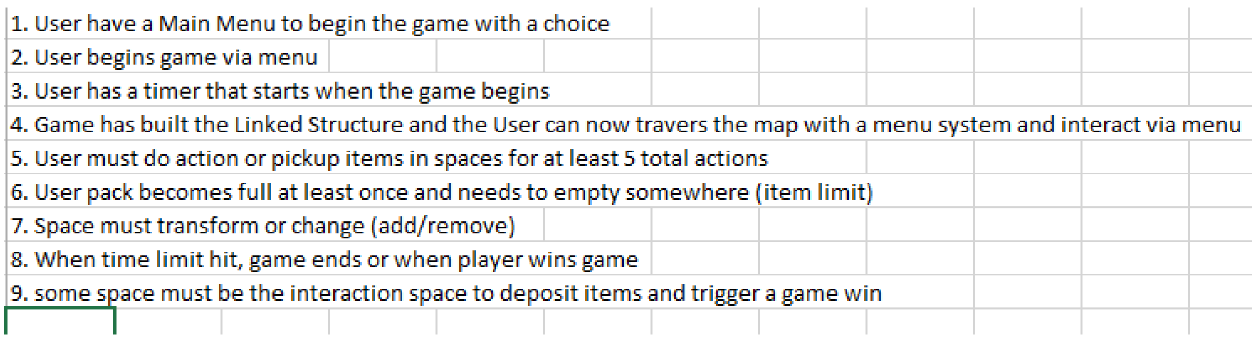
Jared Parkinson
Final Assignment
CS 162: T. Rooker

Final Reflections

To start off I just wanted to do a quick list of a few things you should know about the program:

- You have 7 minutes using time(NULL) as the timer rather than clock() because clock was giving me issues in Flip vs Windows
- The mapArray is only used in randomization and initialization of the Linked Structure. It is not meant to be used to traverse the Linked Structure.
- Add/Remove is only used once per requirements.
- There are **15 spaces** in my map, 3x5 in a rectangle x=3, y=5.
- Every single Object in this game is dynamically created except for the Menu in Main.cpp.

Requirements:
Here is a general walkthrough of how I understood the assignment to function that I wrote in my Design document:

