| | |
|---|---|
| Document Name: | **FinalProject_Design_Reflection.pdf** |
| Author: | Will Clayton |
| Date: | 2017-12-05 |

# Design: Final Project: "A Text-based Game"

## Contents

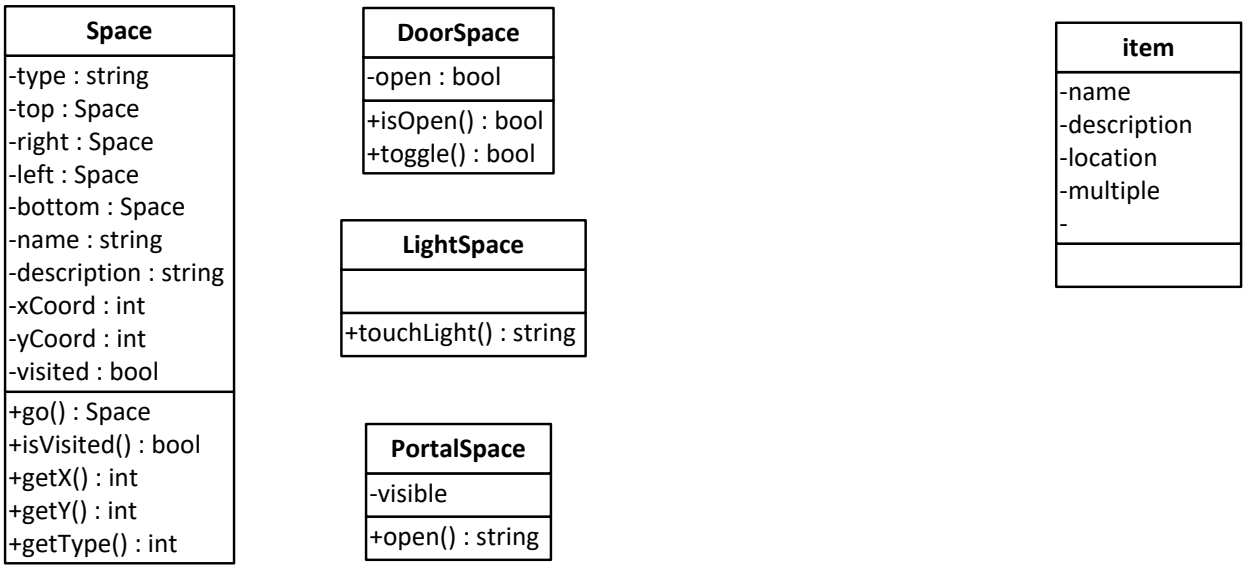


## Introduction

In our introduction to the course, we received advice to not focus on making fun programs that people would like to play – but to focus on meeting the project objectives. I took this to heart for my previous Project assignments, but really let myself slip on this one. It doesn't help that I grew up on text adventure games, and even tried to write one in Turbo Pascal many, many years ago (it wasn't good). The very tight timeframe for this project and large scope left me struggling to balance the project with my own desires. In the end I was able to re-focus, tighten the scope of the program, and am very happy with the results!

## Class Diagram

Here is the initial class design diagram as rendered by Microsoft Visio:




I should have done much more design work up-front, including use cases for each of these classes, which would have revealed the need for additional planning and up-front work. In particular, I should have focused on which objects should manage what happens to them purely internally and which would require data from outside of themselves for key behaviors (I put in locked doors, but forgot that even though a door knew who its neighbor(s) were, it wouldn't necessarily be able to open its partner's connecting door when its own had been opened. This had some funny side-effects, such as doors that were open in one direction only – and I probably should have made "door" a class that connected two Space objects and could deal with them both.

