3460:209 PROGRAMMING PROJECT 2 and 3 - THE TOMB OF THE BLIND DEAD

In the early days of personal computers, text-based adventure games were an extremely popular little time-waster. The purpose of this assignment is to return to those thrilling days of yesteryear and develop our own.

|  |  |
| --- | --- |
| tombs-of-the-blind-dead.jpg | In 1307 members of the Order of the Knights Templar fled persecution in France on ships laden with treasure from the Paris Temple. Among the relics rumored to be in their possession was the Holy Grail, reported dining utensil of Christ at the Last Supper. Though most of the ships found refuge in Scotland, some set sail across the Atlantic, landing in Nova Scotia before proceeding south into modern-day Massachusetts.  Once in the New World, the surviving Knights fell into pagan rites, some involving ritual sacrifice of kidnapped young girls from among the local native tribes. As a result, the tribes declared war on the Templars, killing all and destroying the Temple. In keeping with native custom, the eyes of the dead Knights were gouged out so that their souls would wander sightless for all eternity, unable to find heaven and everlasting rest. |

Recently the remains of the ancient Temple have been unearthed. Many have searched the ruins in vain for the Holy Grail, all to meet a grisly fate at the hands of the undead Templar Knights. You have now made your way to the site, to attempt a quest which all before you have failed to complete.

(Pretty good, huh? Peter Jackson is negotiating for the movie rights.)

The ruined Temple is a maze of interconnected rooms. In the game, you will travel from room to room hunting for the grail and avoiding the undead knights. If you do encounter a zombie, with the right combination of skill and luck, you might destroy it with one of your six specially designed silver bullets. If you find the grail and escape the temple, you win. But beware: if you run out of bullets and become lost in the ruins, the zombies will eventually find you and add you to their number.

# Part 1A. Data Structures and Program Frame

As can be seen from the description, this is a fairly involved project. As such, this first part involves only setting things up in memory properly so that we can *eventually* play. Download the skeleton code and data file from the Springboard content area called Project 2, and edit them according to instructions. As you implement each section of the project, be careful to match function data types to their respective prototypes.

Note at the outset that there are several game variables tracking data relevant to game play:

* currentRoom, zombieRoom, numBullets and numRooms equal the current room for the player, the current room for the zombie, the number of bullets, and the number of rooms, respectively.
* haveGrail is a Boolean value (0/1 for false/true) indicating whether or not you have the grail with you.
* roomArray[][] is a two-dimensional structure with a row for each of the MAX\_ROOMS rooms. For room *x*, roomArray[*x*][0], …, roomArray[*x*][3] are the rooms on the other side of the doors to the north, south, west and east, respectively; while roomArray[*x*][4], roomArray[*x*][5] and roomArray[*x*][6] contain Boolean values indicating whether or not the player, a zombie or the grail are in the given room (all 0 or 1), respectively.

The rest of this part of the project is adding empty functions to your code whose headers match the corresponding prototypes, then filling in code to perform the described tasks. As with any complex program, it is sometimes useful to graphically represent the interactions between the various pieces, like so:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| main() | | |  |  |  |  |  |  |  |  |
| ↑ |  | ↑ | | | | |  |  |  |  |
| reset()  (Step 1) |  | menu() (Step 4) | | | | |  |  |  |  |
|  |  | ↑ |  | ↑ |  | ↑ | | | | |
|  |  | instructions()  (Step 2) |  | printMemory()  (Step 3) |  | setup() (Step 9) | | | | |
|  |  |  |  |  |  | ↑ |  | ↑ |  | ↑ |
|  |  |  |  |  |  | readMaze()  (Step 5) |  | placeZombie()  (Step 7) |  | placeGrail()  (Step 8) |
|  |  |  |  |  |  |  |  | ↑ |  | ↑ |
|  |  |  |  |  |  |  |  | getRandomRoom() (Step 6) | | |

As we can see, main() calls the functions developed in Steps 1 and 4. (You will need to add the functions after main to your code.) The function in Step 4 uses those written as Steps 2, 3 and 9, while Step 9 uses the functions of Steps 5, 7 and 8 (both of which use Step 6). In the write-up that follows, we will encode parts that are subsequently used by later functions. The parameters have been omitted in the write up below to reduce confusion. Notice the template given has the parameters included in the prototypes for Part1A and Part1B. We are also including the names in the parameter list for those prototypes. Feel free to change or keep them. They are used as a guide and relate to those variables declared at the onset of the entire program, in main.

### Step 1. reset()

Called by main() right before calling menu() (see Step 4), this function should set all game variables to zero.

### Step 2. instructions()

This function, called from menu(), executes when the user selects **Instructions** from the main menu (see Step 4). Display the text enclosed in the blue box in the sample image below.