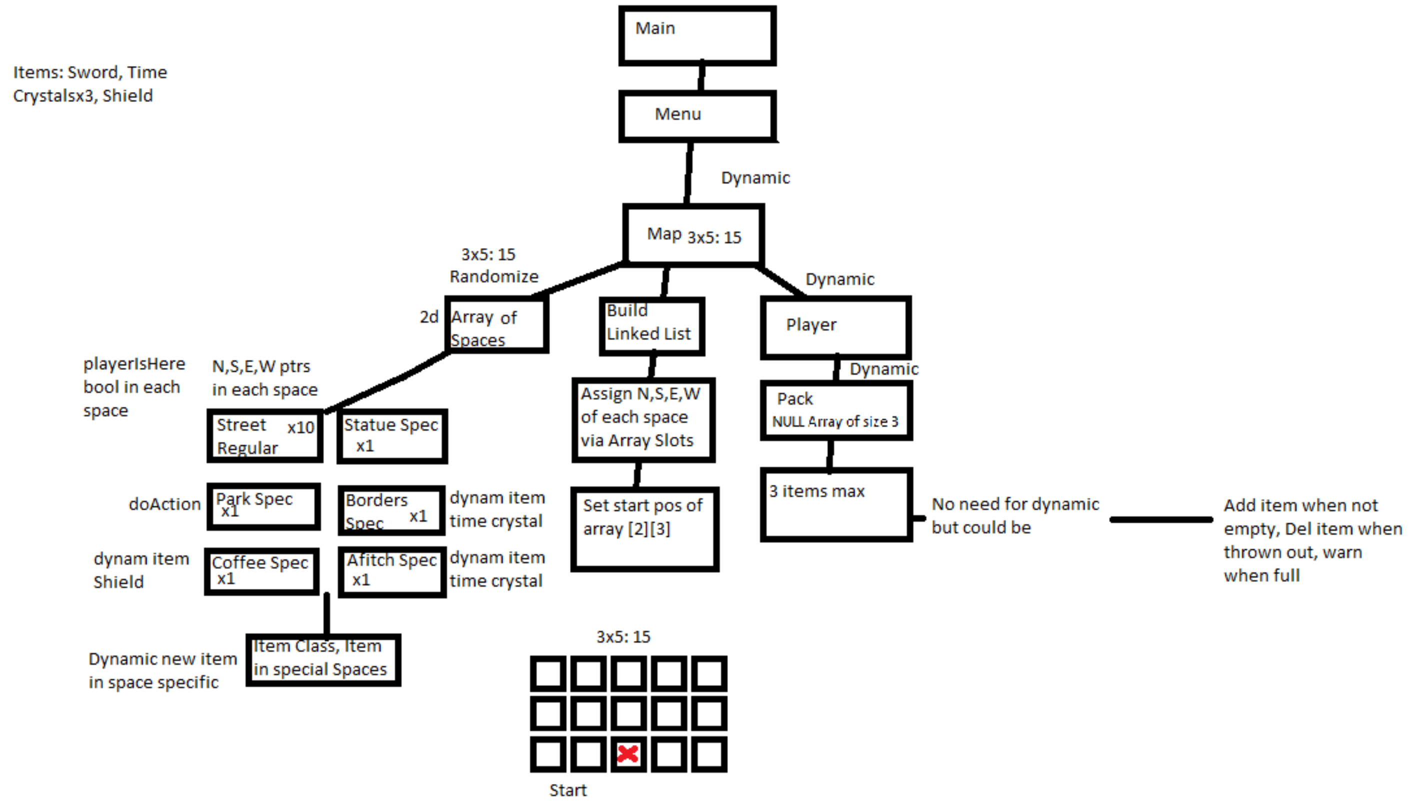


1. User have a Main Menu to begin the game with a choice
2. User begins game via menu
3. User has a timer that starts when the game begins
4. Game has built the Linked Structure and the User can now travers the map with a menu system and interact via menu
5. User must do action or pickup items in spaces for at least 5 total actions
6. User pack becomes full at least once and needs to empty somewhere (item limit)
7. Space must transform or change (add/remove)
8. When time limit hit, game ends or when player wins game
9. some space must be the interaction space to deposit items and trigger a game win

I'd like to overview for you a quick list of how I believe I fulfilled the requirements:

- There is a 7 minute timelimit (so you can read the paragraphs). I have tested and it quits the game when you try to make 1 more move after the time is up.
- The structure is 3x5. As a bonus, this structure randomizes each game. The Linked Structure is built from the random spaces and auto assigns N, S, E, W directions with a function.
- All links work perfectly and never fail in my tests (after fixes!)
- morphToFinalSpace() uses Add/Remove to remove the Current Park Space from wherever it resides on the map. It will add a Final Space in its place. This add/remove was not requested to work modularly.
- Prompt user to start the game is the first menu. Each command is followed and complains if you try to choose any other menu item.
- Actions: 3 gems and 1 sword are needed to win. The game will not progress without 2 pack trips to the Park due to a Pack Size of 3.
- Special() function works as intended with each Space. Even though Street space is generic, it still fires the Special function despite it doing nothing but print.

Before I go into classes, let me show you a general overview of how I designed the class hierarchy. See the below screenshot. Hopefully you can read it:



Items: Sword, Time Crystalsx3, Shield
Main
Menu
Dynamic
Map 3x5: 15
3x5: 15 Randomize
2d Array of Spaces
Build Linked List
Dynamic
Player
Dynamic
Pack NULL Array of size 3
3 items max
No need for dynamic but could be
Add item when not empty, Del item when thrown out, warn when full
Assign N,S,E,W of each space via Array Slots
Set start pos of array [2][3]
playerIsHere bool in each space
N,S,E,W ptrs in each space
Street x10 Regular
Statue Spec x1
doAction
Park Spec x1
Borders Spec x1
dynam item time crystal
dynam item Shield
Coffee Spec x1
Afitch Spec x1
dynam item time crystal
Dynamic new item in space specific
Item Class, Item in special Spaces
3x5: 15
Start

