Class: CS 162
TA: Niza Volair
Leigh Ann Pruitt
Assignment 3: Design Document

Understanding

The purpose of this program was to implement various object oriented programming techniques including inheritance, polymorphism, and pointers. These are utilized through execution of a linked list, with each node including four pointers. The nodes are to be set up to allow a player to move through the rooms (via the pointers) and to interact with each room, or space. The spaces will be designed such that they are inherited from an abstract base class, which allows exhibition of polymorphism. This base class is to include a special pure virtual function (which will cause the base class to be abstract) to allow for differing actions within each space. Polymorphism will allow this special function to be called through the pointer to the base class with each instantiation of a derived class. At least three derived classes from the abstract base class Space are required. These derived classes will be the rooms through which the player moves.

Assembling the Spaces via a linked list is to compile a game with which a player interacts. This game must have a goal, or a win condition. The game must evaluate for this win condition and correctly exit (or continue, as desired by player) correctly. Moreover, the game must be able to track the player's current location. Since a linked list is to be utilized, the game must know in which node the player currently exists. This will allow the game to execute the appropriate functionalities based upon the player's location.

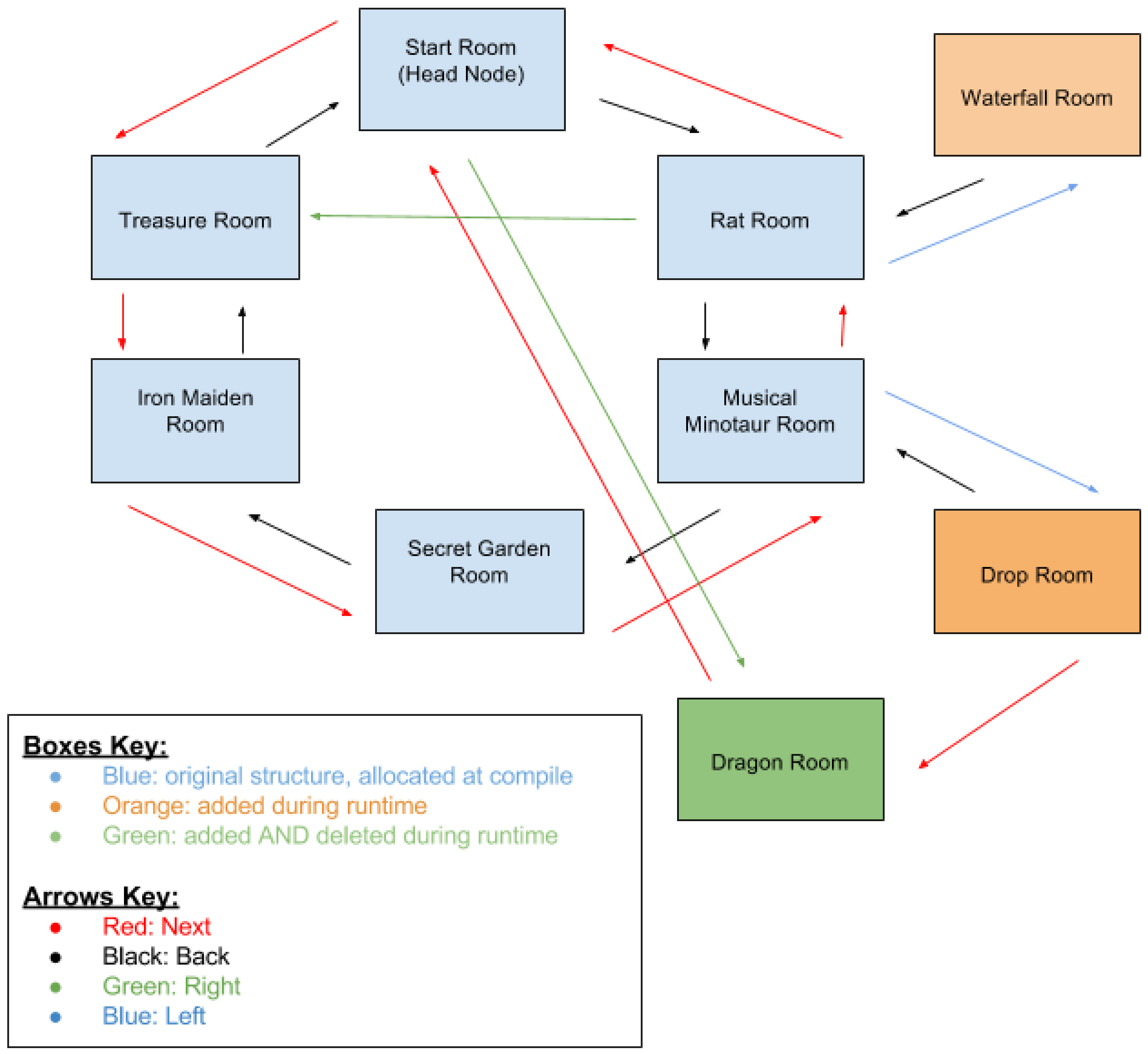
The structure of the spaces is also to change. At least one space must be deleted and at least one space must be added. Both of these actions should occur during runtime and not during compile time or from execution of a destructor. This functionality will change the structure of the linked list during runtime. After execution of an addition, the structure of the list will not match the structure set up at compile time. After deletion of a node, the program will not be able to return to that node. The node may be re-created through the initial logic of its creation if the game allows, but return to the space allocated initially will not be allowed.