### Step 3. printMemory()

This function will display the current state of all data structures. For now, execute it when the user selects **Begin** from the start menu (see Step 4). The function should run through the parameters (including all MAX\_ROOMS entries of roomArray) and display their contents to the screen.

### Step 4. menu()

This function displays the menu to the user, then asks for and validates input. main() calls reset() (Step 1) followed by this from within an infinite loop. If **Instructions** is selected call the function from Step 2. If **Begin** is selected call the function from Step 3 (for now). If **Exit** is selected quit the program.

After compiling and running the output should resemble the image on the next page.

# Step 1B. Initialization.

With the overall program structure in place we now begin game setup. Notice we have a void setup() prototype in your code. We will implement this function according to the design indicated earlier.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Step 5. readMaze() In this function open the file easyMaze.txt and read the numbers (all integers) contained there (you may want to download and open with wordpad). The first is the value for numRooms. Next are the four door values for each of the rooms, to be stored into roomArray[][]. (Note that the rooms are numbered from 1 up to and including numRooms; adjust loop bounds accordingly.) After compiling, running and selecting **Begin** from the main menu, you should see door values for each room which match the diagram at right. You should be able to trace the route from room 1 (the outside world) through all the other rooms and back to 1 is correct.  Once complete, re-edit the printMemory() function (Step 3) so that the loop through the rooms only goes from 1 up to and including numRooms and not from 0 to MAX\_ROOMS. |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | 10 | | | 11 | | |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | 6 | | | 7 | | | 8 | | |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  | 5 | | | 9 | | |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  | 3 | | | 4 | | |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  | 1 |  |  |  |  | 2 | | |  |  |  |  |  | |

### Step 6. getRandomRoom()

Implement this function to return a random room (represented by an integer between 1 and the number of rooms, i.e. numRooms).

### Step 7. placeZombie()

Select a random room *x* (using the function from Step 6) whose number is larger than half the room count (in the case of easyMaze.txt this means rooms 6 and above) and set roomArray[x][ZOMBIE\_INDEX] to 1.

|  |
| --- |
| Enter, if you dare...  \*\*\* THE TOMB OF THE BLIND DEAD \*\*\*  Main Menu  1. Instructions  2. Begin  3. Exit  Enter Menu Selection: 4  You MUST choose an option between 1 and 3.  Enter Menu Selection: 1  YOUR MISSION, SHOULD YOU CHOOSE TO ACCEPT IT, IS TO SEEK THE HOLY  GRAIL WITHIN THE RUINS OF AN ANCIENT CHURCH. TO SUCCEED YOU MUST  ENTER THE CHURCH, AVOID THE UNDEAD GUARDIANS, FIND THE GRAIL AND  ESCAPE. YOU HAVE SIX SILVER BULLETS TO PROTECT YOU. IF THE ZOMBIES  FIND YOU OR YOU RUN OUT OF BULLETS YOU WILL LOSE!  Enter, if you dare...  \*\*\* THE TOMB OF THE BLIND DEAD \*\*\*  Main Menu  1. Instructions  2. Begin  3. Exit  Enter Menu Selection: 2  Game Array:  [current][bullets][ rooms]  [ 0][ 0][ 0]  Room Array:  [ north][ south][ west][ east][player][zombie][ grail]  [R 0] ->[ 0][ 0][ 0][ 0][ 0][ 0][ 0] |
| … |
| [R28] ->[ 0][ 0][ 0][ 0][ 0][ 0][ 0]  [R29] ->[ 0][ 0][ 0][ 0][ 0][ 0][ 0]  Enter, if you dare...  \*\*\* THE TOMB OF THE BLIND DEAD \*\*\*  Main Menu  1. Instructions  2. Begin  3. Exit  Enter Menu Selection: 3 |

### Step 8. placeGrail()

Loop selecting random rooms until you find one whose number is larger than half the room count. When you find such a room *x*, set roomArray[x][GRAIL\_INDEX] to 1.

### Step 9. setup()

This function provides the initial setup for the game and should be called from menu() first thing after the user selects “Begin”. (Re-edit menu() (Step 4) accordingly.) Within this function:

* seed the random number generator with the current system time;
* set currentRoom to 1;
* set haveGrail to 0 (false);
* set numBullets to MAX\_BULLETS.

Now call the functions readMaze(), placeZombie() and placeGrail() in that order. Compile, run, and select **Begin** from the main menu. The values in the top array should reflect the changes made. You should also now see both the zombie and grail in one of the rooms.