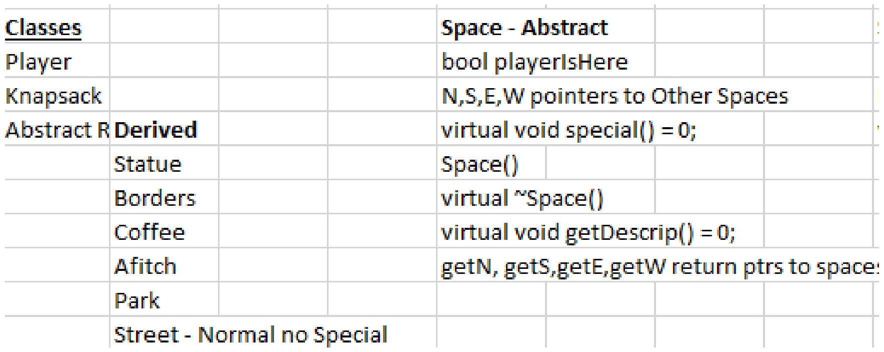
General Overview:

For the general overview of the class structure, let me explain how I envisioned it:

1. The Menu contains the Map, nothing under the map knows about the map
2. The Map contains the Player, Linked Structure, and all the Spacework
3. The Player contains the Pack, and the Pack contains the items. The player only knows of the pack and the pack only knows of the Items
4. The only link that the spaces have between each other are the N, S, E, W pointers. Spaces do not interact with each other.
5. At one point I almost let Park see the Map to get around an issue, but I figured out how I could use a bool to trigger the Final Space to replace the Park Space.
6. I designed it so the map was the same shape each time because I wanted the grader to have an easier time. Plus it was already a beast to program the randomness of it all. The Player always starts in the red X at 3, 3. The landing space under the player is different depending on the game.
7. I wanted every single thing to be Dynamic except the Main.cpp Menu item.
8. Shield was scrapped due to no need for it and lack of Spaces to utilize it as an item.

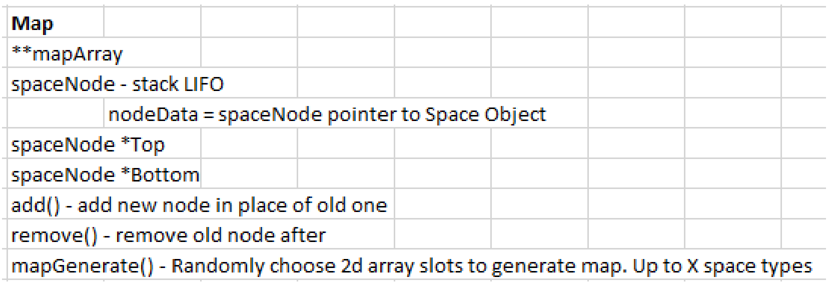
Class Design:

*Rather than a separate section for changes from initial design, I will address changes made to Classes in their sections below.



Classes		**Space - Abstract**
Player		bool playerIsHere
Knapsack		N,S,E,W pointers to Other Spaces
Abstract R	**Derived**	virtual void special() = 0;
	Statue	Space()
	Borders	virtual ~Space()
	Coffee	virtual void getDescrip() = 0;
	Afitch	getN, getS,getE,getW return ptrs to space
	Park	
	Street - Normal no Special	

MAP CLASS:
The Map Class is the world in which everything resides. When the Menu needs to work with something based on a choice, the Map class object is activated and called. Map was initially designed so that it would be using a spaceNode system. After working with it for a while, I felt that this Node system with nodes inside of the Map was a poor design. I then updated it to use a Linked Structure as purely Spaces Linked to other Spaces via N,S,E,W pointers (like the Traffic Light assignment of Linked States). Once again I was severely lacking in my foresight and found that Map needed significant changes.



Map	
**mapArray	
spaceNode - stack LIFO	
	nodeData = spaceNode pointer to Space Object
spaceNode *Top	
spaceNode *Bottom	
add() - add new node in place of old one	
remove() - remove old node after	
mapGenerate() - Randomly choose 2d array slots to generate map. Up to X space types	

Instead of just mapGenerate which would have been a hundred lines or more, I broke it out into 3 parts: Generate the Map Array and fill it with NULL, generate the random Space Creation Order and put that into the Array, generate the starting Linked Structure by connecting all the dots in the Array. This was the extent of the use of the Array and the Links are traversed by the player with pointers and not the Array Slots. The Link creation saves me from having to type 60 manual connections and checks the row on each loop then connects the dots with about 24 actual commands not including the if statements. It wasn't as compact as I had hoped, but easy to understand what it does. Data Members were added to keep track of the amount of spaces randomly generated and generate only the needed amount and exact number of special spaces.

