Matthew McLean
March 21, 2017
CS162 Winter 2017

Final Project Design and Testing

Requirements:

1. Series of rooms/spaces to move through:
   - Each is a class with at least 4 pointer variables that link to other spaces.
   - Has appropriate data members.
   - At least **6** spaces of at least **3** different types. (3 derived classes with special functions).

Satisfied by the Room class, and the derived Courtyard, AdminBuilding, Gate, Library, ProfessorRoom, ScienceBuilding, and SecretRoom classes. All room classes have an explore function. Although this function has the same name, defined in it are special actions for each different room.

2. Implement goal for player to achieve

Close the portal/gate by finding the professor and collecting the Research Notes and Ancient Tome.

3. Keep track of player status

The Player class holds the inventory (max size of 5 items) and various game state variables.

4. Time limit

Sanity drains with every action.

5. Interaction of player and space structures

SecretRoom only accessible after solving combination in Library.

6. Menu function to run the game

Made a menu function in the Room class, to allow potential menu changes based on what room the player is in. Also created the sub-menu function travelMenu() for when the player wants to move to a different room.

Game Design:

The player starts in a central hub area (the Courtyard). A menu is printed to allow them any of the three options:

1) Travel:
   - Lets the player move any of the 4 cardinal directions, as long as there is a room in the chosen direction, and that room is visible and accessible.

2) Check Inventory:
- Lets the player view what items are currently in their inventory. Has a maximum size of 5 (although this is never reached during the game).

3) Explore:
- Lets the player learn more about their current room, or interact with the room's specific plot actions. For instance, the player must explore in Professor Harvey's office to fight a monster and gain the Professor as an ally.

At certain parts of the game, the player must explore to gain the Professor as an ally, gain items, or trade items. Once all required conditions are met (has Ancient Tome and Research Notes, and is with the Professor) the player must travel to the gate and explore to beat the game.

Class Design:

Game

Private Member Variables/Objects:
A player object, room pointers for each of the rooms in the game, used to set up the map in the Game constructor. An additional room pointer currentRoom, which points to the current room, changing whenever the player travels around the map. A boolean gameOver, used as a flag to end the game. A const int windowHeight, used for printing functions that need to know the size of the terminal window in which the game is being played.

Public Member Functions:
- Constructor/destructor: The constructor sets up the connections between the rooms, creating new instances of each room using the private room pointers. The destructor deletes each of these rooms.
- startGame(): Displays the main menu for the game, getting the user's choice to play or not. Returns a boolean true if the user wants to play, false if they do not.
- travel(): Takes the room pointer where the player wants to travel to as a parameter. If this is null, prints that there is nowhere to go that direction, and exits. If not null, then it checks if the room is visible and accessible, outputting proper messages for each if either check fails. If both are true, then the currentRoom pointer points to the passed room pointer.
- refresh(): Lowers sanity by one, then checks the player's sanity and health values and whether the gate is closed or not, setting the gameOver statue to true if health/sanity are reduced below 0 or the gate is closed. Also checks if the secret room has been unlocked, changing the accessibility and visibility if it has.
- run(): Drives the game. Executes the intro sequence, then loops over getting the user's menu choices, executing the desired action for whatever room the player is in. After the desired choice is executed, calls the refresh function. When gameOver is set to true, exits the while loop and enters an if-block which prints the appropriate game ending graphic, win or lose.

Player

Private Member Variables/Objects:
A vector of strings holds the player's inventory. There are integers for sanity and health, and an unsigned int maxBagSize to limit the player's inventory. There are 6 game state variables held here as booleans: tome, monster, papers, professor, unlockedRoom, and closedGate, used to indicate the status

of each, which is needed in the Game::run() function to determine if the game is over. They are also needed in the Room::explore() functions to determine how the world reacts to the player in different conditions.

Public Member Functions:

- A basic constructor
- Getters and setters for all variables
- addItem(): Adds a passed string to the inventory vector, if there is room. Returns true if there was room and the item was added or false if there was not room and it was not added. Checks if key items are added, setting the relevant game state variables when this occurs.
- removeItem(): Searches the inventory for the passed string, removing the first item it finds matching the passed string. Returns true if item was found and removed, and false if item was not found.

Room

Private Member Variables/Objects:
The room class is composed of 4 Room pointers named as the cardinal directions. This provides the layout of traveling room to room. It also contains two bools to determine whether the room is visible and accessible. It has 3 strings, one for the room name, one for the name of the graphic file, and one for the text that is written below the graphic. It also has a vector of strings to hold the menu options.