**And with that, Part I is complete! Go** get some rest and get ready for Part II.

# Part II. Functions Used In Game Play

With the game board in place it’s time to think about actually playing a game. To setup() (Step 9) we will add one final function call to running() (Step 24, the last step) (as well as an appropriate prototype to the overall program). This function will call another, checkRoom() (Step 23, next-to-last), repeatedly until the current room becomes -1 at the end of the game. This function in turn relies on a lot of things we will now develop. Structurally it looks more-or-less like this:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| main() | ← | menu()  (Step 4) | ← | setup()  (Step 9) | ← | running()  (Step 24) |  |  |
|  |  |  |  |  |  | ↑ |  |  |
|  |  |  |  |  | checkRoom()  (Step 23) | | |  |
|  |  |  |  |  | ↑ |  | ↑ |  |
|  |  |  | ***stuff from Part II*** | | |  | waitForMove()  (Step 22) | |
|  |  |  |  |  |  |  | ↑ | |
|  |  |  |  |  |  |  | ***stuff from Part III*** | |

So even though it looks like we’re doing nothing but generating a bunch of random stuff here, all of this will prove useful at the next stage.

### Step 10. checkZombie(int)

This checks to see if the zombie is in the room specified by the parameter. If roomArray[*x*][ZOMBIE\_INDEX] equals 1 then the zombie is in room *x* and return true; otherwise he’s not and return false.

To test, insert code at the end of setup() to place the zombie in room 1 (i.e. roomArray[1][ZOMBIE\_INDEX] = 1), then execute checkZombie(1) and checkZombie(2), outputting the return values of both function calls. Remember to remove any such test code when finished testing and before moving onto the next step.

### Step 11. checkGrail(int)

Do the same as you did in Step 10 for this function using GRAIL\_INDEX. Test as before if you choose to.

### Step 12. checkNearZombie(int)

This uses checkZombie() to see if the zombie is in a room adjacent to the one specified by the parameter. For each room connected to the specified one, check to see if the zombie is there. If he is, the zombie is close and return true. Otherwise return false. As before you can temporarily insert dummy code into setup() to place the zombie in a room and check the execution of this function.

### Step 13. checkNearGrail(int)

Do the same as you did in Step 12 for this function. Test appropriately.

|  |  |
| --- | --- |
| Step 14. winOrLose(int) Implement a function to show the user they have won or lost the game (see right). There are only two options:   * 0 (win): you have arrived back in room 1 (the starting point/the outside world) with the grail in your possession. * 1 (Ioss): you are in the same room as the zombie and he promptly eats your brains.   In either case set the current room to -1; this will end a loop later in the code. | Enter, if you dare...  \*\*\* THE TOMB OF THE BLIND DEAD \*\*\*  Main Menu  1. Instructions  2. Begin  3. Exit  Enter Menu Selection: 2  As you walk in the room the zombie sees you and devour you whole!  YOU LOST!  ==========================================  YOU HAVE EMERGED FROM THE TEMPLE WITH THE  GRAIL!! YOU WIN!!  ===========================================  Enter, if you dare...  \*\*\* THE TOMB OF THE BLIND DEAD \*\*\*  Main Menu  1. Instructions  2. Begin  3. Exit  Enter Menu Selection: ■ |

### Step 15. showConnectedRooms()

Show the user all rooms connected to the current one. (Remember to report only valid rooms numbered 1 and above.) Test by calling from the end of setup(). Remove your test calls after.

### Step 16. isConnected(int)

Test to see if a room is connected to the current one. Return true if so, false otherwise. Test by calling using tunnel one of the current room; this should return true. Test for false using room -1. Remember to remove test calls after.

# Submission time for project 2

**And with that, Part II is complete! We will now submit Part I and Part II for credit.** On Springboard, go to the matching Assignments for the **Project 2**, and submit the program (cpp) to Springboard and upload. Make sure it is the program only and not a project (cbp) file. There are *no separate compilation files* required for this project so include your functions in the main program. You may use the name main.cpp, or any other suitable name of your choice.

Make sure you have observed the standards as outlined. Functions must have header comments that include the purpose of the function. They also must include the pre-condition and post-condition documentation as well. Refer to the Programming Rubric for details located on Springboard: Table of Contents -> main() -> { Course Orientation ->RubricProgrammingforAssigments. Also, refer the rubric for this project, Project 2, to make sure you have met all requirements.

**Projects will not be graded after 11:59 p.m. on the due date.**

# Part III. Game Play

We are now finally ready to finish the code and play a game. Basic structure:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| … | | |  | | |  |  | | | |  |  | | | |  | |  |
|  | ↑ |  |  | | |  |  | | | |  |  | | | |  | |  |
| running() (Step 24) | | |  | | |  |  | | | |  |  | | | |  | |  |
|  | ↑ |  | | | |  |  | | | |  |  | | | |  | |  |
| checkRoom() (Step 23) | | |  | | |  |  | | | |  |  | | | |  | |  |
| ↑ |  | ↑ | | | |  |  | | | |  |  | | | |  | |  |
| *stuff from Part II* |  | waitForMove() (Step 22) | | | |  |  | | | |  |  | | | |  | |  |
|  |  | ↑ | |  | ↑ |  |  | | | |  |  | | | |  | |  |
|  | validateSelection()  (Step 20) | | |  | doSelection() (Step 21) | | | | | | | | | | | | | |
|  |  |  | |  | ↑ |  | ↑ | | |  | | | ↑ | | |  | ↑ | |
|  |  |  | |  | printMemory()  (Step 3) |  | moveRoom()  (Step 17) | | |  | | | shootRoom()  (Step 18) | | |  | moveZombie()  (Step 19) | |
|  |  |  | |  |  |  | ↑ |  | ↑ |  | | | ↑ |  | ↑ |  |  | |
|  |  |  | |  | checkGrail()  (Step 12) | | |  | isConnected()  (Step 16) | | | | |  | checkZombie()  (Step 11) | | | |