Logically, Player is placed on the map so that he can be manipulated when the map requires Player actions. The Player is set in each space that he resides in and removed when he leaves. In addition, the Map knows what space is the Current Space so that it can manipulate that space when needed.

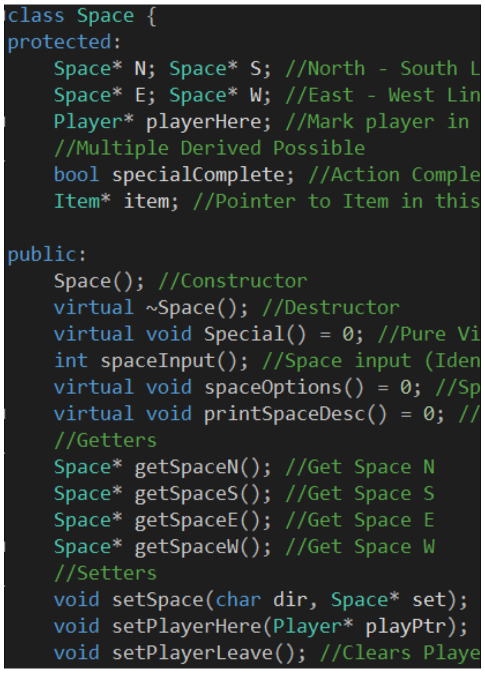
Finally, a needed change was to add the Space* finalSpace to the map so the map could manipulate it in end game. It was not a very attractive update, but was needed so that Park did not know about the Map when it fired off its special function. Rather than give a space that much power over the Map functions, I thought it better to have the map have power over that final space to complete the add/remove requirement.

SPACE CLASS:

The Space Class is pure abstract as requested. The Space class was designed to contain as many functions as it could to eliminate redundancy in the derived classes. For design decision, I intended Space to have Item* item and specialComplete even though some classes like Street would not use these. This is because it made sense from a code repeat standpoint when the majority of derived classes used those features and only one or two did not.

Space had many changes from the original design because my vision of it was cloudy until I started using and activating derived spaces. I realized that I needed more functionality from Space and added setSpace for the Link building. setPlayerHere / *playerHere was added to keep track of where the Player was after each move.

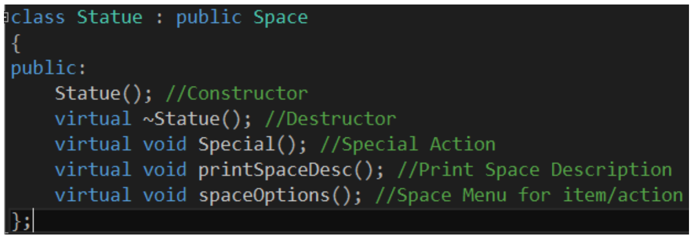
The spaceOptions and printSpaceDesc were added as to not clutter the Special function more than it already was. There is only one space where the Special is a bit longer than I would have liked. The Spaces were not supposed to know about the menu and therefore could not call menu functions. I decided to add a spaceOptions to show the options of the space. The choices for that list are in the Special functions. In addition, the project was large as it was, so I didn't want to add a Utility class unless the other classes got too cluttered. The Menu class is not all that bad and is under 230 lines which is smaller than other projects.


```
class Space {
protected:
    Space* N; Space* S; //North - South L
    Space* E; Space* W; //East - West Lin
    Player* playerHere; //Mark player in
    //Multiple Derived Possible
    bool specialComplete; //Action Comple
    Item* item; //Pointer to Item in this

public:
    Space(); //Constructor
    virtual ~Space(); //Destructor
    virtual void Special() = 0; //Pure Vi
    int spaceInput(); //Space input (Iden
    virtual void spaceOptions() = 0; //Sp
    virtual void printSpaceDesc() = 0; //
    //Getters
    Space* getSpaceN(); //Get Space N
    Space* getSpaceS(); //Get Space S
    Space* getSpaceE(); //Get Space E
    Space* getSpaceW(); //Get Space W
    //Setters
    void setSpace(char dir, Space* set);
    void setPlayerHere(Player* playPtr);
    void setPlayerLeave(); //Clears Playe
```