As mentioned above, the player is to interact with each Space and not simply pass through it. Therefore, each space should have functionality which indicates to the player the current state of the game and allows the player to choose the actions desired. These actions should then lead to differing game states, and should allow for a player loss if incorrect decisions are made. Similarly, a time limit should be imposed upon the game. This may be implemented through use of a clock, a "countdown," or some other tracking method in which the player's time in the game is limited. That is, the player should not be able to move about in the structure limitlessly.

Design

The theme for my game will be “Doughnut Dungeon.” The game will be a tongue-in-cheek take on the myriad of dungeon crawler role playing games that have populated the gaming market since the inception of Dungeons and Dragons. Each room will display a special menu (via the special function as per the instructions) that allows the player to choose the action desired for that room. Each action will either display a message or cause some change in game state (stowed item in knapsack, change in player health, change in location, player death, etc.). The structure of the dungeon is displayed below.

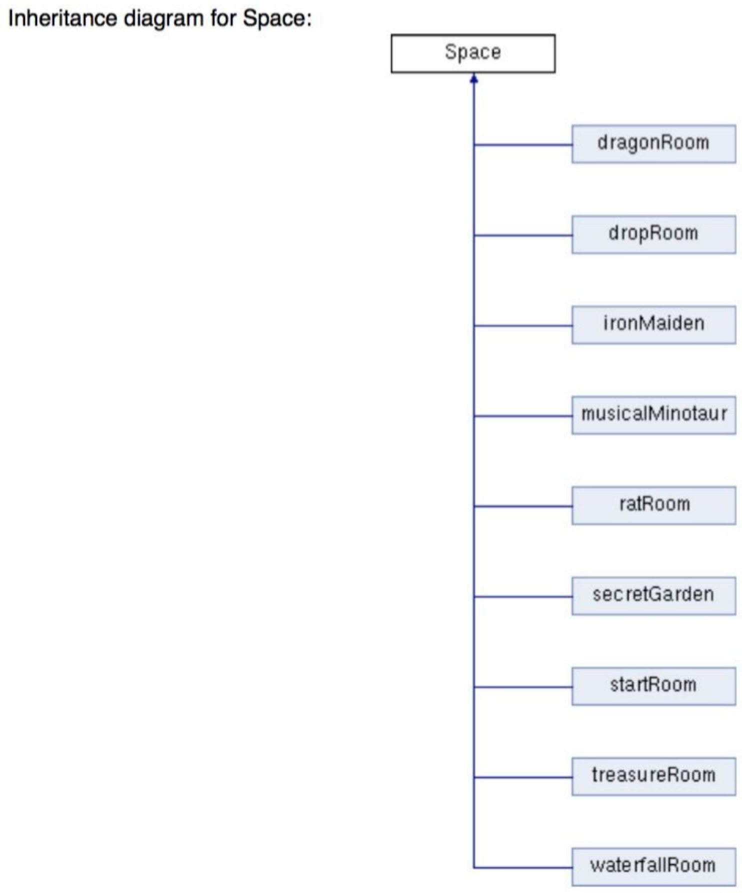
Doughnut Dungeon



spaceList - Linked List

As shown above, the initial layout of my game will be a doubly linked circular list (blue boxes). These will be implemented and designated when the game is compiled by the spaceList constructor. At this point, the player will be able to move between the blue rooms in either direction. For regular movement, the player can only move into an adjacent room using the next/back pointers. However, the rat room offers a portal that will pull the player across the structure into the treasure room. Implementation of this functionality is discussed below in the room-specific section. When certain requirements are met within the Musical Minotaur Room, the Drop Room will be allocated and constructed. This room is not deleted during runtime. The player may move back and forth from this room, as a certain item is required to activate the "drop." This prevents this room from becoming a "trap" in which a player could become stuck if the required item had not yet been obtained. By allowing the player to move back into the original ring structure, the player can search for the item required to activate the drop. When the drop is activated, the Dragon Room is allocated and the