Alexander Densmore

CS 162-400

Final Project Documentation

Program Description:

This program implements a game with linked spaces that is set in Ancient Rome. The board is a 3 x 3 grid of 9 spaces, each space being of a different type. The user takes on the role of a time traveler who has gone to Ancient Rome to research Roman music (although it is known that Roman music existed, little is known about what it would have actually sounded like). The user must visit the emperor Nero for dinner, record him singing, and get a copy of written music from him. However, there are challenges the user must overcome before they will be given permission to visit Nero. They have 75 total steps to go between the spaces, complete the tasks, and collect the items necessary to get an audience with Nero. The user must be careful, because certain actions (such as fighting gladiator battles) could result in their character's death and the end of the game regardless of the number of steps taken so far.

Program Files:

Header and Source Files:

- Space.hpp / Space.cpp
    - Bibliotheca.hpp / Bibliotheca.cpp
    - DomusAurea.hpp / DocmusAurea.cpp
    - Ludus.hpp / Ludus.cpp
    - CircusMaximus.hpp / CircusMaximus.cpp
    - Forum.hpp / Forum.cpp
    - Theatrum.hpp / Theatrum.cpp
    - Colosseum.hpp / Colosseum.cpp
    - Thermae.hpp / Thermae.cpp
    - CampusMartius.hpp / CampusMartius.cpp
- Board.hpp / Board.cpp
- Game.hpp / Game.cpp
- enterValidInt.hpp / enterValidInt.cpp
- getRandomInt.hpp / getRandomInt.cpp
- menu.hpp / menu.cpp
- pressEnter.hpp / pressEnter.cpp
- finalProjMain.cpp

Text Files:

- Game_Instructions.txt
- Board_Images.txt
- Bibliotheca_Description.txt
- DomusAurea_Description.txt
- Ludus_Description.txt
- Ludus_Questions.txt
- CircusMaximus_Description.txt
- Forum_Description.txt
- Theatrum_Description.txt
- Colosseum_Description.txt
- Thermae_Description.txt
- Thermae_Narration.txt
- CampusMartius_Description.txt
- Ludus_Questions.txt
- Ending_Sequence.txt

Initial Ideas:

Theme: Ancient Rome

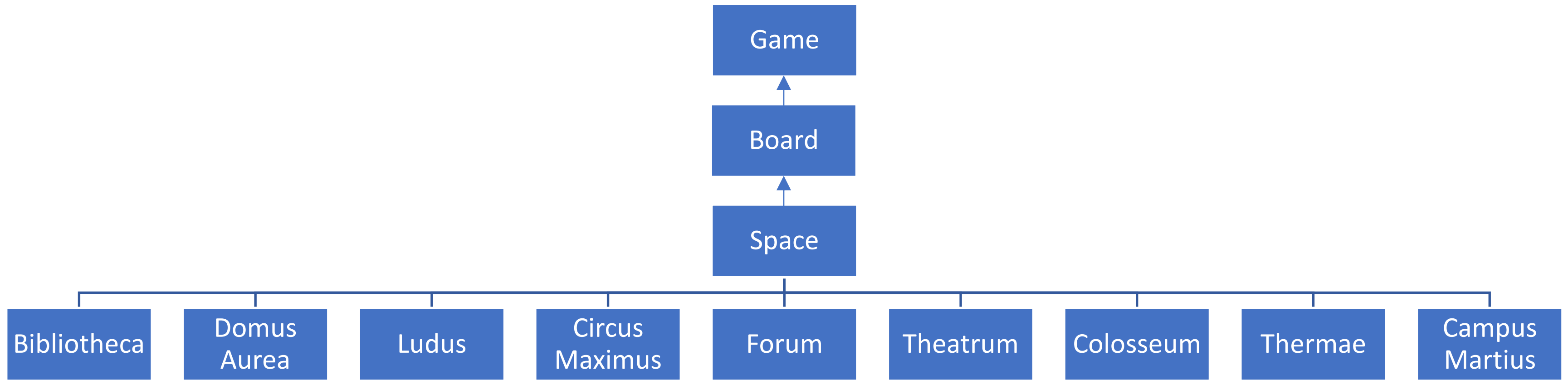
Different Types of Spaces:

