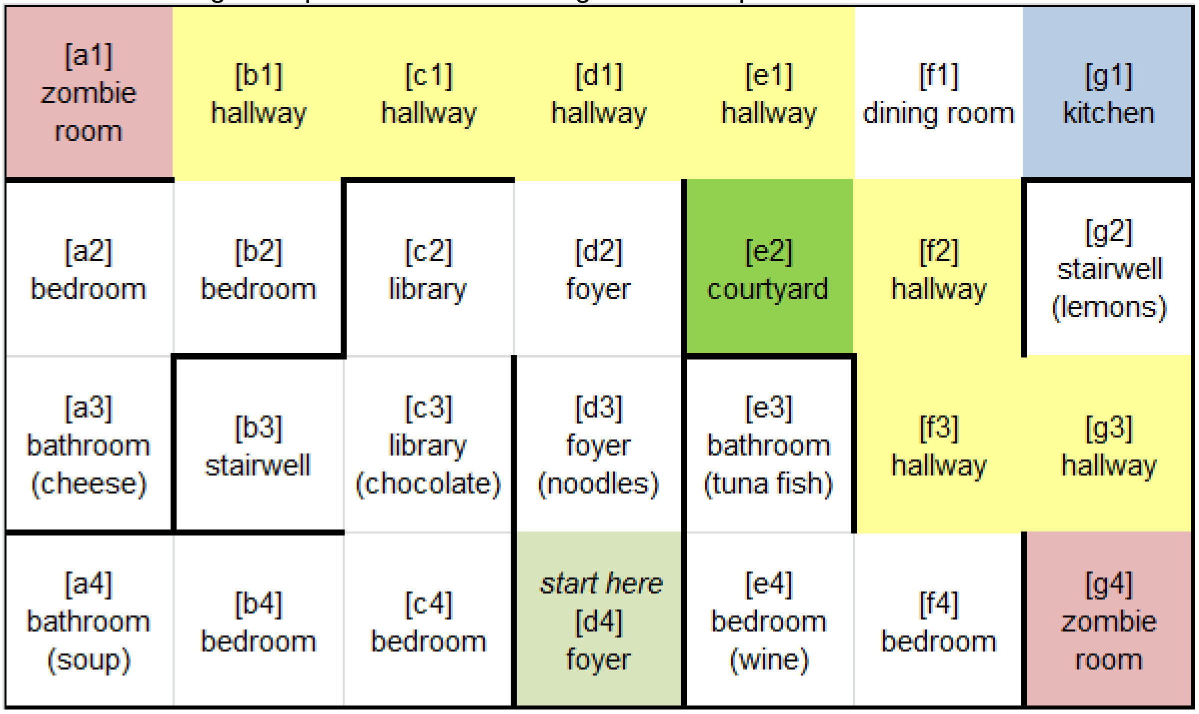
Gunnar Martin
Report for Final Project
CS162 Fall 2014

I. Understanding
   a. The project: Our final project assignment was to create a “game” with series of rooms for a player to move through. The environment we are to create would be a “room” class with pointers in 4 directions to other “rooms.” The environment needed to have a minimum of 10 rooms. We were to create a “room superclass” that would be the parent class to the derived rooms. The derived rooms were to have characteristics different from other derived rooms. The player has a bag which he can store objects in and the objects must help him win the game.
   b. The challenge: Though we have worked with linked lists, pointers and STL structures before, this project asked us to bring them all together. I worked off of a previous lab to get the north, east, south and west navigation, but I knew that I would need to figure out the rest of the functionality during design and testing.