ALMOST IDENTICAL CLASSES: Afitch, Borders, Coffee, Statue

You will see that the Afitch, Borders, Coffee and Statue classes have identical HPPs. These classes are all similar and only differ in their use of Special with printing output and their Constructors that have different Items available to work with. As seen below, this is their general makeup.


```
class Statue : public Space
{
public:
    Statue(); //Constructor
    virtual ~Statue(); //Destructor
    virtual void Special(); //Special Action
    virtual void printSpaceDesc(); //Print Space Description
    virtual void spaceOptions(); //Space Menu for item/action
};
```
As I worked with these Spaces, I ended up having to make additional changes as time went on. I wanted the grader to know when they had completed an action. I have a bool for specialCompleted to allow for many checks in Spaces. One of the things it does is triggers a different message when you return to a space that you have completed. If you FAIL to pickup an item due to your pack being full, you will need to return to that same space later and it will still have the correct message and options.

UNIQUE CLASSES:
There were three unique classes that were created for special purposes of this program. Street, Park and Final.

Street:
Not much to the Street Space really. Street was created to be the generic equivalent to a Barbarian fighter in assignment 4. It is a filler space to make the map a bit larger and have nothing to accomplish in that space. The space interaction requirements are still met via the other spaces. Street did not get any updates since initial design other than to include the use of Special just for requirement purposes. I wasn't sure if every space actually had to use Special, so I made it use Special.

Park:
The park space is the key interaction space in the entire game. You must collect everything and place the items at this space. You do not place your sword here, but keep it in your pack. The Park space is also the trigger to Add/Remove a linked space from the structure. When you trigger the end game with this space after completing the objective, you will remove this Park space you are standing on in the Link and replace it with the Final Space right under your feet! As far as the player is concerned, he has no idea that it happened and assumes he was sent to another spot.
Some things about this space:

1. Park has unique Data Members for counting the Gems, bool for checking for your sword and String names to check the names of your items. Data Member strings / totalGems and haveSword were added to Park differing from original design.
2. Parks Special is unique in that it rummages through your pack and removes items that match its requirements. It will not remove your sword but it will tell you if you have it in your inventory or not. It will not allow you to continue if you do not have the sword.
3. Park Sphere at the monument will kick you out of the space after 1 option attempt and force you to choose a direction.

Final:
The final Space is the game ending space where you fight a final battle. The user has no other option in this space other than to fight and win. When the player is transferred here, he will begin the final game-ending scene.
Some things about this space:

1. I had issues working with this space and the add/remove when working with Park. It was due to the space being worked with on a garbage pointer.
2. This Space was not meant to do anything other than be a final ending scene and fulfil the add/remove requirement.

ITEM CLASS:
The Item Class does not do much other than hold a name. It is purely for creation using the constructor to name an item. The pack is the workhorse when manipulating items. I did not modify or change this class at all since I created it.

KNAPSACK CLASS:
The Knapsack Class was created as a container for the Items and dynamically creates an array to hold the Item Object Pointers. It was initially designed to do 3 main things.

1. Add an Item
2. Remove an Item
3. Check if the Pack is full

Updates:
This worked fine initially, but I needed more functionality of this class so I added more to it over my original design as I was working on the project:

```
void checkPackInv(); //Check Pack Inv (print)
bool searchPack(std::string passName); //Search Pack for item, no remove
```

I wanted the user to be able to see what was in the pack and print it to the screen to help them see what was in their inventory. I made checkPackInv() to print that inventory to the user screen.
In addition, I created searchPack(name) to find the specific objects needed to finish the game in the Park space. The space simply looks for the pre-defined names and removes them!

PLAYER CLASS:
The Player Class was designed so that it would be the owner of the Pack which was the owner of the Items. The player would interact with the pack and return any issues he had with the pack to the screen. I did not originally have the usePack function or the gameWinScene / getGameWinScene / setGameWin on the player. This was placed there as I mentioned earlier because it was not available (in the Map) for the Space to mess with inside of the Space Special. The usePack was much needed so that the Spaces could have the player reach in and add/remove/grab from their menu actions.