## Design Description

CLASSES & FILES:

- makefile – Program makefile
  - Compile all files
  - Phony to execute Valgrind on result
  - Phony to zip files for submission
- wcLibrary [Standard functions I have developed for reuse over the semester]
  - validInteger – Return a valid integer between two bounding values
  - validFloat – Return a valid float between two bounding values
  - marquee – Wrap a message in stars and display it
  - doOrNot – Prompt and select a binary decision (e.g. Play/Quit)
  - getInteger – Return valid integer between two values with prompt
  - getFloat – Return valid float between two values with prompt
  - randomInt – [Deprecated – use newRandomInt]
  - getString – Prompt for and return a string up to a specified maximum length
  - getChar – Prompt for and return a single character from a provided set
  - newRandomInt – Return a random integer between two bounding values

    - appendFileName – Append a supplied string before a filename suffix
    - stringOrDefault – Prompt for and return a string, or a supplied default value
    - intAsString – Simple conversion from integer to string since I use this library anyway
    - pause – Pause until user input
- Space – Abstract Base class for all game locations
    - Variables (protected):
        - string type – which type of child function is being represented
        - Space* top – the location to the Space's north
        - Space* right – the location to the Space's east
        - Space* left – the location to the Space's west
        - Space* bottom – the location to the Space's south
        - string name – the name of the Space
        - string description – a description that will tell the player about the Space, hopefully in ways that will feel like you're really there
        - string specialMessage – any special conditions that should be called out to the player
        - bool upsideDown – if the Space is in the "upside-down" parallel universe, just being there hurts the player
        - bool showDescription – Allows the game to provide an abbreviated description of Spaces you've already visited
    - Functions (public):
    - Space() Constructor
    - void updateDescription(string) – set the description value
    - string getType() – get the (sub)class type
    - string getName() – get the name assigned to the Space
    - string getDescription() – get the Space description
    - void setMessage(string) – Assign a special message to the Space
    - bool hasMessage() – Check if a Message is assigned
    - string getMessage() - get any special message assigned to the Space
    - Space* go(Space*, string) – the Space evaluates its neighbors and, if one exists in the indicated direction, it returns its pointer location to be the new "current location"
    - void setNorth(Space*) – Assign a neighbor to the north
    - void setEast(Space*) Assign a neighbor to the east
    - void setSouth(Space*) – Assign a neighbor to the south
    - void setWest(Space*) – Assign a neighbor to the west
    - void setUpsideDown(bool) – Mark the Space as "upside-down"
    - bool isUpsideDown() – see if the Space is "upside down"
    - void setShowDescription(bool) – Set whether the space should show its description or not
    - bool doShowDescription() – see if the Space should show its description or not
    - virtual string action(int) – child class-specific actions
    - virtual string spacer(int, Space*) – child class-specific actions
    - virtual string stringer(int, string) – child class-specific actions
    - virtual ~Space() - Destructor

- NormalSpace – Uses only parent class behaviors (but can be used because it's not abstract)
- DoorSpace – Spaces that conditionally connect to others in the map
    - Lock the door
    - Unlock the door
    - Hold the door open under special circumstances
    - Determine whether the door is open
    - Set the Space as having an east door
    - Set the Space as having a west door
- LightSpace – Special space in which a character in the "upside-down" can cause electric lights in the real world to flash (and thus influence special events)
    - Touch the lamp to cause the behavior
- PortalSpace – Spaces in which object on other parts of the board can be accessed through 'portals' that are inside of containers that can be opened or closed
    - Open the container
    - Close the container
    - Control specific "get" behavior for these items
- Item.hpp – Things that can be had in the game. Some required to win; others extra
    - Variables (private)
        - Name – String describing the item to the player
        - string word – single word by which the player interacts with the item
        - Space* location – the primary location of the object
        - Space* portalLocation – secondary location from which the object may be accessed via a 'portal'
    - Functions (public):
        - Item(string, string) - Constructor
        - string getName() – get the name of the item
        - string getWord() – get the item 'word'
        - void setLocation(Space*) – set the item primary location
        - Space* getLocation() – get the item primary location
        - void setPortalLocation(Space*) set the item 'portal' location
        - Space* getPortalLocation(); - get the item portal location
- Game – The primary logic and control for the program
    - Variables (
        - wcLibrary* lib – my standard functions I've worked up over the semester
        - Space* location[6][7] – A 2D array of Space pointers that I use to interact with the linked grid of Spaces. This made it easy to initialize them, to do memory clean up in my destructor, and to randomly access spaces without having to iterate through them looking for the one I want. But the Space objects themselves store the map for the game.