player will enter that room. After completing the Dragon Room (whether through defeating the dragon or being defeated by the dragon), the only way out of the Dragon Room is via the portal (the weird orb) that sends the player back to the head node (Start Room). The Dragon Room is then freed and the player can continue to play or exit the dungeon (if key was obtained from dragon). Below, the description of the functionalities of specific rooms is discussed.

As specified in the instructions, the rooms are implemented with polymorphism and inheritance. An abstract base class named Space is utilized and each room is inherited from it. Virtual functions displayMenu(Player&), play(Player&), and displayWelcome(Player&) are defined as pure virtual functions, enforcing the base class as abstract. The inheritance diagram for the rooms is displayed below (generated by Doxygen).



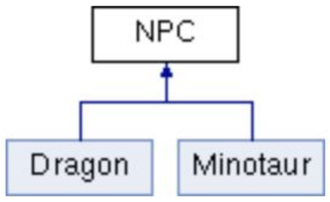
The spaceList class also contains a moveRooms function that controls player movement throughout the structure. This function utilizes a while loop to “run” the game while the player has not died and has not selected to quit (set as 0). Within the function, a variable called moveWay is used to indicate to the function the direction in which the player should be moved and/or the actions that should be executed. This is controlled by a switch statement. As per the assignment requirements, new rooms are created during gameplay. These rooms are indicated in both orange and green above (Waterfall Room, Drop Room, and Dragon Room). Also per the assignment specifications, one room is deleted during gameplay. This room is indicated above in green (Dragon Room).

Player

The player is implemented as a standalone class. Functions are implemented to allow for battle with NPCs, as well as personal functionality such as viewing knapsack, checking knapsack for food, etc. A player is instantiated in the main function and is then passed by reference to the functions that modify a player’s current status (for example, attacks by non-player characters (NPCs), healing functions within rooms, addition of items to knapsack, etc.). The player has a knapsack, which is an array of 25 Items. Items are struct containing values indicating the name and value of the item, as well as whether or not it can be eaten or has been deleted. As the player traverses the structure, Items can be accumulated and added to the knapsack array. As mentioned above, functionalities are provided for interacting with the knapsack.