On with the code… **please notice** you may need to pass other parameters (such as the roomArray or gameArray,etc.) in addition to what has been outlined for these functions.

### Step 17. moveRoom(int)

This moves the user to the room given as the parameter. Steps involved:

* Test to see if the given room is connected to the current one.
  + If so, set the current room player index to 0, the given room player index to 1, and the current room to the parameter.
  + If not, let the user know they cannot move there.
* If the player finds the grail in the new room, set haveGrail true.
* If the player already had the Holy Grail with him, move the grail from the old to the new room.
* If the player had the grail with him and has now entered room 1, declare a win and end the game.

### Step 18. shootRoom(int)

This shoots a bullet into the room given as the parameter. Test to see if the given room is connected to the current one. If so, decrement the bullet count and check to see if the zombie was in that room. If he was he’s dead now and remove him from the game (i.e. zero out the ZOMBIE\_INDEX column of roomArray and zombieRoom). Otherwise tell the player that they missed.

### Step 19. moveZombie()

If the zombie is “alive” (i.e. not in room zero) choose one of the four doors of his current room. If that door leads to a non-zero-numbered room move the zombie there.

### Step 20. validateSelection(string)

This function takes a string and makes sure the first character is "Q" "D" "S" or "M". Return true if it is, false if not. Test accordingly. Later this function will be used for game control.

### Step 21. doSelection(string)

This function allows the user to control the game. There are four commands:

* Q - quit the game
* D - print memory
* S # - shoot into the indicated room, and
* M # - move into the indicated room.

Depending on the first character of the parameter, call the appropriate function. If the choice is either “shoot” or “move” move the zombie after completing the player’s instruction. Test accordingly. HINT: use atoi(string) and substr(int) to get the room number of calls that need it.

### Step 22. waitForMove()

Show a command menu to the user. Wait for input, then validate the input they give (with validateSelection (Step )). Loop until a valid command is given. Once a valid command is given, execute that command. Test accordingly.

|  |  |
| --- | --- |
| Step 23. checkRoom() This is a large complex function. Implement the pseudocode at right. | * Display the current room and number of bullets to the user. * If the player has the grail say so, otherwise do nothing. * If the zombie is in the room lose game, otherwise do nothing. * If the zombie is nearby, player can hear it;   if the grail is nearby, player can sense it;  otherwise do nothing.   * Show rooms connected to this (using showConnectedRooms()). * Show the menu and wait on user input (with waitForMove()). |

### Step 24. running()

Basic function that runs the game. Inside a while loop that ends when the current room is -1 call checkRoom().

Edit setup() so that it calls running() as its final instruction and that’s it. **Congratulations! The game is complete!**

**Program and Submission Details**

1. Always rigorously test your code and manually check your results to see if they are correct.
2. Refer to the Programming Rubric for details located on Springboard: Table of Contents -> main() -> { Course Orientation ->RubricProgrammingforAssignments
   1. Use a header comment with your name and the description of the program.
   2. Use pre and post condition comments for each function.
   3. Use prototypes for the functions (pre & post comments go with them).
   4. Include all your functions at the end of the program.
   5. Use appropriate code comments.
   6. Use appropriate variable names.
   7. Use constants where appropriate.
   8. Use good coding practices (i.e., spacing, indentation, etc.).

# Submission time for project 3

**We will now submit Part III for credit.** On Springboard, go to the matching Assignments for the **Project 3**, and submit the program (cpp) to Springboard and upload. Make sure it is the program only and not a project (cbp) file. There are *no separate compilation files* required for this project so include your functions in the main program. You may use the name main.cpp, or any other suitable name of your choice.

**Projects will not be graded after 11:59 p.m. on the due date.**

*Last Updated 6.26.2016. Based on an assignment by A. Deeter adapted from “Apple Pascal Games” by Hergert and Kalash, 1984.*

*Be aware that programming falls under all of the rules of plagiarism. Be careful when using any coding found in the outside world that is not your own. Any evidence of plagiarism is subject to sanctions like forfeits, suspension, and even ejection, as determined by the Department of Student Conduct and Community Standards.*