Nathan Zimmerman
03/21/2017
CS162 400 W2017

Final Project – Project Plan

Identify Problem

Task create a text based game that allows a player to move through a series of rooms and interact with special spaces. The character will need to gather items and reach a goal to complete the game. The game should have time limit. Also, there must be a menu function implemented to control the game. The program must implement polymorphic design principles as well as making use of pointers to allow a character to travel to different rooms.

Identify Inputs

User controls to move, 'w', 'a', 's', 'd'. Also, item menu input 'i' and action input 'e'. Lastly menu input 'm'. A quit command 'q' will also be implemented. Lastly for menu options the user will press a number corresponding to the menu's option.

Identify Outputs

Output print a map that is an array of characters to the screen. Print enemy character movements on the map as well as user character movements to the map. Print menus to allow user to see item options and game menu choices. Also, print results of player interaction with special squares on in the form of an updated map. Game should output information about key events including, loss of life, game steps, and game state (i.e. win or loss).

Character Move Algorithm

For this game, I will try and implement a real time turn based move algorithm. I will make use of the ncurses library to allow the game to update information about the state of the game even when the user is not inputting characters. For the user when characters 'w', 'a', 's', 'd' are entered a switch statement will check the current direction of the player and rotate them so that they are facing the correct way according to input. If the character is already facing the correct way, then the game will cause the player to move one array index in the appropriate direction. For instance, if the player types w and the character is currently represented as '^' then the character will move one index position. In this case the y index will be decreased by one. A general map will be stored in a vector of strings. The appropriate index position of the character will then be overwritten with the character's symbol. After each action the map will be reprinted with only the character's current position update to reflect the appropriate character symbol. The correct character symbols are 'v', '>', '^', '<'. The character's current index position will be tracked as two integer private data members that can be accessed with get functions. When a user inputs a move character these data members are incremented or decremented to track the character's current position. When a character moved to a different room, the private index data members tracking x and y position will be updated to reflect the new map.

Zombie Algorithm

Zombie's will move independently of user input. The ncurses library will make this possible. When a user is not inputting anything, a conditional statement will be triggered that will call an overloaded update map function that takes no parameters. The update map function will call a zombie move

function that will have the zombie move one index position every second. The map will be updated in the same way the it will be for the user. A general map template will be created as a vector of strings. The zombie character char will override a specific character in the vector corresponding the zombies current position. Each time the zombie moves that map will be updated. If a zombie encounters a wall a setter function will multiply a zombies private data member representing their move speed by -1. So instead of their index being incremented by 1 it will become -1 x 1 = -1. The zombie will therefore move in the opposite direction than the way he was moving before encountering a wall.

Interact With Map Algorithm

The user will be able to interact with light switches 'L', debris 'O', Keys 'K', and Speakers 'S'. A function will check what the space in front of the user is depending on the specific user input 'w', 'a', 's', or 'd'. For instance, if d is pressed the space at index x, y + 1 will be checked to see what character that space is. If the space is a special character that the user can interact with then a private data member bool for the specific room class will be triggered. For instance, if 'e' is pressed and the next character is a 'L' then a light switch bool will be triggered to true and the map will be updated accordingly. Conditionals will be present in the print map functions to always check for successful interactions. Characters will be replaced on the map as appropriate to represent changes that have taken place.