NPC

Four types of NPCs are encountered in this program - zombie, rat, minotaur, and dragon. Inheritance is utilized among these as well. NPC is used as a base class, but is not abstract. Both zombie and rat are instantiated as objects of type NPC. This base class has a non-special attack function that simply returns a random number up to the object's attack value (atk). Both the health and the attack value (atk) of the object are initialized through a constructor that takes two integer parameters. The minotaur and dragon are inherited classes of the base class NPC. The minotaur has a special attack function in which the minotaur has a chance (1/atk) of damage being increased by 2 - the violin attack. Similarly, the dragon has a chance (1/atk) of damage being increased by 3 - the flametongue attack. The inheritance diagram for NPC is displayed below (generated with Doxygen).



Start Room

The start room is placed at the head node of the linked list. This room begins with an explanation of the player's circumstance and offers to look around the room. Since "looking around the room" is rather necessary, the program continues to prompt if the user chooses no (n), and will eventually roll to the description of the room that is gained by looking around the room. After the description has been shown, the player is offered a list of options. The results of each choice are:

1. Try to open the door - this will display one of two messages for the user.
    a. After the key has been obtained (end game), a message regarding exiting the dungeon is displayed. User is given option to leave dungeon. If n is entered, program will force exit.
    b. At the current state (begin game), a message will simply indicate that the door is locked. Display returns to menu.
2. Investigate the orb - displays a message that the orb is weird, and to leave it alone. Display returns to menu.
3. Inspect lantern - displays a message that the lantern is hot. Display returns to menu

4. Take left path - directs to rat room (see appropriate pointers in diagram)
5. Take right path - directs to treasure room (see appropriate pointers in diagram)
0. Quits game (program exits)

From the start room, either path can be taken for gameplay. No objects in this room are required for win state. If the player returns to this room, a different description will be shown that indicates player has been here before.

Rat Room

The rat room is placed at the tail of the list (rat tail?). Upon initial entry, player is attacked by a rat (instance of NPC). Battle ensues. If rat defeats player, game is terminated. If player defeats rat, a description of the room is displayed along with a menu of options. The results of each choice are:

1. Investigate hole in wall - this calls the ratHole function, which rolls for a random event:
    a. Portal (chance 1/6) - This action changes player's current location to the Treasure Room (described below).
    b. Revenge (chance 2/6) - This action displays a message regarding a rat bite and deals one point of damage to player. Display returns to menu.
    c. Cheese (chance 3/6) - This action indicates to the player that the rat's cheese has been retrieved. If user chooses to eat the cheese, 3 points of hunger will be restored. If user chooses not to eat the cheese, the option will be given to stow cheese in knapsack. If user chooses to stow cheese, Item cheese will be created and added to player's knapsack. Display returns to menu.
2. Return to starting room - directs to Start Room
3. Take left path - directs to Waterfall Room.
4. Take right path - directs to Musical Minotaur Room.
5. Examine rat corpse - displays silly message. Display returns to menu.
6. View items in knapsack - calls viewKnapsack function, which displays list of current items. Also allows user to search for items to eat (allows user control over hunger). Display returns to menu.
7. View current player stats - calls playerStats which displays current player health, hunger, armor, and attack points. Display returns to menu.
0. Quits game (program exits)

From the Rat Room, either path can be taken for gameplay. If the left path is chosen, player must return to Rat Room to continue in path. No objects in this room are required for win state. If the player returns to this room, a different description will be shown that indicates player has been here before.

Waterfall Room

The waterfall room is placed at the left pointer from the tail of the circular linked list. Upon initial entry, a description of the room and menu are displayed. The results of each choice are:

1. Investigate waterfall - sets players health to full (20). Displays message regarding healing. Display returns to message.
2. Jump into river - Displays message that player has drowned. Deals 999 damage to player, which results in player death. Game exits.
3. Return to rat room - directs to Rat Room.
4. View items in knapsack - same action as view knapsack in previous room. Display returns to menu.
5. View current player stats - same action as view stats in previous room. Display returns to menu.
0. Quits game (program exits)

From the Waterfall Room, player must return to Rat Room to continue moving about the structure. No objects in this room are required for win state. If the player returns to this room, a different description will be shown that indicates player has been here before.