The following Spaces define the game world (green Spaces are 'upside-down' and white spaces are in the 'real world'):

| | | | | | |
|---|---|---|---|---|---|
| [0,0] | NW Hall & Lounge [1,0] | North Hall [2,0] | NE Hall [3,0] | Intern Restroom [4,0] | [5,0] |
| [0,1] | West Hall | Break Room [2,1] | East Hall [3,1] | Intern Farm [4,1] | [5,1] |
| [0,2] | SW Hall | South Hall & Cooler [2,2] | SE Hall [3,2] | [4,2] | ,2] |
| Unknown Hallway [0,3] | Unknown Hallway [1,3] | Unknown Hallway [2,3] | Unknown Hallway [3,3] | Unknown Hallway [4,3] | U wn H ay |
| Applied Psychiatrics Lab | NW Hall & Lounge [1,4] | North Hall | NE Hall | Intern Restroom [4,4] | Utility Corridor [5,4] |
| Lobby [0,5] | West Hall [1,5] | Break Room [2,5] | East Hall [3,5] | Intern Farm [4,5] | Utility Corridor [5,5] |
| Security Entrace Lobby [0,6] | SW Hall | South Hall & Cooler [2,6] | SE Hall | Breaker Room [4,6] | Utility Corridor [5,6] |

- Space* exit – Special Space marking the end of the game
- Space* currentLocation – where the player is now
- Space* demogorgonLocation – Monster location
- bool demogorgonSleeping – whether the monster is active
- string result – A description of what happened following the player's most recent action

    - Item* inventory[5] – array of game items that are out on the map
    - Item* used[2] – Array of game items that have been used and can no longer be accessed
    - Item* hand[2] – Array of items the player is currently holding (2)
    - bool accomplishment[7] – array of game objectives the player has met
        - Eat Eggo
        - Drink Coke
        - Open Door
        - Get Key
        - Reset Breaker
        - Use Terminal to Submit Code
        - Escape from the facility
    - int health – player's current health value (0 = dead)
    - int score – players current score (all accomplishments = 100)
    - Functions (public)
        - Game() – initialize the game
    - string parseInput(string) – identify a valid verb (and noun) for player input; respond accordingly
    - void play() – control the flow of the game, including reactions and end-states
    - ~Game() - destructor
- strangerThings.cpp – Main()
    - Simple container to call the game
        - Create the game
        - Play the game
        - << I probably should have included an option to play again! >>

## Test Plan

| Operation | Driver Functions | Test Case | Input | Expected Outcome(s) | Observed Outcome(s) |
|---|---|---|---|---|---|
| Player Input | WLCMenu.getString(), | Input Blank | "" | Reject and re-prompt | As expected |
| Player Input | WLCMenu.getString(), | Input too long | 1234567890 1234567890 1234567890 12345678901 | Reject and re-prompt | As expected |
| Player Input | WLCMenu.getString(), parseInput() | Contains con-text characters | G3et | Strip characters and process | As expected |
| Player Input | WLCMenu.getString(), parseInput() | Leading Spaces | " get" | Strip characters and process | As expected |
| Player Input | WLCMenu.getString(), parseInput() | Multiple spaces between verb and noun | "go e" | Strip characters and process | As expected |
| Player Input | WLCMenu.getString(), parseInput() | Extra words after verb and noun | "go e extra" | Ignore extra words and process | As expected |
| Game Logic | WLCMenu.getString(), parseInput(), play() | Health drops each turn in "upside-Down" | {Play} | Drop each turn | As expected |
| Game Logic | WLCMenu.getString(), parseInput(), play() | Player out of health | {Play} | Die at health=0 | As expected |
| Game Logic | WLCMenu.getString(), parseInput(), play() | Boost player health | Use Eggo, Use Coke | +health | As expected |
| Game Logic | WLCMenu.getString(), parseInput(), play() | Boost player health (cheat) | "cheat" | +health to 100 | As expected |
| Game Logic | WLCMenu.getString(), play(); DoorSpace() | Lock Door | Attempt to Traverse locked door | Treat as Wall | As expected |
| Game Logic | WLCMenu.getString(), play(); DoorSpace() | Unlock Door | Attempt to Traverse unlocked door | Treat as Passage | As expected |
| Game Logic | WLCMenu.getString(), play() | Leave without completing game objective(s) | Leave without lunch, code submit | End game – lose | As expected |
| Game Logic | WLCMenu.getString(), play(); PortalSpace() | Open Portal | Open fridge | Contained item available to get; Update Description | As expected |
| Game Logic | WLCMenu.getString(), play(); PortalSpace() | Close Portal | Close fridge | Contained item not available to get, Update Description | As expected |
| Game Logic | WLCMenu.getString(), play(); PortalSpace() | Open (Empty) Portal | Open Empty fridge | New description, Contained item not available to get | As expected |
| Game Logic | WLCMenu.getString(), play(); LightSpace() | Light Touch results | Touch Light | Unlock Door, Update Space descriptions with guidance | As expected |
| Game Logic | WLCMenu.getString(), play(); LightSpace() | Second Touch – No Results | Re-Touch Light | Custom Message | As expected |
| Game Logic | WLCMenu.getString(), play() | Use Special Objects | Use Terminal, Use Button | Update Descriptions, Score, Accomplishments | As expected |
| Game Play | Complete Game | All Objectives Met | Leave, Objectives met | WIN Message, Terminate Program | As expected |
| MEMORY LEAKS | Main, Ant, WLCMenu | Execute using Valgrind | `valgrind --tool=memcheck --leak-check=yes --track-origins=yes` | `"no leaks are possible" "0 errors from 0 contexts"` | As expected |