Lights On or Off Algorithm

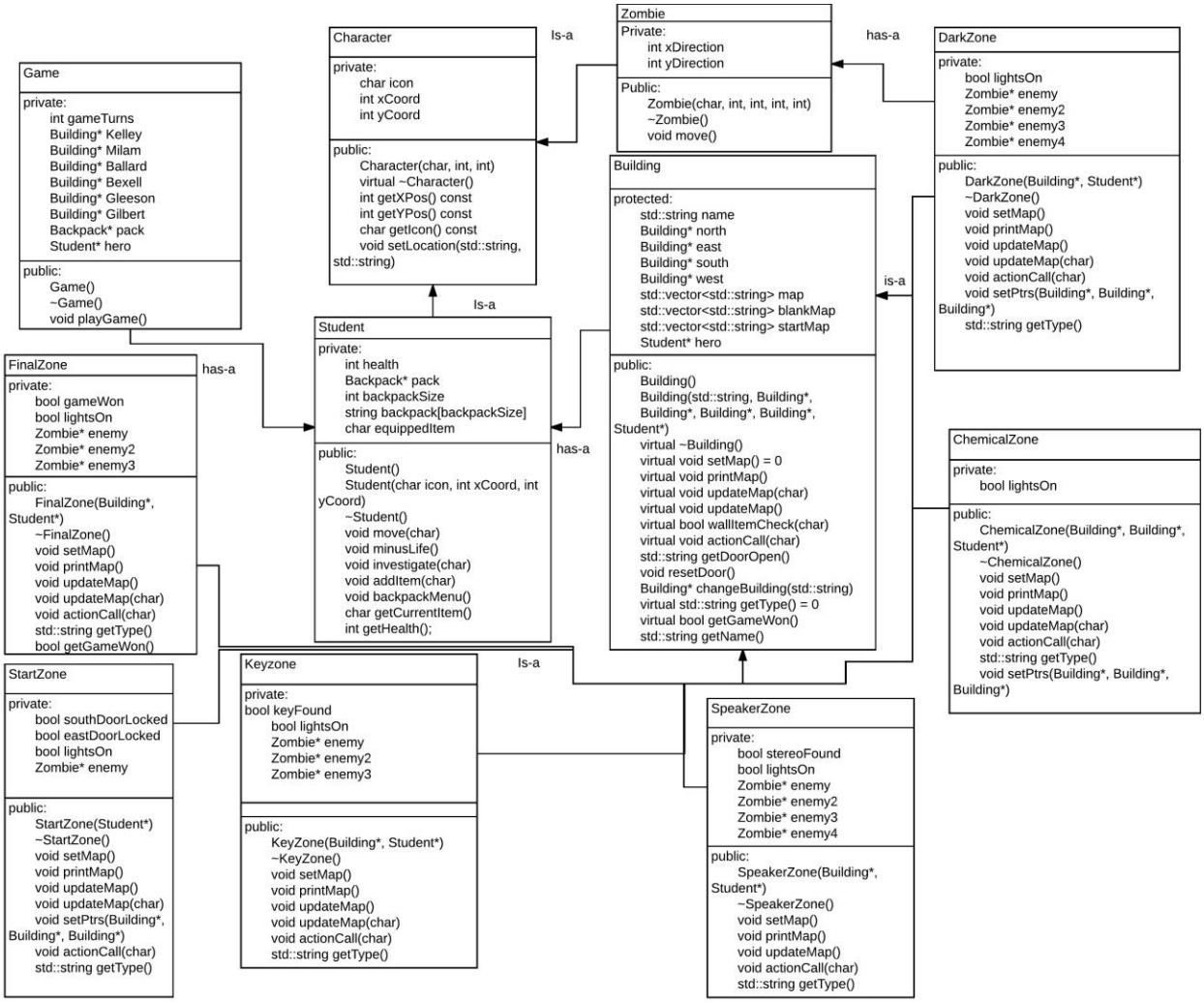
Two maps will be created in each room classes set map function. One map will show nothing but walls. The other map will show special characters such as doors and keys. A for loop will loop through the map that shows just walls and will print squares that are at a distance of 5 from the characters x axis and 3 from a characters y axis. Then a for loop will print squares from the map containing the special characters. The second loop will only print squares within a distance of 3 from the y axis and 5 from the x axis. This way the character will only be able to see a certain distance in a dark map. When a light switch is interacted with a bool will be triggered and only the map showing all special characters will be printed. The complete map will be printed so that the character can now see everything.

