Ryan Wholey
Final Project

General outline:

- The room class handles rendering of the map
- The room class has a list of mapItems that each have x and y coordinates with an icon.
- The room class will be responsible for serializing and deserializing room maps (static assets)
- The goal here should be to program in such a way where at the end, the developer can create a cool game by creating a series of static map files without ever having to touch the code
- Static maps should denote a map layout with pieces, which maps are adjacent and if there are any actions that any of the pieces on the board should be equipped with
- Whenever anything changes the map will rerender itself, reprinting itself with all of its mapItems
- MapItems will be the base item class, each have an x, y, icon
- The game engine will be a while loop and a queue
- While the is not done, the queue will be polled for actions that will be added to the queue
- When an action gets added to the queue, the game engine will read it from the queue, react to it in some way and then delete the action item
- Every moving piece to the game should be handled with an action on the action queue
- A process should be running to listen for key presses to add directions for the player to move to the queue
- The player is a derivation of the MapItem class with some added features like move (maybe shoot? Maybe sprint? etc)
- The player should also have an inventory (list) of items that can be used in the game
- The game engine should analyze each "MOVE" action and look for collisions between mapItems, then call the appropriate collision methods

test	setup	expected	actual
hit hjkl key puts a player move action in queue	start game	player moves	player moves
change direction changes player icon	start game move	player direction changes	player direction changes
player cannot move through walls	move toward and then into an X	player does not walk through wall	player does not walk through wall
player cannot move through walls coming from any direction	going up going left going right going down	nope nope nope nope	nope nope nope nope

player can move onto door squares '_'	move into door	player icon replaces door icon	player icon replaces door icon
Player can pick up key	1 key 1 player	- key deletes from map - key added to player inventory	- key deletes from map - key added to player inventory
Player can't move through lock (without key)	1 player, 1 key, 1 lock	player blocked by lock	player blocked by lock
Player can move through lock with key & lock gets removed from room	1 player, 1 key, 1 lock	- player moves through lock, - key removed from inventory, - lock removed from map	- player moves through lock, - key removed from inventory, - lock removed from map
Player can move between rooms	walk through a door	- new room loaded - player at the appropriate location on new room	- new room loaded - player at the appropriate location on new room
Room can load a static map	give room a file	room layout loaded	room layout loaded
Room can create items to add to map from static map file	put items L and K on map	room adds L and K to roomItems list	room adds L and K to roomItems list
Player can pick up laser	player walks over a G tile	- G tile removed - Laser added to inventory	- G tile removed - Laser added to inventory
Player can shoot with laser	press f (with laser)	laser projects from player out	laser projects from player out
MapActions can be loaded into queue as reactions from interactions liek collisions	fire laser at a target that has a reaction on it	reaction added to action queue	reaction added to action queue
Lasers change direction when they collide with an angle item	fire laser at / and \ from different angles	laser bounces in approriate direction	laser bounces in approriate direction
Player cant move through fence '#' but laser can	player with laser and fence	- player cant walk through fence - laser travels through fence	- player cant walk through fence - laser travels through fence
Timer ticks down 1 second each second	start game	Timer ticks down 1 second each	Timer ticks down 1 second each

		second	second
when time runs out, losing screen is shown	let time run out	losing screen is shown	losing screen is shown
winning screen is shown when player picks up crown	pick up crown	winning screen is shown	winning screen not shown, however game changed, must walk through door with crown to win
laser can remove block	fire laser at block	- block removed from room	- block removed from room
no memory leaks!	run valgrind	no memory leaks	i can't tell :(valgrind wont run correctly
player can move backwards throughout the game (ensure that the back rooms are set correctly)	go to end, then walk back to beginning	nothing weird happens	nothing weird happens

Reflection:

I think i did a lot of things right and a lot of things wrong here! One of the best decisions I made was to have an ActionQueue that takes MapActions. This allowed me to essentially "store actions for later". I could program a MapItem to have a stored action on it and then when something happened later in the game, I could get that action and just pop it on the queue. It made collisions and stuff uniform across items and really made it quite simple!

One of the things I **know** I didn't do right is my liberal use of threads. There are multiple threads that all have access to the ActionQueue but it's wholly un thread safe. I looked into using a mutex but I came to the decision that I only had a couple weeks and threads are a whole new concept that I really didn't have the time to get into enough to do a good job. The biggest downside gamewise to not being threadsafe around the action queue was that I kept getting seg faults when I allowed more than one missile at a time on the game board. Each missile that is fired has its own while loop to check for collisions as it travels across the room, putting actions onto the action queue. The more missiles fired, the higher the likelihood of a seg fault. This really put a stop to my idea of having enemies since they would probably have to be on their own thread as well so they could move around without blocking the main thread.

I really took advantage of subclassing the MapItem class. Most items didn't need anything special outside of what the base MapItem class already offered but having the getType be a virtual pure function allowed me items to react differently with other sibling instances on collision or postcollision etc.

I tried to use all of the different types of std containers that we'd learned about in 161 and 162. I used the queue for the game engine as the ActionQueue to interpret actions as the game progressed. I used a list to handle roomItems. The Room instance has a list of items that it runs through each render and puts them onto its layout printing. I found iterating over the list to be a nuisance. I kept wanting to just use a for loop but most of the documentation recommended that I use an iterator. I found this interaction clunky and kind of a pain. I kept wishing that I'd just used a vector instead but for the sake of the exercise, I'm glad i used a list. I used a stack for serializing the layout. When reading from the static map file, if the line was read to be an "action line" (a line that denotes an action that should be stored in a mapItem), I pushed the raw string onto a stack. Then after reading the static file, I used a while loop until the stack was empty to read and implement each action string. I used a vector for the player inventory and the dynamic room layout (after loading a static map) in the room class.

I had some real trouble with valgrind for some reason. I don't really get it. I used to be able to run valgrind on my mac at home but I had to upgrade to Sierra and valgrind essentially broke. Bummer but I thought I could just test on flip. No luck there either! For some reason my code loads EXTREMELY slow and it's basically impossible to test. The only thing different I did in this lab was to use the ncurses library and use multi threading. It looked like in the documentation for ncurses that when run with valgrind, it reports memory leaks anyway so I had to make peace with the fact that I wasn't going to be able to test for memory leaks and that I would just take the hit from you, my ta, when you went to check it. I promise that my game runs fine without valgrind though!

Overall I really enjoyed this project. I really went off the rails and didn't implement all of the guidelines exactly like I have in the other projects, but I stand by my decision! For the most part my implementation did not stray far from my original plan. I intentionally left it a bit loose but I did have to make some concessions. I had originally thought that every mapItem would have an ID and that ID would be how other mapItems should interact with it but I quickly found that coordinates were a far easier way of finding an item (look through each of Room's roomItems and find matching coordinates).

Another big one was that I originally thought each static map would be its own room instance and they'd all be linked together in some sort of dynamic data structure but this got messy quickly. Transferring map items between rooms wasn't very straight forward. It ended up being much more convenient to just "clean" a room on room move, save the map state in a file, load a new map layout and place the player at the correct spot.

I could go on and on about this project. I hope you enjoy playing it! I hope it's not too difficult to play. I'll probably include a walkthrough in my file so no one will get stuck. Thanks for an awesome quarter!