Testing:

Test Runs	**Action**	**Expected**	**Pass/Fail**
Menu System	Choose Option 1 in main menu	Start Game	Init: Pass, already working in other game
	Choose Option 2 to quit	Quit Game	Init: Pass, already working in other game
Movement in Map	Choose N, S, E, W directions	Attempt move in those directions	Init: Pass, Attempt move worked 1st try
	Actually move Player N,S,E,W	Move from one space to another seamlessly	Init: FAIL, the Player attempted to move to a garbage space that was not properly init

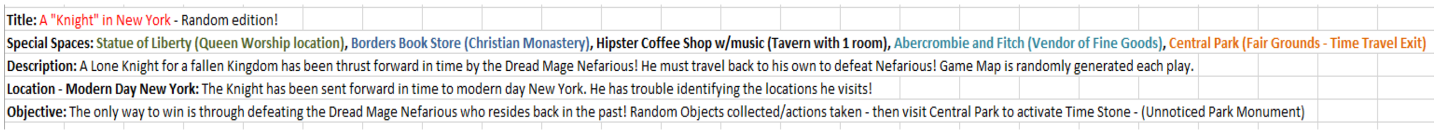
			2nd: FAIL, the Player moved, but was not set in the Space properly 3rd: Pass, the PlayerHere was fixed
	Proper Menu Display after Move	Menu is proper and does not change on fail	Init: FAIL, the menus were swapping on the Spaces that had Special Menus when you pressed a wrong Int 2nd: Pass, fixed the Menu mixup
Space Interaction	Pickup an item in a Space	Place the item in the player pack	Init: Pass, the action attempt was tried, even though the pack action failed. This action was copied to other spaces.
	Attempt adding Gems	Gems get removed	Init: FAIL, the gems were not properly found in the pack. 2nd: Pass, the gems were removed from the pack and “deposited”
	Gem Count on space	Gem count goes up when removed	Init: FAIL, gem count was not incrementing 2nd: Pass, gems needed increment add on each remove of Gem
	Activate Monument	Trigger game end scene	Init: FAIL, the monument was infinitely just sitting there. There was a loop that was an infinite loop 2nd: FAIL, the monument did not recognize the Sword 3rd: Pass, sword bool fixed
Help Menu	Choose Help	Show Help	Init: Pass, but horrible formatting, did not clear 2nd: Pass, Fixed formatting
Item Menu	Choose Item Menu	Show Items	Init: FAIL, items were not properly displaying names. 2nd: Pass,
	Choose Item Menu at start of Game	Show Items as empty even if you have none	Init: FAIL, Did not show anything

			2nd: Pass, added <Nothing>
Map Array Build	Start the Game	Map array gets built	Init: Pass, used this from the Creature assignment
Map Randomizer	Start the Game	Map gets randomly assigned	Init: FAIL, map was attempting to access an invalid address 2nd: FAIL, map was not assigning the special spaces properly 3rd: Pass, map assigned all spaces properly when Int tick implemented
Linked Structure Build	Start the Game	Linked Structure is built assigning N, S, E, W	Init: FAIL, the spaces were not properly assigned 2nd: Pass, fix a j iteration issue and an if statement
Knapsack	Add item	Add item to pack	Init: FAIL, item was duplicating 2nd: Pass, loop fixed
	Remove Item via Park Space	Remove the item	Init: FAIL, could not find the item. Temp ptr improperly used 2nd: Pass, assigned names fix
	Fill pack beyond capacity	Pack will not let you fill it beyond capacity	Init: Pass, Yay! I got this one right!
Game End Trigger	Activate Monument after all actions taken	Game Ends after forced battle	Init: FAIL, this is because the Final Space failed 2nd: Pass, Fixed Final Space generation add/remove
Add/Remove Space	Activate Monument after all actions taken	Player is taken from Park Space and placed into Final Space which has been added in place of the Park Space. Park Space is removed from everything.	Init: FAIL, Player was not properly added to FinalSpace 2nd: FAIL, removal was causing fault 3rd: Pass, fault fixed in deletion where add was referencing a deleted space
Time Limit	Set Time to 3 seconds and let it timeout. Then try to do an action	Timeout will end the game when an action is taken after timeout	Init: FAIL, timeout was not working with clock()

			2nd: FAIL, bool was not properly being set on fail 3rd: FAIL, menu system was not properly exiting after a timeout 4th: Pass, time(NULL) used and menu fixed to break out

Final Reflections:

*As this document is already massive for you to grade, I will discuss some of the problems I had with this assignment, but not all.