Public Member Functions:

- Getters and setters are present for each of the private variables listed above.
- explore(): a pure virtual function, as the explore functionality is different in each of the 7 derived room classes.
- printState(): prints the graphic and blurb, using the getPadding and printGraphic function in the utility functions file to format the output.
- printMenu(): prints each menu string from the menuList vector.
- travelMenu(): prints each possible room to visit if visible and accessible, accessing the current room's cardinal pointers.

Room-derived classes:

Private Member Variables:

The Library, ProfessorRoom, and SecretRoom classes have the boolean firstVisit, as they have logic in their explore functions which only occurs the first time the player explores the room with the correct items.

Public Member Functions:

All sub-room classes have a constructor, which uses Room class setters to set the name string, the graphic string, and the blurb string. The values of each are unique to each room. The class SecretRoom additionally uses the setAccessible and setVisible functions of the Room class to, as the name suggests, make the room unaccessible and invisible until a certain condition is met.

The explore function for each room takes the player pointer as a parameter, giving each room access to the player's inventory and game state variables stored in the player object.
Each room's explore function contains unique logic that allows for the different interactions in each room. For each room, this logic at least determines which graphic and blurb to print when the player explores, based on the game state variables and the player's inventory.

Following is logic of specific rooms:
Courtyard/AdminBuilding: Blurb changes based on game state variables.
ProfessorRoom: Contains the logic for a fight sequence with a horrible monster:

- Checks firstVisit variable so fight only happens once. Sets the graphic to the monster graphic, and the blurb to the monster blurb. Prints graphic, blurb, and the different attack strategies the user can take and validates user input. Sets the blurb depending on player's choice, then prints the graphic/blurb and the player's health/sanity. If the player is alive and the monster is not defeated, loops back to getting user input. If the player dies, exits function, where control returns to Game::run(), where lose screen will be printed. If the player is alive and the monster is defeated, proper graphic and blurb is set, the Monstrous Corpse is added to the player's inventory, and control passes back to Game::run().
- If it's not the first visit, then the blurb is changed dependent on game state variables.

ScienceBuilding: Contains logic to trade the Monstrous Corpse for the Research Notes.

- Checks if the player has the professor and has the monster corpse in inventory. If they do, executes dissection chain of graphics and blurbs, waiting on user to press enter between printed scenes. The Research Notes are added to the player inventory, and the Monstrous Corpse is removed.
- If the player doesn't have the monster, the blurb is changed based on the rest of the game state variables.

Library: Contains logic to solve a combination lock to reveal and unlock the secret room.

- Checks if player has the professor, has the Research Notes, and has not yet unlocked the secret room. Then checks if firstVisit is true. If these checks are passed, then a series of graphics and blurbs is displayed, until reaching the combination lock screen, which lets the user decide to solve or leave. If all but the firstVisit check passes, then the graphic/blurb leadup to the combination lock screen is skipped, going directly to the menu with solve or leave options. If the use leaves, the graphic and blurb are set back to their original states.
- If the user chooses solve, they are prompted to enter the 3 digit combination. If it's correct, the secret room is unlocked, changing the game state variable held in the player object. Graphics/blurbs indicating such are printed, then the function exits. If it's incorrect, explore displays graphics/blurbs indicating as such and the function exits.
- If the user has already unlocked the secret room, blurbs are set depending on other game state variables.

SecretRoom: Contains logic to add the Ancient Tome to the player's inventory.

- If the firstVisit bool is true, the Ancient Tome is added to the player's inventory, and the graphic is updated.
- Otherwise, the blurb is set to a comment by the professor, printed, then changed back to the default.

Gate: Contains logic for the final graphic/blurb sequence that ends the game.

- Checks if the player has the notes, the tome, and the professor. IF this check passes, the final sequence of graphics/blurbs is printed.

- Otherwise, the graphic/blurb is set depending on game state variables. If the player doesn't have the professor, they lose 5 sanity as well.

Utility Functions

A file was made to hold various utility functions used by the game to consolidate code that was found to be repeated often.
These include:

- validateInt(): A barebones integer validation tool. Reads in an integer. If the integer is within the min and max passed to it, the integer is returned. Otherwise, -1 is returned.
- printGraphic(): Prints 200 new lines to clear the terminal. Then tries to open a file matching the passed string, outputting the contents to the terminal until EOF is reached.
- getNumLines(): Tries to open a file matching the passed string. Increases a counter for each line in the file, then returns this number.
- getPadding(): Calculates the padding an image requires based on passed values for the number of lines to be displayed, the size of the terminal window, and the number of lines in a file matching the passed string (as determined by getNumLines()).
- calibrateScreen(): Simply prints a graphic that consists of an outline matching the ideal terminal window size so the user can resize their window. Waits for an enter input, then exits.