II. Design
   a. Overview
      i. Premise: My game would have a player in a mansion, navigating the 2-dimensional space, moving from room to room by moving north, east, south and west. This mansion has two zombies in it and the player must figure out how to get rid of the zombies.
      ii. Map: I started designing my game by first drawing out the map. Once I had the map drawn out, I know I would be able to design my classes and then gameplay functionality. My map is set up in a 2D grid. The mansion has 28 rooms (7 wide by 4 deep). If a player tries to navigate through a wall (ex: tries to go East from d4), he will run into the wall and remain in his original square. Here is an image of the map:



[a1] zombie room	[b1] hallway	[c1] hallway	[d1] hallway	[e1] hallway	[f1] dining room	[g1] kitchen
[a2] bedroom	[b2] bedroom	[c2] library	[d2] foyer	[e2] courtyard	[f2] hallway	[g2] stairwell (lemons)
[a3] bathroom (cheese)	[b3] stairwell	[c3] library (chocolate)	[d3] foyer (noodles)	[e3] bathroom (tuna fish)	[f3] hallway	[g3] hallway
[a4] bathroom (soup)	[b4] bedroom	[c4] bedroom	*start here* [d4] foyer	[e4] bedroom (wine)	[f4] bedroom	[g4] zombie room

      iii. Items: Each room can hold one item. On the map, the starting items are in parenthesis. If the player is in a room with an item in it, he can pick up that item and put it in his backpack. The packpack can only fit 4 items. The player can also discard items from his backpack onto the floor, but he can’t discard an item into a room that already has another item in it.
      iv. Goal: The goal of the game will be for the player to navigate around the mansion and collect the items necessary to make tuna fish casserole (tuna-fish, noodles, cheese, soup). Once the correct 4 items are in the backpack, the user must go to the kitchen to put them in the oven to bake. After the tuna-fish casserole is baked, the player must

place casserole on all the shelves of the hallways, in order to draw the zombies out into the courtyard to fight and destroy each other.

v. Hints: The user will be given hints of varying degree in the following rooms: foyer, zombie rooms, courtyard, kitchen and hallway.

b. Classes: For my room classes, I knew that I would need one room "superclass" that never actually get implemented and then derived room classes that are the objects used for the actual rooms.

   i. Superclass: I called my room superclass 'room'.

      1. I gave it the following member variables

         a. cell address – for the cell address, ex "a2"
         b. North, east, south and west pointers to other rooms
         c. Room type – for the type of room, ex "foyer"
         d. Item – for the item that is in that room for the user to pick up, ex "noodles"

      2. I created several member functions for this base class

         a. Get functions for type, item, address and room pointer
         b. Virtual functions for the hallway classes get and set functions
         c. A function to display the user's standard options in every room
         d. A virtual function to displayer the user's options in that specific derived room
         e. Virtual function for "welcome" text to display when player enters a room (specific to each derived class)

   ii. Derived Classes: The derived classes communicate information to the user as they enter each room by using the virtual welcome function to display text.

      1. Kitchen: The kitchen class of room needed to be different from the others, since the user needed to have the option to empty his backpack contents into the oven. I used the virtual function from i.2.d above to accomplish this. Also in the kitchen's function for the welcome text, I included a recipe for the casserole as a hint for the user.
      2. Hallway: The hallway class of room needed to be different from the others, since the hallway need to have shelves for the player to put the baked casserole on in order to lure the zombies into the courtyard. The hallway class has a unique Boolean member variable called 'has_casserole' which is set to false originally, but then set to true when the user puts casserole on that shelf. In the main program, I wrote a function that tests to see if all of the hallway rooms have casserole on their shelves. If they do, then the victory conditions are met and the game will end. I designed the map in a way that it would make logical sense to fill all the hallways with casserole to lure the zombies in. (ie. There are no hallway class rooms in random places, only between the zombies and the courtyard).
      3. Bathroom: The bathroom class has a welcome function that welcomes them into the bathroom. The bathroom class also gives a hint that something is on the floor, since the game starts with items in all the bathrooms.
      4. Bedroom: The bedroom class has a welcome function, but no hints.
      5. Courtyard: The courtyard class has a welcome function and also gives the player a hint about zombies being able to fight in the courtyard.
      6. Foyer: The foyer class gives the user the information that there are zombies in the mansion and gives the user an idea of what the goal could be.
      7. Library: The library class has a welcome function that does not give a hint.
      8. Stairwell: The stairwell class lets the player know that the stairwell is blocked and he cannot go upstairs!
      9. Zombie Room: The zombie room class tells the player about the zombie in that room and also hints that the zombie likes casserole.