Program Design

The main.cpp file will create an instance of the Game class. The Game class will call a play game member function to initiate the game. Once the game is initiated a message will appear that goes over the rules and objectives of the game. The user will press a key to continue, at which time the game will commence with an introductory message. The game function will run a while loop that will call an update map function until the user enters a key that is present in a switch statement to monitor user input. If the user does not press a key then a conditional statement will recognize this and an update map function will be called. This update map function will make use of the Zombie algorithm described above to allow the zombie to move despite the user not entering in anything. Next when a user enters a move command or a command to access one of the various menus, the switch statement will pass the command into an update map function. The update map function will be called on a Building pointer. The pointer will point to a derived class for the Building base class. The Building base class will be a pure virtual class. Each derived class will have a special implementation for the update map function that will be called from the main Game class. In each of the derived classes there will be special functions used to track special characters such as keys. Using the 'interact with map' algorithm described above, each

of these separate classes will track when a user has interacted with the map. When a user encounters a door, ‘d’, the main switch statement in game will check the current map the user is in and the position of the door. A conditional will then update the current map to become one of either four Building pointers pointing to other derived classes of Building. In this way the user will move seamlessly between maps. The user will equip and use their phone to receive hint on how to complete the game. The user will continue through rooms avoiding zombies and moving and collecting items until they reach the final room. In the final room they will reach the final antidote object thus ending the game.

Class Hierarchy Diagram



Test Results

Test Case	Input Values	Purpose	Expected Outcome	Outcome
Test Map	Call printMap function for: StartZone, KeyZone, SpeakerZone, DarkZone, ChemicalZone, FinalZone	Check that map function is being displayed correctly.	Program displays the map including special characters, character, and enemies	Program displays the map including special characters, character, and enemies
Test lights on off	Call printMap function with lights off and then on for complete list of the 6 maps. StartZone, KeyZone, SpeakerZone, DarkZone, ChemicalZone, FinalZone	Check to see that the dark map does not print any special characters. Check that the character can only see a radius of x = 5 and y = 3 squares around him.	printMap function correctly prints a map containing no special characters when the lights are off and a map containing all special characters when the lights are on.	Error: Map printing area not near character. Fix, reversed the x and y axis input into the for loop printing the map.
Test character move	'w', 'a', 's', 'd'	Check to see that character rotates and moves the appropriate number of squares on the map.	If a character is not facing the direction of the input command, then they will rotate. If they are facing the direction of the command, then they will move one index value.	If a character is not facing the direction of the input command, then they will rotate. If they are facing the direction of the command, then they will move one index value.
Test Doors	Test all rooms north, south, east, west doors. Test to make sure character cannot enter locked doors.	Check pointers point to the correct rooms. Check that character is placed correctly in the new room.	Door function causes pointers to be used to assign the current room to the correct new room. Character cannot enter locked doors;	Bug: Entering door causes character to spawn in incorrect position in new room. Fix, had to change the x, y coordinates in change room function.
Test enemies	Test all rooms. No user input, input to place character in front of zombie.	Check that with zombies move correctly with no user input. Check	Zombies move at the correct speed and in the correct direction.	Bug: Zombies not moving. Had to update zombie constructor to

		that zombies reverse direction when a user gets in their way. Check that zombies reverse direction when hitting a wall.	Zombies reverse direction when encountering a player or wall.	make sure that their data members are initialized. Bug: Zombies move to fast. Fix: Had to use a counter to ensure that zombies only move when the counter reaches a certain amount.
Test Character life	Input: Move character in front of a zombie. Input move character in a chemical spill.	Check that character life is reduced correctly. Check that when life reaches zero the game ends.	Life is reduced by one point per damage event. Life reaches zero and the game ends.	Bug game did not end when life reached zero. Fix: Changed conditional in Game class to correct value.
Test steps counter	Complete game cycle. Move character through each room.	Test that the step counter is high enough to complete the game. Test that when the counter reaches zero the game will end	Step counter set high enough to complete the game easily. If the step counter reaches zero the game ends.	Step counter set high enough to complete the game easily. If the step counter reaches zero the game ends.
Test Item container/Equip	Press 'I' to open container. Press 'w' and 's' to select container items. Press 'e' to equip item. Press 'e' to use equipped item.	Test that the item container registers new items, allows user to equip items, and once equipped allows the user to use the items	Menu opens. Menu allows user to make selections. Items equipped. Equipped items function	Menu opens. Menu allows user to make selections. Items equipped. Equipped items function
Test Game winning item	Press 'e' when facing the zombie antidote.	Check that game ends once the goal has been accomplished.	Game ends and menu comes up to allow the user to play again.	Game ends and menu comes up to allow the user to play again.
Test Debris pickup	Press 'e' when facing debris 'O'. Move debris around. Press 'e' drop debris.	Check that debris can be picked up, moved, and dropped.	Debris is successfully moved and dropped. Character behaves as normal when debris is moved.	Bug: Character can move through debris when holding debris. Fix: changed conditional to check two spaces