Treasure Room

The treasure room is placed at the next pointer of the head node. Upon initial entry, a description of the room is displayed, followed by a menu. The results of each choice are:

1. Open the chest - this calls the treasureChest function, which has two possible actions:
   a. If the chest has not yet been opened, a congratulatory message will be displayed regarding receiving a sword, and the player's attack points are doubled.
   b. If the chest has been opened, a message regarding such will be displayed, and the sword will not spawn again. Display returns to menu.
2. Inspect hole in the wall - displays a message indicating the hole smells like rats (clue to portal from Rat Room). Display returns to menu.
3. Return to starting room - directs to Start Room.
4. Open door and proceed down hallway - directs to Iron Maiden Room.
5. View items in knapsack - same action as view knapsack in previous room. Display returns to menu.
6. View current player stats - same action as view stats in previous room. Display returns to menu.
0. Quits game (program exits)

From the Treasure Room, player may proceed to either Start Room or Iron Maiden room. Either choice is acceptable for gameplay. No objects in this room are required for win state. However, acquiring the sword increases likelihood of defeating dragon and minotaur in later rooms. If the player returns to this room, a different description will be shown that indicates player has been here before.

Iron Maiden Room

The Iron Maiden room follows the Treasure Room. Upon initial entry, a description of the room is displayed, followed by a menu. The results of each choice are:

1. Return to treasure room - directs to Treasure Room.
2. I'm feeling lucky! Roll a random number between 1 and 4 to choose which iron maiden to inspect - calls maidenLuck function, which rolls randomly for four equally possible outcomes:
    a. Torture - message is displayed that you are pulled into one of the devices. One point of damage is dealt, indicated by message. Display returns to menu.
    b. Spider - silly message regarding spider is displayed. Display returns to menu
    c. Zombie - has two actions:
        i. if player has not yet battled the zombie (instance of NPC), message is displayed regarding initiation of battle with zombie. Battle function is called. If zombie defeats player, game exits. If player defeats zombie, a chest plate is dropped by zombie that increases player's armor by 1. Player is given option to pick up chest plate. If y selected, armor is incremented.
        ii. If player has previously defeated zombie, menu simply displays message regarding such. Display returns to menu.
    d. Passage - displays message regarding secret passage. Directs to Secret Garden Room. This is the only way to proceed to Secret Garden Room from this room
3. View items in knapsack - same action as view knapsack in previous room. Display returns to menu.
4. View current player stats - same action as view stats in previous room. Display returns to menu.

0. Quits game (program exits)

From the Iron Maiden Room, player may proceed to either Treasure Room or Secret Garden room. Either choice is acceptable for gameplay. No objects in this room are required for win state. However, acquiring the chest plate increases likelihood of defeating dragon and minotaur in later rooms. If the player returns to this room, a different description will be shown that indicates player has been here before.

Secret Garden Room

The Secret Garden room follows the Iron Maiden Room. Upon initial entry, a description of the room is displayed, followed by a menu. The results of each choice are:

1. Inspect blue flower - calls blueFlower function that describes flower and allows player option of eating, stowing in knapsack, or ignoring. If user eats, hunger is increased by 1 (same action occurs if eaten from knapsack). Otherwise, item is stowed in knapsack via addToKnapsack or ignored. Display returns to menu.
2. Inspect the thing that looks like lettuce - calls lettuceThing function, which is similar to blueFlower above. Option to eat, stow, or ignore. Display returns to menu.
3. Inspect left vine - Silly message is displayed, and secret doorway is opened. Directs to Musical Minotaur room.

4. Return to room with iron maidens - directs to Iron Maiden room.
5. Climb right vine - Displays message regarding falling out of window. Deals 999 damage and results in player death. Game exits.
6. View items in knapsack - same action as view knapsack in previous room. Display returns to menu.
7. View current player stats - same action as view stats in previous room. Display returns to menu.
0. Quits game (program exits)

From the Secret Garden Room, player may proceed to either Iron Maiden Room or Musical Minotaur room. Either choice is acceptable for gameplay. No objects in this room are required for win state. This room provides a re-spawning food source for management of hunger if extended gameplay is desired. If the player returns to this room, a different description will be shown that indicates player has been here before.