1. Bibliotheca (Can do favor for librarian of retrieving missing book from schoolteacher to earn some money; go there to get musical score after seeing Nero's performance).
    - 5 coins for delivering scroll
2. Domus Aurea (have dinner with Nero, who then invites you to see his performance in the theater. Go in chariot with him to library (where he picks up with music he will be performing and) and to the theater. Nero signs and gives you the music after the performance).
3. Ludus (school; participate in trivia against schoolchildren. Each game consists of 5 questions with user and 1 computer players. There are 5 games total that can be played. Children get random score of 0-5. Tiebreaker questions at end of an individual game where computer contenders either get 0 or 1 point for each question until a definitive winner is determined or 5 tiebreaker questions have been asked).
    - 4 coins for winning game; 2 coins if tie – ties after 5 tiebreaking questions asked
    - Just playing against 1 student each game (5 students total, each with a specific name)
        - Cornelius, Marcus, Julius, Philemon, Quintus
    - If less than 5 rounds played, schoolmaster says "come back later to play again."
    - if 5 rounds played, schoolmaster says "my students are dismissing for the day"
4. Circus Maximus (can bet on chariot races)
    - can bet 2, 4, or 6 coins (only allows a given bet amount if player has that amount)
    - 3 chariot teams; pick which one will win
    - Statement to come back to bet again
5. Forum (one merchant from whom to purchase items)
    - expensive (14 coins)

        - olive oil
        - sandals
        - wine
    - cheap (8 coins)
        - grain
        - salt
        - tunic
    - if satchel is full, statement telling them to go to the theater (which is taking collections for a local orphanage)

6. Theatrum (See Nero perform)
    - ability to donate items to neighboring orphanage
    - *cannot* donate key items (scroll or permit)
7. Colosseum (participate in gladiatorial fights; can earn a lot of money, but chance of dying; sword-net-shield fighting system, like rock-paper-scissors).
    - 10 coins if they win
    - Statement to come back to play again
8. Thermae (must pay to get a bath and then take shortest route possible to Domus Aurea to stay fresh and clean)
    - 2 coins to bathe
9. Campus Martius (get scroll from soldier giving permission to see Nero)
    - same 3 questions with 3 choices
        - of what country are you a citizen?
        - who is the best emperor?
        - what was Nero's role in the great fire?
    - game randomly selects 1 cheap and 1 expensive item that soldier wants. Fills two vectors in constructor, one with expensive items and the other with cheap items. Randomly selects 1 item of each type, setting private data members of class accordingly.

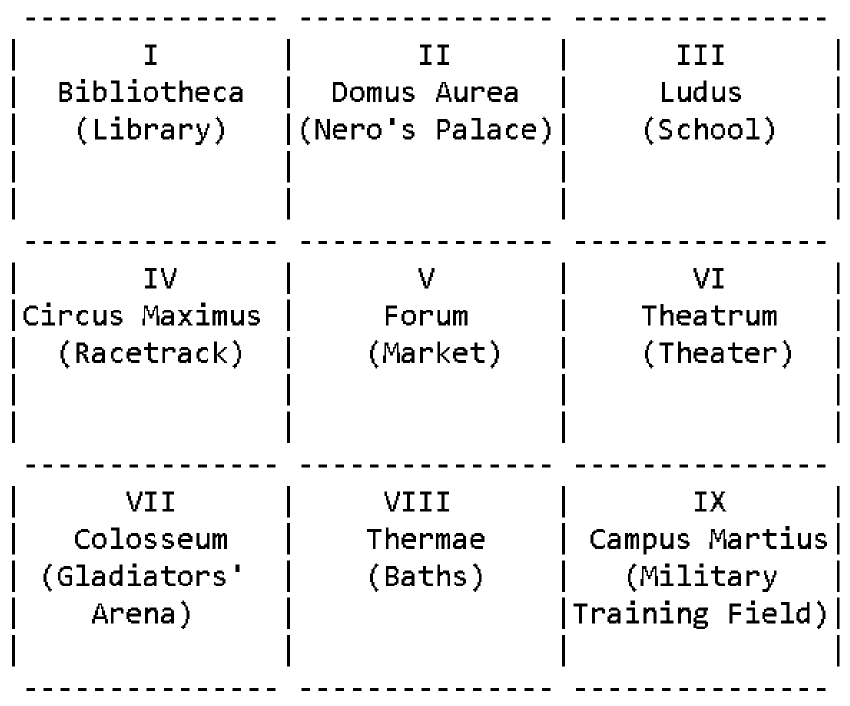
Class Hierarchy Diagram:



