Name: Artem Slivka
Course: CS162
Date: 8/15/17

Final Project Design and Reflection

Initial design

General program requirements: design a game with pointer-linked rooms using OOP concepts

Program's goal: The player will obtain 2 different items after killing creatures in 5 floors. On the 6th floor, he will place these items on the altar and will obtain the Holy Grail, a source of immortality. He will lose the game if:

- he loses strength after fighting with the creatures in the floor
- doesn't obtain the 2 items(max 2 only) necessary to enter the 6th floor
- doesn't fight the monster in each floor.

Program architecture:

- Class **Space**: this base class holds general information any room in the game. It's an abstract class.
- Class **BatCave**: this derived class holds batcave specific info only. This is where player will obtain the vampire's blood and fight vampire.
- Class **Swamp**: this derived class holds swamp floor specific info only. This is where the player will fight the Medusa creature and get one of her scales.
- Class **Altar:** this derived class holds altar information only. This is the room where the user obtains the Holy Grail after placing each of the items he obtained in room 1-5.
- Class **Dungeon**: this is a class that stores a pre-set linked list of rooms used in **DungeonRaid** class.
- Class **Player:** this base class holds player specific information only. It contains player's stats, inventory, etc.
- Class **Inventory:** this class maintains info on player's inventory.
- Class **DungeonRaid**: this class supervises the game so player can progress between 6 floors in the dungeon.

class **Player: public Creature**
PROTECTED • Item* inventory • int steps
PUBLIC • Player() • ~Player() • getSteps() • void incrementSteps(); • bool addItem(Item* newItem) • int getInventorySize() • void displayInventory() • `virtual int attack();` • `virtual void defend(int opponentAttack);`

class **Space**
PROTECTED • string roomName • Space* north • Space* south • Space* east • Space* west • Menu* menu1 • bool itemAcquired • Item* roomItems • `int playersTurn;` • `int winner = 0;` • `string playerNames[2];` • Creature* fantasyCreature • Creature* player
PUBLIC • Space(int name) • ~Space() • string getName() • Space* getNextRoom() • Item* getItem() • void setItem(string itemDesc, int quantity) • virtual void setMenu()=0 • virtual void printMenu()=0 • virtual void exit() =0 • `void createCreature(int player, int choice)` • `void start()`

• `bool playRound()` • `void play()` • `void flipPlayerTurn()`

class **BatCave: public Space**	class **SwampRoom: public Space**
PUBLIC • Batcave() • ~Batcave () • virtual void setMenu() • virtual void printMenu()	**PUBLIC** • SwampRoom() • ~SwampRoom() • virtual void setMenu() • virtual void printMenu()

class **Altar: public Space**
PUBLIC • Altar() • ~SwampRoom() • virtual void setMenu() • virtual void printMenu()

class **Dungeon**	class **DungeonRaid**
PROTECTED •	**PROTECTED** • Dungeon* rooms • Space* currentRoom • Creature* humanPlayer
PUBLIC • `Space* front;` • `Space* back;` • `Dungeon();` • `~Dungeon();` • `void addBack()` • `int getFront()` • `//void removeFront()` • `void displayQueue(string direction)` • `bool isEmpty()` • `displayNode(Space* input, string name);`	**PUBLIC** • DungeonRaid() • ~DungeonRaid () • void introduction() • void start() • void play() • bool exitGame() • void movePlayer(char) • void gameEnd() • void displayInventory()

Menu options

1. Explore dungeon's first floor
2. Display game shortcuts
3. Display player's inventory
4. Display player's current state
5. Quit game

Menu after finishing fight
Pick up item: y/n?

Menu after finishing first floor

1. Go to next room
2. Display player's inventory
3. Display player's current stats
4. Quit game

Game flow
Play game
Go to floor 1
Attack monster
If killed, lose game
If not killed, pick up item? y or no

Go to floor 2-5
Attack monster
If killed, lose game
If not killed, pick up item? y or no

Go to floor 6
Release items
If items not there, lose game
If items there, exchange items with holy grail, win game

Test plan and results

Test description	Input value	Expected result	Pass?
Floor menu			
go to next floor	1	explorer moves to next floor	yes
display explorer's inventory	2	displays inventory	yes
display explorer's stats	3	displays stats	yes
change explorer's stats	3, y	change stats from 2nd floor - 5th floor	yes
quit game	4	will go to external menu	yes
Floor 1, 3, 5			
Vampire activates Charm?		50 % chance of charm activation	yes
Vampire drops vial?		Vampire drops vial	yes
Item can be picked up?	y/n	Item picked up/not picked up	yes
Floor 2,4			
Medusa activates glare?		1/6 chance of glare activation	yes
Medusa drops scale?		Medusa drops scale	yes
Item can be picked up?	y/n	Item picked up/not picked up	yes
Testing losing conditions			
Pick up no items		Explorer dies - no items collected	yes
Pick up 1 item		Explorer dies - not enough items collected	yes
Fight 4/5 monsters		Explorer didn't kill all of the monsters	yes
Explorer losing strength		Explorer dies after losing strength(added feature to change strength for easy debugging)	yes
Inventory			
pick up multiple items	y/n	Gives duplicate item error - shows inventory limit	yes
pickup Holy Grail	g	Grail picked up correctly	yes
Memory leak			
Playing game on 1 first try		no leaks or segmentation faults	yes
Playing another game		no leaks or segmentation faults	yes

Final reflection

After looking over the requirements of this project, I felt it was a little too open-ended. At first, I decided to do a spaceship game, killing aliens at the end by venting them through an access hatch after luring them with bait acquired through the rooms. Although an interesting theme, the implementation didn't go smoothly so I downgraded to the current dungeon game. Due to time constraints, I didn't make a game with a map, just text-based only.

After deciding on the dungeon theme, I didn't have too much trouble coding the lower level classes in the game. For **Player** class, I used the creature class from Project3 and added extra members for tracking inventory, stats, and monsters killed. This was needed to check that the player completed the walkthrough the dungeon correctly. If he didn't collect all of the required items or didn't kill all of the monsters, he would die upon stepping on floor 6.

For **Space** class and its derived classes, I copied the code from Project3's game class for gameplay and modified it slightly to suit the new project's requirements. I had to add the drop feature so the player could collect items in each room. To link the rooms together, I just tweaked the linked list from Lab6 and unlinked the front and back pointers so the player couldn't go in circles from floor 1-6 and back to 6 from 1.

The game class for the project**, DungeonRaid** was more difficult. Since during design stage, I planned this class the least, that's where I spent the majority of my time coding. The rest of the classes were just slightly modified but this one needed a lot of work. I had issues on deciding the conditions where the player will die so it took a while to test and make sure the player won/lost under the right conditions. While implementing the display stats feature, I added the change strength feature to allow easier debugging and not rely on chance for Medusa to kill the player.

Finally, one major issue I had in this project was the memory leaks. On the surface, the rule for memory allocation is simple, delete after using new operator. In my case though, I was passing an item object (class member pointer to item object) from the majority of the rooms to the player. The player was then adding it to the inventory vector. The problem came about when trying to free this item from memory. Both the space object and the player object would try the free the same memory for the item object. As a result I had nullpointer and other errors during runtime for a while. After around 5-8 hours, I got that resolved, no more memory leaks.

I had fun doing this project but I would've liked more time to implement a map version of the game. An additional week would've been better to accomplish this.