				in front of the character to make sure there is nothing in his way.
Test menu function	Input: 'm' to bring up menu. '1' , '2', '3', '4' to go through menu options	Check that menu options function correctly. Check that menu opens and exits multiple times without issue.	Menu option play game returns to game. Menu option controls brings up the controls message. Menu option quit game quits the game. Menu option current steps shows steps left.	Bug: Menu options for steps was changing for each room. Changed steps data member to Game class instead of Building.
Test Input	Input -1000000, 0 1000000, ;lkjjkss, $$$	Check that random input does not cause the game to crash. Check that random input does not trigger any functions.	Random input has no effect. No functions trigger.	Random input has no effect. No functions trigger.

Reflection

Planning and implementing this program has proven to be a tremendous challenge. For this final project, I wanted to do something special, to go above and beyond what was required to create a fun user experience. Part of going above and beyond was implementing a real-time component to the game. Learning how to use the ncurses library to implement this real-time component, while also programming what is my largest project to date, was certainly a challenge. Combined with the real-time component of the game were functional components, such as implementing a map where a character can only see a few squares at a time. Another functional component was being able to pick up objects and move them around, allowing both the user and enemies to interact with these objects. These factors combined to make this project the most time consuming and difficult task yet.

As a result of the difficulties encountered completing this project, I have learned a couple of important planning concepts. Firstly, I recognize that implementing the extra functional features associated with real time movement and real time object manipulation where very time consuming. As a result, I have had less time to debug than I normally do. A valuable aspect of planning a project is sticking to project specs, completing requirements, and adding extra features once all core requirements are complete. In planning from the onset to add difficult features I unknowingly increased the time burden of the project beyond what I normally allow for. In the future, I will more carefully limit functional components of a project to adequately account for time constraints.

In regards to debugging, the large scale of the project and complex nature of the functional components proved to make debugging more difficult than I had originally anticipated. Managing the freeing of memory has proven to be a challenge due to the large scale of the project. In attempting to solve the problem I have traced the location of all usages of the new operator and attempted to match them with the appropriate delete command. Valgrind has helped me eliminate most memory issues. Valgrind currently shows no errors but unfortunately is showing more allocs than frees. I am unsure whether the issue may be caused by the use of the ncurses library. I have looked up the issue and found this source: http://stackoverflow.com/questions/32410125/valgrind-shows-memory-leaks-from-ncurses-commands-after-using-appropriate-free. Stating "Perhaps you used a tool such as dmalloc or valgrind to check for memory leaks. It will normally report a lot of memory still in use. That is normal.

The ncurses configure script has an option, --disable-leaks, which you can use to continue the analysis. It tells ncurses to free memory if possible. However, most of the in-use memory is "permanent".

Any implementation of curses must not free the memory associated with a screen, since (even after calling endwin()), it must be available for use in the next call to refresh(). There are also chunks of memory held for performance reasons. That makes it hard to analyze curses applications for memory leaks. To work around this, build a debugging version of the ncurses library which frees those chunks which it can, and provides the _nc_free_and_exit() function to free the remainder on exit. The ncurses utility and test programs use this feature, e.g., via the ExitProgram() macro."

In choosing to use ncurses I was not aware of these issues. It has been a valuable learning experience and in the future in my planning phase I will do more to research the libraries I am using to make sure that no such issues arise.