CS162 Final Project – 'Till the End of Time

Michael Ton

Design & Reflection

Design Outline (original)

Spaces (abstract)

- (at least) four pointers to Space: top, bottom, right, left
- Game "map" made by using linked spaces
- Game ends upon finding both halves of gem and mount
- **three** derived classes from Space that need to have special interactions
    - Combat
        - 3 goons – two hold the fragments
    - Scavenge (1)
        - Upon discovering this space, player searches and receive the mount for the gem fragments
    - Free -> camp/rest or meditate if not done in previous X turns
        - rest: restore HP, lose time
        - meditate: increase two stats by one, lost time. Must rest after

Item

- variables for name, item type
- use enum to determine type?
- potentially items to heal, maybe buff items using a temporary version of train/meditate function

Character (abstract)

- name, combat stats (atk, def, etc)
- array of item pointers to contain inventory
    - functions to add/remove items, Boolean if inventory is full?
- display stats to user
- stat growths? maybe make variables for growths and use RNG system to determine level up (Fire Emblem style)
- **two derived classes:**
    - Player/Hero
        - probably needs to contain member variables for location/movement
        - handle/track win condition?
    - Enemy
        - will not move -> bound to Combat space
        - initialize inventory to represent potential loot

Linked Spaces

- Start user in bottom left

- Randomize integer when moving into a space, set space to be type based on integer?
- Or should I hardcode map?
- re-link all spaces at end of each map?

if random:

- after map initialized, pass through each element in array
- read spaces example: if right (row, col+1) != nullptr
    - if(col+1 < cols)
        - board[row][col]->setRight = board[row][col+1];
    - else
        - right is nullptr (should be already initialized by default)

Design

Because the game operates using linked spaces to form the board, changes to any spaces will need to cause the linked spaces to be re-initialized to maintain changes. Perhaps, if a space is discovered, a certain character can represent that, and undiscovered spaces will appear as a blank space or '?'. Either way, I will aim to make some sort of function that can be called in a loop while the game runs to ensure consistent re-linking of the space pointers. Ideas for Space derived classes has been difficult, but I will go with a combat, a "blank" space where the player can choose from multiple options, and a scavenge type where the player will receive a lore related item – potentially change this so the player has to search in these spaces for the item and use RNG to determine if they find it, or find another item like a potion.

In past projects/assignments, I use a Game class to effectively run the game, basically including all needed files and calling functions to make it work, so I will implement something similar here. It will probably use Player class members to establish position of the player and move them around the board. It makes more sense for Game to track turn count and limit, so I will place that in Game. I want to place spaces randomly on the board as opposed to a hard coded map, so the board will be equal dimensions and I will use a method similar to my group project to randomly place different spaces. I think it will be best to place the player in a pre-determined spot, and randomly place around it. Randomly placing the player might be weird from functionality standpoint, and probably story standpoint.

I will re-use my own Menu and Utility class per usual. Menus will be made to also display help/information to user during events if they choose to do so. Player status needs to be kept, so maybe menu could be used here? Or just call a Character function by Game class?

Since I have a combat space, I will likely implement a separate class that handles all combat related functions that can just be called by the combat space type, though it might make sense to include it in Combat itself? Not sure, seems like that may be a lot of code to put in a space compared to the others. The other spaces should really only have a function to perform the "event" of a space. Defeated enemies should drop loot, and they may potentially have multiple drops with different drop rates, or maybe just one guaranteed drop.

Potentially implement a space to “perform” the win condition, but I think it would be easier to just implement a victory once conditions are met at any point. In the past, I have struggled with memory leaks, so I will focus on establishing a good basis before adding in “accessories”.