Musical Minotaur Room

This room may be accessed from either the Secret Garden room or the Rat Room. Passage through this room is required for access to the Dragon Room and completion of the game (since the dragon drops the key required for exit). Moreover, within this room, the minotaur must be defeated and his violin obtained. In the next room (Drop Room), the violin is required for passage to the Dragon Room (and completion of the game).Upon initial entry, a description of the room is displayed, followed by a menu. The results of each choice are:

1. Return to mutant rat room - Directs to Rat Room.
2. Return to secret garden room - Directs to Secret Garden Room
3. Proceed through the small doorway - Directs to Rat Room (provided as menu option for user who has not yet visited rat room)
4. Pluck the stone violin strings - displays message regarding transport, directs to Drop Room. This is provided as a secondary passageway to Drop Room if user wishes to defeat minotaur and then return to Waterfall Room to heal prior to facing final Dragon.
5. Pull the minotaur statue’s tail - two actions:
    a. If minotaur has not yet come to life this game, a message is displayed indicating the minotaur has been animated. Battle function is called and battle ensues. Two outcomes:
        i. If minotaur is defeated, a violin is dropped and player has option to pick up. Violin is required to proceed past Drop Room. Program will allow player to ignore violin. Portal is then opened and player is given the option to pass through. Program will not allow negative answer and will loop until user agrees. Directs to Drop Room
        ii. If player is defeated, game exits.
    b. If minotaur has come to life, message indicates such and display returns to menu.
6. View items in knapsack - same action as view knapsack in previous room. Display returns to menu.

7. View current player stats - same action as view stats in previous room. Display returns to menu.
0. Quits game (program exits)

Drop Room

This room can only be accessed via the Musical Minotaur room - by defeating the minotaur or by plucking the violin strings. This room immediately precedes the final room with the dungeon boss, the Dragon, and is the only means by which to access the Dragon Room. Upon entering the room, a message is displayed regarding the room's description, followed by a menu. The results of each choice are:

1. Play the violin - knapsack is checked for violin. Two outcomes:
    a. If player has obtained the minotaur's violin, the room will collapse. Directs to Dragon Room.
    b. If player has not obtained violin, message will display indicating need for violin. Display returns to menu
2. Return to room with minotaur statue - displays message regarding statue coming to life (intended as player hint). Directs to Minotaur Room.
3. View items in knapsack - same action as view knapsack in previous room. Display returns to menu.
4. View current player stats - same action as view stats in previous room. Display returns to menu.
0. Quits game (program exits)

As described above, the Drop Room must be accessed to complete the game, as it leads directly to the Dragon Room. No objects are obtained in this room.

Dragon Room

As with the previous two rooms, this room must be accessed for the win condition to be acquired. This room is activated after a player with the violin in the knapsack chooses option 1 in the Drop Room. Upon entry to the Dragon Room, a message is displayed regarding the boss fight and battle ensues. If the dragon defeats the player, the game exits.

If the player defeats the dragon, the player is restored to full health and given an option to open a chest that has appeared. If user selects to not enter the chest, program will loop until the user opens the chest. A message regarding a doughnut is displayed.

Next, a message regarding the dropped key is displayed. For exit from the dungeon, this key must be picked up. Program will allow the user to ignore the key. After defeating the dungeon and interacting with the chest and key, a descriptive message regarding the room is displayed, followed by a menu. The results of each choice are:

1. Investigate the orb - displays message regarding portal. Directs to Start Room.
2. Eat the doughnut - displays silly message. Display returns to menu.
3. Stow the doughnut in your knapsack - adds doughnut to knapsack (if doughnut is not stored in knapsack in Dragon Room, it will not be viewable in subsequent areas of the game)
4. View items in knapsack - same action as view knapsack in previous room. Display returns to menu.
5. View current player stats - same action as view stats in previous room. Display returns to menu.