# REFLECTION:

## Original Design

As I mentioned previously, I went way overboard on the game, and really should have focus on the programming exercise. That said, I was able to do some really neat things with the game. Using the linked list of spaces allowed me to create non-linear game flow, and even an overlapping area to represent the same Spaces over time. The complexity of the design I attempted warranted a lot more architecture work, and the implementation would have gone more smoothly if I had. I probably also should have encapsulated more functionality into standalone functions or objects. The game came out really neat!

I really didn't want to mess with vectors, etc. but probably would have done better. I used simple arrays to hold finite lists of things, and ended up needed to iterate through them manually much more than I expected. That design also means that were I to expand the game I'd have to remember all the places I hard-coded these numbers, and change them there. Not the most elegant way to do it.

## Changes

The game was supposed to revolve around the "demogorgon" monster from the show, but as time ran out I realized that, though he would help the game, he didn't really contribute to the project requirements. So he gets to stay home and the adventure became a more simple themed escape room. I sure would have liked to include that monster, though!

I ended up simplifying a lot of the game logic. Instead of figuring out what you want to open or close, it just opens or closes whatever is in the room with you – since the rules currently allow only one openable thing per space.

I also simplified the portal logic so that you could grab items through a portal but couldn't put them back in/through. It wasn't necessary to the game play, and would have been a lot more code.

The original design called for doors that close automatically behind the player, but these also weren't necessary for game play, and would have been quite a bit more complicated. The way my game clock worked (you get sick and die over time) it made sense to keep them open as well.

## Problems Encountered

The 'web' of spaces linked to one another in two dimensions made it really easy to get the pointer addresses wrong and cause segmentation faults. At first I only initialized the spaces I knew I'd be using, which also cause some issues when I managed to forget which ones these were.

I should have absolutely fleshed out the class interfaces. With this many classes all working together, it was too easy to get something subtle wrong and really mess things up. I encountered some errors with this program that I'd not seen before. The program, or Valgrind, would fail and simply never come back. I never figured out quite what caused it, but it wrecked the entire session. Hopefully I didn't leave any sort of artifacts from those adventures on *Flip*.

The real challenge, though, was too much scope for the time allowed. This is an important lesson for any programmer, and highlights the need for some sound project management to go with one's programming skills.

## What I Learned

I ran into a bunch of places where a good ol' ternary operator would have chopped a number of lines from my code. I am playing nice in class, but in real life I'm kind of a fan. This is mostly because long ago when I first ran into it I was completely flummoxed and had to ask my little brother to explain it. He is a good explainer.

On numerous occasions I have forgotten how critical the order of steps is. Specifically I would NULL a pointer and then attempt to use it to access an object. For some reason I have this trouble with pointers and cause segmentation faults.

I really had fun with the assignment, and would have liked to been working on this the entire semester (I guess our combat games were kind of like this, but needed a lot more work.

I need to focus on specific requirements, include only as much passion as the job requires, and get it done more quickly. This was a great capstone for this course!

[1]

[1] Picture Rights: https://nerdist.com/wp-content/uploads/2017/10/Demogorgon-featured.jpg