Title: A "Knight" in New York - Random edition!
Special Spaces: Statue of Liberty (Queen Worship location), Borders Book Store (Christian Monastery), Hipster Coffee Shop w/music (Tavern with 1 room), Abercrombie and Fitch (Vendor of Fine Goods), Central Park (Fair Grounds - Time Travel Exit)
Description: A Lone Knight for a fallen Kingdom has been thrust forward in time by the Dread Mage Nefarious! He must travel back to his own to defeat Nefarious! Game Map is randomly generated each play.
Location - Modern Day New York: The Knight has been sent forward in time to modern day New York. He has trouble identifying the locations he visits!
Objective: The only way to win is through defeating the Dread Mage Nefarious who resides back in the past! Random Objects collected/actions taken - then visit Central Park to activate Time Stone - (Unnoticed Park Monument)

<u>Project Size:</u>
You may not be able to see this, but because of what I submitted in my initial description, I think I made myself bite off a project that was much larger than a lot of students here. The reason for that is because I chose to have the Map randomly generate. At first this was a novel Idea because I had no idea how intensive this project was. Because of that one line, I spent a huge amount of time getting the Random system up and going and having it properly implement the Linked Structure system.

<u>Time Limit</u>
Firstly, I would like to apologize for the 7 minute time limit. I wanted you to read each thing and take your time. Please leave an additional Flip open or go take a few minute break if you truly want to test the time limit. I thought this time limit was a fair amount of time to try the whole game and test things out. You can always use vim and edit the Menu.cpp constructor and set the time limit to like 3 seconds.

At first I tried to use the clock() function, but it was failing miserably at the last moment (Flip vs Windows). I attempted a few different things to get it up for about an hour or more and ended up giving up on it in favor of the time(NULL) function. This clock works great on flip. As I understand it, there are only a couple drawbacks to time over clock. Something with the daylight savings or so.

<u>Linked Structure:</u>
I don't know if you will disapprove of my method or not, but I was actually happy that I was able to accomplish the Linked Structure built of Spaces via the initial Array generation. I spent a few hours getting the random up and going and ran into a lot of issues with the Link assignments before I figured out what the deal was. It ended up being and issue with the j && statement and I totally missed it! After you spend about 3 hours looking at code your brain starts to slow down. Not so much on the easy code, just the thinking code! This function (makeLinkedStructure) is not as efficient as I had hoped but it gets the job done 100% of the time and is understandable.

<u>Knapsack:</u>

I made mistakes with the Knapsack array and I was getting faults when adding and manipulating the contents of the pack. After reviewing the issue and debugging I found that the problem was in the for loop. When it would find a slot that was NULL, it would continue to fill the pack with the same item making it impossible to add another item later. On top of that, it was not incrementing totalItems so it would tell the user that they had 1 item when they really had 3. There are a couple of places I use break; to get out of loops when something has been accomplished. I am not sure that this is bad practice, but it is needed to stop the loop from doing additional things after it has accomplished the needed action. Sometimes I use while statements to fix that, but it just doesn't work well with looping through an array.

gameComplete bool Menu:
When I first envisioned the project, I had thought I would use a gameComplete bool inside of the Menu. The problem I ran into when trying to finish the game was that there was no way for any of the Spaces Special() moves to activate the bool without knowing about the Menu. I ended up moving the game completion into the Player space since the player was known to both the space and the map.

Traversing Map:
I actually had to debug the traversing of the map structure for an hour or two because it kept getting faults when I tried to move certain directions. It turned out that the whole reason this was happening was not due to a NULL pointer in that direction, but due to the fact that the Player* playerHere pointer was not properly being set. This was causing an issue when you stepped into the space and failed. Correcting the order in which the Player is copied and moved within the setPlayerHere and setPlayerLeave functions in a Space resolved this issue.

Overall:
Overall, a lot of my issues still stem from lack of foresight. I envision something and write down what I think is needed, only to make massive changes later when trying to get the pieces to stick together. I feel like I am getting better, especially after such a large project. Although I spent a cast amount of time on this, I actually enjoyed the learning experience and the problems that I overcame by looking at things in a different light. It certainly makes you learn more when you literally have no one to ask about learning new methods. The TA groups were helpful for me to clarify requirements but of course they are not going to tell you what to do. That is up to us as the creators to come up with.

Side Note:
I appreciate the advice you give and you helpful responses. Thanks for all you have done for me this term and for working with me on issues I have had. I hope you have a long and successful career! *Sorry for the document length!