Unit Testing:

Player Class

Function	Expected Outcome	Pass/No Pass (P/NP)
Player::addItem()	Item adds to bag if room. If key item, game state variable should be changed. Returns true if added and false if no room.	P
Player::removeItem()	Item removed if there, otherwise return false.	P

Room classes

Function	Expected Outcome	Pass/No Pass (P/NP)
Room::printState()	Print the graphic, print the current blurb.	P
Room::printMenu()	Print all menu items, numbered.	P
Room::travelMenu()	Print all locations if visible and accessible.	P
Courtyard::explore()	Print correct blurb/graphic based on game state variables.	P
AdminBuilding::explore()	Print correct blurb/graphic based on game state variables.	P
ProfessorRoom::explore()	Print correct blurb/graphic based on game state variables. If firstVisit, fight monster.	P
ScienceBuilding::explore()	Print correct blurb/graphic based on game state variables. If player has monster, execute dissection scene, remove	P

	monster, add notes.	
Library::explore()	Print correct blurb/graphic based on game state variables. If player has professor and papers, and the secret room has not been unlocked, execute combination lock path.	P
SecretRoom::explore()	Print correct blurb/graphic based on game state variables. If first time, add Tome to player inventory, change graphic to tomeless one.	P
Gate::explore()	Print correct blurb/graphic based on game state variables. If player has Tome, Notes, and professor, initiate finale sequence of graphics and blurbs.	P

Game class

Function	Expected Outcome	Pass/No Pass (P/NP)
Game::startGame()	Print main menu and correctly validate user input	P
Game::~Game()	No memory leaks, determined by valgrind	P
Game::run()	Execute intro, loop menu choice/choice execution until gameOver = true. Once gameOver = true, print correct end game screen.	P
Game::travel()	Don’t allow travel if nothing there, change currentRoom if passed location is visible and accessible.	P
Game::refresh()	Update gameOver, sroom visibility/accessibility correctly dependent on player health and sanity ints and unlockedRoom and closedGate booleans.	P

Utility Functions

Function	Expected Outcome	Pass/No Pass (P/NP)
calibrateScreen()	Print calibration graphic and wait for enter.	P
printGraphic()	Print graphic matching the passed string, with a 200 line buffer to clear the previous output.	P
getPadding()	Returns the properly calculated amount of padding to fit around the passed number of lines with room for a the passed graphic name.	P
getNumLines()	Returns the correct number of lines of the graphic file name passed to it.	P
validateInt()	Returns the input int if in range of passed min and max values, or returns -1 if outside of these values.	P

Game Testing:

Process	Expected Outcome	Pass/No Pass (P/NP)
Intro sequence	Display all graphics and blurbs, wait for user to continue.	P
Out of order explore attempts	Don't move forward with special explore logic paths. (ScienceRoom, Library, Gate all have item-dependent logic paths).	P
Monster fight logic	Print correct graphics/blurbs, remove correct health, add monster corpse if correct choice.	P
Dissection logic	Print correct graphics/blurbs, wait for user to proceed.	P
Combination lock logic	Print correct graphics, blurbs, change room to unlocked if correct combination is entered, otherwise allow user to explore again to try again.	P
Finale sequence	Display all graphics and blurbs, wait for user to continue, then return to main menu.	P
Game ends immediately if all sanity lost or all health lost	No more output, should immediately print game end screen when control is passed back to Game::run()	P

Solving Problems:
After writing some initial code to test the graphic and text printing process, I found that I could consolidate a fair amount of rewritten code into a few reused utility functions. I the packaged these together in the util.cpp file. Here I also added a new validateInt function different from the one I used most of the quarter, as having a stripped down version with very basic validation functionality was significantly easier to use when dealing with so much output.

Initially, I ran into problems figuring out how to allow room objects aware of general game state variables. Having duplicate game state variables passed back and forth between the Game and Room objects seemed an inelegant solution. I eventually settled on changing the explore function to take a Player pointer as a parameter. This way, I could store the game variables on the Player object, and have access to these and the player inventory whether control was in the Game::run() function or the Room::explore() function. This enabled all game state functions to exist on a centralized object, with the only updating necessary being a quick check of the player object. The exception was the gameOver boolean, which I kept in the Game class.

For a while I had a Room::update() function that checked and updated game state conditions in each room. This proved underutilized however, and so the main functions were moved to the Game::refresh() function, and the update() function removed.

See README.md for cited sources I used to make some of the graphics.