Test Plan

Input/What is Being Tested	Expected Outcome	Actual Outcome
Player movement, linked spaces	Player moved properly and displayed accordingly on board, spaces not affected by player movement unless interacted with, no loss of memory/lost information, spaces update properly	Player moved properly and displayed accordingly on board, spaces not affected by player movement unless interacted with, no loss of memory/lost information, spaces update properly
Battle; player dies	Player dies, death “screen” displayed, end game, memory handled	Player dies, death “screen” displayed, end game, memory handled
Battle; player runs	Player goes to next turn and moves to another space; monster heals to full; monster has full health upon re-encounter	Player goes to next turn and moves to another space; monster heals to full; monster has full health upon re-encounter
Battle; monster dies	Monster ‘deleted’, drops loot, loot added to player inventory, inventory/status updated and maintained through rest of game	Monster ‘deleted’, drops loot, loot added to player inventory, inventory/status updated and maintained through rest of game; memory leak upon end of game (monster item pointer not handled)
Search; player receives item	Player receives item in inventory upon visiting space; re-visiting does not add item again and displays different message	Player receives item in inventory upon visiting space; re-visiting does not add item again and displays different message
Search; player never gets item	Player never visits space, game ends/loss without important item. No memory leaks	Player never visits space, game ends/loss without important item. Memory leak: did not handle item pointer properly.
Free; player ‘awakens’	Player levels up two stats randomly, moves next turn and stats not lost	Player levels up two stats randomly, moves next turn and stats not lost
Free; player ‘rests’	Player heals X HP, health maintained, health does not exceed max health	Player heals X HP, health maintained, health does not exceed max health

Free; player moves on	Player does not rest or 'awaken', player goes to next turn	Player does not rest or 'awaken', player goes to next turn
Turn count exceeded; not all items in inventory	Game ends, player loses, all unobtained item pointers de-allocated, display out of time text	Game ends, player loses, all unobtained item pointers de-allocated
Player collects all three items, last item is either the brace from Search or a fragment from Battle	Game ends, display victory text, no memory loss, no issues with game ending in victory regardless of space it ended on	Game ends, display victory text, no memory loss, no issues with game ending in victory regardless of space it ended on

Reflection

This project was time-consuming, but did not feel as difficult as some in the past. While I formerly often had trouble with memory leaks, I did not have many at all here during writing and testing. I made it a point to get good ground work going early – have my linked spaces working at the start, slowly implementing and testing new functions and classes bit by bit, and always making sure the game worked at each step before proceeding to another. It makes me feel that I have learned a lot from my mistakes, and have a bit better grasp on the course materials.

One issue I had was that my initial plans and ideas were too grand of a scope – for the time I had and the minimum specs of the projects, things like stat growths and item usage did not seem necessary, and are more suited for a project on my own time. So I cut some of these ideas so I could focus on the core specs and aim to satisfy requirements, which kept my work a bit more concise and helped me to keep organized.

Another issues was how I wanted to implement items, because I was not familiar with using enums or using them in another class/function. I had to search how to use the syntax and how enums could be used, and used this in things such as player movement. I opted to not use enums for items other than to determine the name, which I could only do because I made the decision to simplify items to story/win condition items and just focus on getting an item container to work. My container of choice was an array, because initially I had intended to allow other items (i.e. combat items) into a player's inventory, and intended to be able to move items in/from the middle easily, so an array made sense to me since it is a limited capacity.

A third issue I had was with when the game ended due to death, or turn limit, and some items had not been collected from enemy loot or from the Search space. I was initially not properly deallocating memory in this case, so I improved destructors and function call logic to make sure this was done appropriately.

I am a huge fan of video games, playing many ranging from sports games, shooters, RPG/JRPGs, action games, etc. I am particularly fond of JRPG/strategy-RPG types, and my project is inspired by some game mechanics such as Fire Emblem, Final Fantasy, and Slay the Spire – the theme is heavily inspired by Fire Emblem: Awakening, and the stat increase mechanic is named after that game (though mostly because "train" didn't make sense in the context and "meditate/focus" didn't sound right, so it's really a filler name). I enjoyed the project thoroughly, even if I am not particularly happy with my final code upon submission.

There is much room for improvement in terms of readability/adaptability, and some things were hard-coded where others were coded with the intent to be re-used/adjusted easily. My daily life schedule made managing time tough, but I managed to get a good start on the project after many lessons learned in this course about time management. This project has me motivated to improve/rebuild the code and create a better text-based game that can serve as my own “passion” project.