Author: Daniel Koning

Date: 11/28/2017

Description: Design Description, Test Plan, and Reflection for a text based game (CS162 - final project) in which the player moves through various rooms in a cave completing challenges and collecting items in an effort to ultimately escape.

DESIGN DESCRIPTION

Main()

1) Seed random functions with srand()
2) Create a Game object
3) Display game instructions
4) Call runGame function

Game (class)

Run Game function:

1) Call setRooms() to build game play area
2) Set current room to room 1
3) Display starting location and get first move
4) Player takes 10 move damage for first move
5) Start loop until player wins or runs out of energy
6) Call current room's action() function and store return in varDamage
7) If varDamage is 0, call room's getItem() function and add any new items to the backpack with the addBack function or enable blessings with player set functions
8) If varDamage is -1, call player rest() function (or print error if rest count = 3)
9) If varDamage is -2, call backpack search function for 2 keys and either set player win to 1 or tell player to find more keys
10) If varDamage is not 0, -1, or -2, call takeDamage function for the player and set current room to last room (returned to previous room)
11) Call display status to print current game status
12) Call menu function to get player's next move
13) Call Player's take damage function to take move damage

Set Rooms function:

1) Dynamically allocate memory for room1 (class StartRoom)
2) Dynamically allocate memory for room2 (class TrollRoom, QuizRoom1, or QuizRoom2)
3) Dynamically allocate memory for room3 (class TrollRoom, QuizRoom1, or QuizRoom2)
4) Dynamically allocate memory for room4 (class TrollRoom, QuizRoom1, or QuizRoom2)
5) Dynamically allocate memory for room5 (class TrollRoom, QuizRoom1, or QuizRoom2)
6) Dynamically allocate memory for room6 (class EndRoom)
7) Set Top, Bottom, Left, and Right pointers for each room to build the game play area

Display Status function:

1) Take the current room in and print map based on player location
2) Call Get functions for player Energy, player Rests, and backpack Contents

Game destructor:

1) Delete dynamically allocated memory for room 1-6

Player (class)

```
Player constructor:
    1) Set energy to 100 and rests, dampen, and revive to 0
Get Energy function:
    1) return energy
Get Rests function:
    1) return rest count
Get Dampen function:
    1) return dampen status
Get Revive function:
    1) return revive status
Set Dampen function:
    1) enable dampen
Set Revive function:
    1) enable revive
Take Damage function:
    1) get damage input as parameter
    2) if dampen is active, cut damage in half
    3) subtract damage from energy
    4) if energy reaches 0, revive if possible, otherwise display end of game message
Rest function:
    1) return energy to 100
    2) increase rest count by 1
```

Space (class)

```
Action function:
    1) pure virtual
Get Item function:
    1) pure virtual
Menu function:
    1) Loop through each of the current room's (passed in) pointers and add menu option for each
       that isn't NULL
    2) Get user menu choice + validate input
    3) Return next room choice based on menu option selected
Set top function:
    1) top = topIn
Set bottom function:
    1) bottom = botIn
Set right function:
    1) right = rightIn
Set left function:
    1) left = leftIn
Set name function:
    1) name = nameIn
Get top function:
    1) return top pointer
Get bottom function:
    1) return bottom pointer
Get right function:
        1) return right pointer
Get left function:
        1) return left pointer
Get name function:
        1) return room name
```

StartRoom (class)

```
Action function:
        1) Ask user if they want to rest or continue
        2) Get response + validate
        3) Return -1 to rest or 0 to do nothing
Get Item function:
        1) return "0", no item
```

TrollRoom (class)

```
Action function:
        1) Check if room has already been completed (if yes, tell user and return)
        2) Display player / troll attack rolls
        3) If player roll >= troll roll, player wins, return 0 damage
        4) If player roll < troll roll, troll wins, return 40 damage
Get Item function:
        1) Check if room has already been completed (if yes, return "0", no item)
        2) If not completed, mark as complete and return "Key"
```

QuizRoom1 (class)

```
Action function:
        1) Check if room has already been completed (if yes, tell user and return)
        2) Display gnome question 1
        3) Get player heads or tails guess + validate
        4) Get random num 0-1 (0 = tails, 1 = heads)
        5) If player guess = num, player wins, return 0 damage
        6) Otherwise, return 20 damage
Get Item function:
        1) Check if room has already been completed (if yes, return "0", no item)
        2) If not completed, mark as complete and return "Revive"
```

QuizRoom2 (class)

```
Action function:
        1) Check if room has already been completed (if yes, tell user and return)
        2) Display gnome question 2
        3) Get player answer + validate
        4) If player guess = 3, player wins, return 0 damage
        5) Otherwise, return 20 damage
Get Item function:
        1) Check if room has already been completed (if yes, return "0", no item)
        2) If not completed, mark as complete and return "Dampen"
```

EndRoom (class)

```
Action function:
        1) Ask user if they want to try to open the lock or continue
        2) Get response + validate
        3) Return -2 to try to open the lock or 0 to do nothing
Get Item function:
        1) return "0", no item
```

Backpack (class)

```
Backpack constructor:
        1) Set head pointer to nullptr
Is Empty function:
        1) Check's if the head pointer is set to nullptr and returns true or false
Add Back function:
        1) Takes an item string in and creates a new node in the queue for it
        2) Update appropriate prev and next pointers
Get Front function:
        1) Return the item in the head node
Remove Front function:
        1) Deletes the current head node
        2) Set head to nullptr if this was the last node
Print items function:
        1) Loop through the queue from head to tail and print item names
Item Search function:
        1) Loop through the queue from head to tail and count occurrences of passed in string
        2) Return the number of items found
Backpack destructor:
        1) Loop through the queue from head to tail and delete all nodes
        2) Set head to nullptr
```