c. Actions: For the gameplay functionality, the player is continuously presented with a menu of options until he wins the game. Each room has a set of options that are always available, with potentially different options depending on the class of room the player is currently in.

   i. Here are the options throughout the game

      1. The standard options are:

a. (g) go to another room
   i. (n) move north
   ii. (e) move east
   iii. (s) move south
   iv. (w) move west
b. (b) check my backpack
   i. This displays the backpack contents, or “empty”
c. (r) drop item from backpack
   i. (1) item 1, if applicable
   ii. (2) item 2, if applicable
   iii. (3) item 3, if applicable
   iv. (4) item 4, if applicable
d. (d) look down
   i. (y) pick up item, if applicable
   ii. (n) don’t pick up item, if applicable

2. The hallways have an option:
   a. (s) check the shelf
      i. (y) place casserole, if applicable
      ii. (n) don’t place casserole, if applicable
3. The kitchen have an options:
   a. (o) empty backpack contents into oven

d. Backpack: For the backpack, I used a vector of strings.
   i. Pick-up items: Before picking up an item, the backpack was “checked” to make sure it was not at capacity. If it was full then the item couldn’t be picked up. When the user chose to pick-up and item, that string was loaded into the vector, and the string “nothing” was set as that room’s item variable.
   ii. Drop items: When the user chose to drop an item, the item was removed from the vector and the string of the item was set as that room’s item variable. If there was already on item in that room, the player would not be able to drop that object
   iii. Loading to oven: If the player had the correct ingredients to load into the oven, all 4 items would be cleared from the backpack and replaced with one called “tuna-fish casserole.” This item cannot be removed from the bag, but now casserole can be set on shelves in the hallway.
e. User Option Error Checking: The flow of action is all controlled by the user select valid options from his current menu. For every set of options that the player is given, I build in a function to check and make sure the player is entering a valid character from that specific menu. If he enters in invalid character, he is met with an “invalid, please re-enter:”.
f. In-line Header File: Though most examples of class header files that we were taught used separate implementation and interface files, for the purpose of this project, I chose to design my implementation and interface into one file. I feel that by coding all of my functions “in-line” I am able to keep much better track of my member variables and function than if they are in a separate CPP file. Since I have no security reason to use two files, I opted to stick with one.

III. Testing: Please see separate testing file PDF
   a. I completed my testing by navigating to each of the 28 rooms and exhausting every combination of variables in each room to make sure that the program was given the feedback I expected. My testing plan is in a separate PDF, since it is quite a large document. For each room, I listed its type, the user inputs I was entering in (ex: “g”, “n” from the user will execute move north), the state of the item on the floor, and for hallways, the state of the shelf (whether it had casserole or not) and the state of the casserole (whether it was in the backpack or not). For each combination of variables and user inputs for each room, I listed the expected outcome and the observed outcome.

IV. Implementation
   a. I loaded my code from my IDE to the Flip server, and make sure that it compiled and ran using the makefile that I included.

V. Reflection

a. Overall, I felt that this project was very challenging, but helpful in developing my understanding of linked structures, inheritance, OOP design and STL containers. The most challenging part was figuring out how to use the linked structure in conjunction with the “backpack” (I used a vector) and then also make sure to test for a winning condition using my vector of hallway objects. In the end, I felt this it was very good final project to solidify my understanding of linked structures, inheritance, polymorphism and OOP. For the first time in 20 weeks, I feel that I have a good handle on pointers and on their syntax, which was a subject that I never thought I’d get this comfortable with. Throughout this class, I have felt pushed to become a better programming and my understanding of C++ has improved vastly. I’m excited to learn more computer science!