After the player has defeated the dragon and obtained the key, player may then unlock door in Start Room.

Shortest route to win condition: Start Room > Rat Room > Musical Minotaur Room(must obtain violin) > Drop Room(must play violin) > Dragon Room(must pick up key) > Start Room. However, the nature of the objects encountered make this path unlikely to be successful as player will likely be killed by the Minotaur or the Dragon.

Testing Plan

Development

First, during development, I created the Player class and Item struct. In this way, I could be certain that any unexpected response in further development was due to my new additions and not my Player object. This object required various functionalities. Testing for them is displayed below. To test my functions I first instantiated a Player and then called each function for that Player. For testing of knapsack functions, an Item was instantiated Item bread(“bread”, true, 5);

Input	Expected Output	Actual Output	Case(s) Tested
Player instantiated, accessor called: getHealth()	20	20	Health set correctly by constructor, accessor returns correct value
setHealth(10); getHealth();	10	10	setHealth adjusts health correctly
playerStats()	Outputs message with current health, armor, attack,	Outputs message with current health, armor, attack,	Each value maintained correctly, function accesses

	hunger, and information regarding game end for health/hunger, message regarding hunger incrementing	hunger, and information regarding game end for health/hunger, message regarding hunger incrementing	and outputs correctly

setHealth(0);
isDead();
	Returns true	Returns true	isDead() checks correct variable, returns correct boolean value

attack(); (use cout to print value)	Value between 1 and atk	Value between 1 and atk	Number generated correctly, randomized, values within range
viewKnapsack();	Message that knapsack is empty	Message that knapsack is empty	Knapsack correctly recognizes empty

addToKnapsack(bread);
viewKnapsack();
	Item1: bread	Item1: bread	Recognizes item and outputs correctly

check forItem(“bread”);	Returns true	Returns true	Recognizes item and outputs correctly
checkForFood(); , select yes, select yes; playerStats	Indicates that item 1 is edible, no more edible items, knapsack is empty, hunger lessened by 3	Indicates that item 1 is edible, no more edible items, knapsack is empty, hunger lessened by 3	Recognizes edible item, adjusts hunger, deletes item, recognizes that item is deleted
allItemsDeleted();	Returns true	Returns true	Recognizes all items have been deleted;

takeDamage(5);
takeHunger(5);
multiplyAttack(2);
incrementArmor();
playerStats;
	Health = 5, hunger = 5, attack = 1-6, armor = 2	Health = 5, hunger = 5, attack = 1-6, armor = 2	Stats appropriately adjusted

During development I first created each of the space objects and interacted with them individually prior to implementation in a linked list. This allowed me to ensure that within-class functionality was correct prior to attempting to move about the structure. For testing purposes, I set the program to exit for each menu option that would later be used for movement.

Input	Expected Output	Actual Output	Case(s) Tested
Instantiation of each Space	See menu functions above in room-specific descriptions, game exits when move selected or at other appropriate time (death)	See menu functions above in room-specific descriptions, game exits when move selected or at other appropriate time (death)	Initial description displays correctly, menu displays, menu options function correctly (except move)
Set beenHereBefore to false	Different room entry message	Different room entry message	displayWelcome displays correct description based on game status
ratRoom - battle	Battle begins automatically on entry, menu displays after battle	Battle begins automatically on entry, menu displays after battle	Battle and menu functions called appropriately
treasureRoom - open chest	1st time - message re sword, attack points are doubled (use display stats to ensure) 2nd time - message says chest already opened	1st time - message re sword, attack points are doubled (use display stats to ensure) 2nd time - message says chest already opened	Chest is opened and adjusts attack points, cannot be opened twice
ironMaiden - maidenLuck	Randomly rolls one of four functions described above (repeat x10 to ensure that all outcomes occur)	All outcomes occur, player health appropriately adjusted during battle	Battle and menu functions called appropriately, battle adjusts player health
musicalMinotaur - pull minotaur’s tail	Battle occurs appropriately, violin is dropped and options output correctly, violin	Battle occurs appropriately, violin is dropped and options output correctly, violin is	Battle and menu functions called appropriately, battle adjusts player health,

