for each option provided to the user;  ensure stone and numFish variables update and remain updated through game play (i.e. if the player leaves the room and returns);  ensure all other functions execute properly. | Performs as expected |

|  |  |  |  |
| --- | --- | --- | --- |
| Table 11: Forest Class | | | |
| Function | Input | Expected Output | Actual Output |
| Forest() | NA | Loads correct data into data members | Performs as expected |
| spaceInfo() | Pointer to player object | This function advances the player through the space and into other spaces. Test if:  correct if statements execute for each option provided to the user;  ensure all other functions execute properly. | Performs as expected |

|  |  |  |  |
| --- | --- | --- | --- |
| Table 12: River Class | | | |
| Function | Input | Expected Output | Actual Output |
| River() | NA | Loads correct data into data members | Performs as expected |
| spaceInfo() | Pointer to player object | This function advances the player through the space and into other spaces. Test if:  correct if statements execute for each option provided to the user;  ensure gun variable updates and remains updated through game play (i.e. if the player leaves the room and returns);  ensure all other functions execute properly. | Performs as expected |

|  |  |  |  |
| --- | --- | --- | --- |
| Table 13: Nbeach Class | | | |
| Function | Input | Expected Output | Actual Output |
| Nbeach() | NA | Loads correct data into data members | Performs as expected |
| spaceInfo() | Pointer to player object | This function advances the player through the space and into other spaces. Test if:  correct if statements execute for each option provided to the user;  ensure disc and numFish variables update and remain updated through game play (i.e. if the player leaves the room and returns);  ensure all other functions execute properly. | Performs as expected |

|  |  |  |  |
| --- | --- | --- | --- |
| Table 14: Temple Class | | | |
| Function | Input | Expected Output | Actual Output |
| Temple() | NA | Loads correct data into data members | Performs as expected |
| spaceInfo() | Pointer to player object | This function advances the player through the space and into other spaces. Test if:  correct if statements execute for each option provided to the user;  ensure pouch and doorOpen variables update and remain updated through game play (i.e. if the player leaves the room and returns);  ensure all other functions execute properly. | Performs as expected |

|  |  |  |  |
| --- | --- | --- | --- |
| Table 15: Wbeach Class | | | |
| Function | Input | Expected Output | Actual Output |
| Wbeach() | NA | Loads correct data into data members | Performs as expected |
| spaceInfo() | Pointer to player object | This function advances the player through the space and into other spaces. Test if:  correct if statements execute for each option provided to the user;  ensure bottle and firstEncounter variables update and remain updated through game play (i.e. if the player leaves the room and returns);  test if rand() is working correctly (~50% of each option occurring when selected);  ensure all other functions execute properly. | Performs as expected |

Pseudocode

/\*

In function play

do:

call getSpaceName() (member of space) and print

Print player stats

Ask user if they want to look in their backpack

if y - print contents, else continue

call spaceInfo() (member of space)

while: spaceInfo() == 1 and water > 0 and strength > 0

In spaceInfo()

print info on space

prompt user to interact with space, print info on edges and paths

walk to edge?

if yes, walk to edge

if death

end game

else if item

print "room in bag"

if full

prompt to swap

if yes, swap, else leave item in place

else

prompt to take item

if yes, add to bag, else leave in place

walk back to space

call spaceInfo()

walk down path?

update player info based on selection

space player is in

food amount

water amount

return 1

\*/