TEST PLAN

CONDITION	EXPECTED OUT	ACTUAL OUT
User enters something other than the move menu option numbers	Program should tell user that this is invalid and prompt them again	As Expected
User enters a valid move menu option number	Player location should update to the correct, user selected room, player energy should decrease by 10 (5 w/ dampen)	As Expected
User enters something other than "1" or "2" at the startroom menu	Program should tell user that this is invalid and prompt them again	As Expected
User enters "1" at the startroom menu	Player should rest and have energy restored to 100 (or error if rest count = 3)	As Expected
User enters "2" at the startroom menu	Player should be prompted for their next move	As Expected

User enters something other than "1" or "2" at the endroom menu	Program should tell user that this is invalid and prompt them again	As Expected
User enters "1" at the endroom menu	Program should check to see if player has 2 keys and either open the final door (2 keys), or tell player how many are still needed	As Expected
User enters "2" at the endroom menu	Player should be prompted for their next move	As Expected
User enters a TrollRoom	Program should have the user fight a troll	As Expected
User enters a QuizRoom	Program should give the user a gnome challenge (Heads / Tails or Quiz)	As Expected
Player energy reaches 0 with no revive	Game ends, player loses	As Expected
Player energy reaches 0 with revive active	Player regains 100 energy, revives no longer available	As Expected
Player loses to troll	Player takes 40 damage (20 w/ dampen) and is returned to the last room	As Expected
Player beats troll	Player takes 0 damage and gets a Key	As Expected
Player loses to gnome	Player takes 20 damage (10 w/ dampen) and is returned to the last room	As Expected
Player beats gnome	Player takes 0 damage and gets either Dampen or Revive blessing	As Expected
Player energy hits 0 or player unlocks final door	Program exits	As Expected

REFLECTION

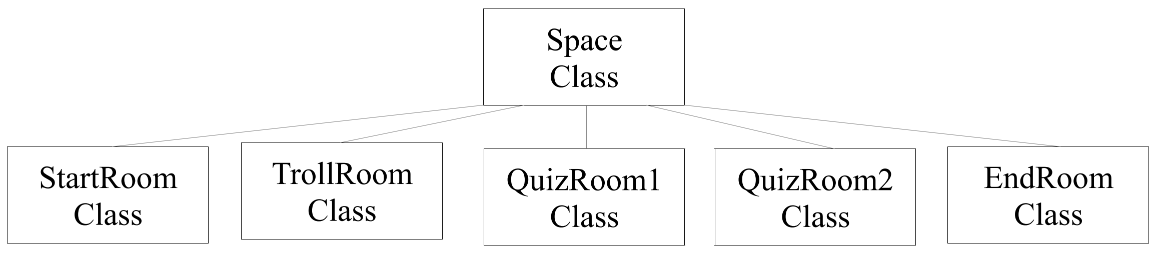
The final project had unique challenges that I didn't run into as much in other Labs and Projects throughout the course. Probably the greatest challenge for me in the final project was in how to begin. The final project was incredibly open ended and it really didn't have a defined path forward. This forced me to approach the final project in a fundamentally different way. The other major difficulty I ran into was in trying to create classes that were independent of one another. I initially tried to create a backpack as part of the player creation, but quickly realized that this made it difficult when trying to access backpack function from the main, rungame function without using a player function for access. I found other parts of the project far easier however in that there weren't really any new concepts to learn and, once the base design was locked down, the high degree of freedom afforded meant I could solve problems I ran into in ways that were most intuitive to me.

The first problem, the open ended nature of the project, was largely solved be spending extra time planning out the project ahead of time. While important in all of the labs / projects in this course, there was at least some basic structure to work with for the other coursework. The way I found my way through this difficulty was by brainstorming a few basic project ideas before doing anything else. I then went through each one and evaluated them against the core requirements for the project and the degree of difficulty that I thought would be required in building them. I spent a good amount of time at this stage also brainstorming features that I thought would be fun to add and started thinking about how they may be able to be worked into the overall design. Once I settled on an idea and put the core gameplay ideas down on paper, much of the detail work with the project came much more easily.

The second major problem I ran into was in trying to keep each part of my code independent of each other part. It seems to me that functions and classes are more useful when they serve a specific purpose that isn't reliant on too many other classes or functions outside of themselves. My first instinct with implementing the item system was to associate an item list with the player character. This meant that the player class needed to act as an intermediary whenever my game functions wanted to access items. I had a hard time getting this to work and the complexity that was added seemed to make the approach impractical. I eventually got frustrated and attempted a different way of solving the problem. Instead of explicitly tying the item list to the player by putting it within the player class, I simply created an object for the player's backpack in the game program. This way, the player class and the backpack class could be accessed directly from my game class without unnecessary middleman functions. It also came to mind that, if I wasn't the only one working on the project, it would be very difficult for someone else to follow the complex way in which the classes related to each other in my original implementation.

Ultimately, I found this project to be more enjoyable than any of the others in the course. I attribute this mostly to the amount of freedom allowed with setting the direction and content for the project which allowed for some more creativity and a real sense of ownership. I found that I added a few features above and beyond the base project requirements just because I wanted to make a game that would actually be somewhat enjoyable. I was able to add a personal touch to some of the challenges in the rooms in a way that I found fun. There is definitely a time and place for structured learning but, sometimes, letting the creativity flow a little bit helps keep things interesting.

SPACE CLASS HEIRARCHY


Space Class
StartRoom Class
TrollRoom Class
QuizRoom1 Class
QuizRoom2 Class
EndRoom Class