	is instantiated and added to knapsack correctly, portal is indicated and loop continues until user enters y	instantiated and added to knapsack correctly, portal is indicated and loop continues until user enters y	player can add violin to knapsack
dropRoom - play violin (without violin) Play violin (with violin)	If no violin - message displays If violin - message displays and game exits	If no violin - message displays If violin - message displays and game exits	Presence of violin appropriately tested for
dragonRoom - battle	Battle begins automatically on entry, menu displays after battle	Battle begins automatically on entry, menu displays after battle	Battle and menu functions called appropriately
dragonRoom - after battle	User is forced to acquire doughnut (via chest), key drops, user has option to pick up. Key is added to knapsack if user selects.	User is forced to acquire doughnut (via chest), key drops, user has option to pick up. Key is added to knapsack if user selects.	Menu options, key instantiation and addition
startRoom - with key, select locked door	Key is recognized and user is given option to exit dungeon - force exit if selects no	Key is recognized and user is given option to exit dungeon - force exit if selects no	Key is recognized and appropriate menu option displayed

The linked list was then implemented as described above. For testing of the linked list, each menu option related to movement was selected and the resultant menu evaluated for correct move result. Valgrind's memcheck tool was utilized for evaluation of correct memory deletion.

Reflection

As described above, I used incremental development so that I could tell immediately with each new addition whether the addition functioned as expected. I implemented the Player class first and fully tested the class so that later implementation issues could be more accurately pinpointed within the program, rather than questioning Player functionality. Throughout development, I tested each addition thoroughly before proceeding. This made determination of error sources much easier. For example, my program did not have a memory leak prior to implementation of my Minotaur class. I found that after adding the Minotaur, I did have a leak, with one fewer frees than allocations. Initially, I believed that the program was not deleting my Minotaur correctly and attributed the error to some mis-implementation of inheritance or polymorphism. However, I found during further testing that the Minotaur did delete correctly if the player was defeated during the battle with the Minotaur. This led me to check the logic of my battle function, in which I found that the function was set to return if the player defeated the Minotaur, and the delete statements were never reached in this case.

An issue that I encountered during development was the interaction of the program with the Player. Previously, I had used functions to pass values back and forth between objects, with neither having knowledge of the other. However, due to the amount of Player interaction needed (damage, hunger, knapsack management, etc.), doing so would cause an immense amount of complexity and extraneous code. Because of this, I arrived at the solution of passing my Player object by reference to allow for interaction and adjustment of the Player as needed.

My next problem was the issue of spawning. I allow the Player to move about in the structure freely (as long as the player’s hunger counter does not reach ten). Most of my NPCs should spawn only once, since the player should not have to fight the rat every time the rat room is entered. To accomplish this I utilized boolean values to determine whether that object had previously spawned. If so, it did not spawn again. This was not implemented in the case of the Dragon, which should exist each time a Player enters the Dragon Room. Similarly, I tailored my welcome to the room messages to a boolean variable that checks whether the Player has been in that room before.

One of the main issues I had was moving the Player about in the structure. Initially I set up an extremely large and inefficient if/else statement so that I could continue on with testing while attempting to solve how to move the Player. A suggestion made on the class message board led me to set a variable that indicates to the program which direction the Player should be moved in, and to move the Player’s current location along that pointer. This allowed me to implement a much more concise switch statement.

Again, the importance of planning was reinforced. With each issue, I tended to stop and contemplate for some time, plan an approach, and test. This was both more efficient and more effective than methods I have used in the past, including the “try everything” approach. There were some functionalities that I did not have time to implement. Given more time, I would likely have added functionalities to allow the Player to search their knapsack and remove items as desired. Also, I would have implemented more pointers to differing locations. It took me some time to resolve issues with the pointer-based structure, and I was able to implement many of the movement functionalities I desired, but not all.