Game Board Layout (Simplified Map of Rome - Linked Spaces)

*Note: Each space will have 8 pointers: north, northeast, east, southeast, south, southwest, west, northwest

I have created the board in a .txt file so I know what characters to print to form each space and the text within the spaces. Below is a screenshot of that board (formatting does not remain as intended when copying and pasting into Word).



```
 --------------- --------------- ---------------
|       I       |      II       |      III      |
|  Bibliotheca  |  Domus Aurea  |     Ludus     |
|   (Library)   |(Nero's Palace)|   (School)    |
|               |               |               |
|               |               |               |
 --------------- --------------- ---------------
|      IV       |       V       |      VI       |
|Circus Maximus |     Forum     |   Theatrum    |
|  (Racetrack)  |   (Market)    |   (Theater)   |
|               |               |               |
|               |               |               |
 --------------- --------------- ---------------
|      VII      |     VIII      |      IX       |
|   Colosseum   |    Thermae    | Campus Martius|
| (Gladiators'  |    (Baths)    |   (Military   |
|    Arena)     |               |Training Field)|
|               |               |               |
 --------------- --------------- ---------------
```

The player's position will be marked with an asterisk centered in the bottom row of a space. The player must move to one of the squares their current square points to. For instance, a player at the Thermae can go to the Colosseum, Circus Maximus, Forum, Theatrum, or Campus Martius. However, they *cannot* go directly to the Bibliotheca, Domus Aurea, or Ludus.

The player will start in the Forum.

Pseudocode:

Space.hpp / Space.cpp

Constants

- const char DELIM = ‘#’ // used with getline function for file input
- const int SATCHEL_CAPACITY = 3
- const string SCROLL = “scroll” // string constants used for satchel
- const string PERMIT = “permit”
- const string OLIVE_OIL = “olive oil”
- const string SANDALS = “sandals”
- const string WINE = “wine”
- const string GRAIN = “grain”
- const string SALT = “salt”
- const string TUNIC = “tunic”

Protected Data Members (inherited by child classes)

- Space* north
- Space* northeast
- Space* east
- Space* southeast
- Space* south
- Space* southwest
- Space* west
- Space* northwest
- string name
- int num // indicates space number on the board
- string description

Public Member Functions:

- Space(string name, int num, string inputFileName)
    - set all pointers to nullptr
    - this->name = name;
    - this->num = num;
    - // load space description from file
    - ifstream inputFile(inputFileName)
    - getline(inputFileName, description, DELIM)
    - inputFile.close()
- virtual ~Space()
- string get_name() const
- int get_num() const
- string get_description() const
- Space* get_north() const

- Space* get_northeast() const
- Space* get_east() const
- Space* get_southeast() const
- Space* get_south() const
- Space* get_southwest() const
- Space* get_west() const
- Space* get_northwest() const
- void set_north(Space* spacePtr)
- void set_northeast(Space* spacePtr)
- void set_east(Space* spacePtr)
- void set_southeast(Space* spacePtr)
- void set_south(Space* spacePtr)
- void set_southwest(Space* spacePtr)
- void set_west(Space* spacePtr)
- void set_northwest(Space* spacePtr)
- virtual void interact(set<string>& satchel, int& money, int& stepsSinceBathing, bool& hasBathed, bool& knowsAboutScroll, bool& stillAlive, bool& withNero) = 0;

Bibliotheca.hpp / Bibliotheca.cpp

Private data member:

- bool returnedScroll

Member functions:

Bibliotheca() : Space("Bibliotheca", 1, "Bibliotheca_Description.txt")

- returnedScroll = false

virtual ~Bibliotheca()

virtual void interact(set<string>& satchel, int& money, int& stepsSinceBathing, bool& hasBathed, bool& knowsAboutScroll, bool& stillAlive, bool& withNero) override

- if (knowsAboutScroll == false)
  - Print: "Hey, could you do me a favor? That schoolteacher over at the Ludus has not returned the scroll he borrowed from here. Can you go get the scroll for me? I will give you a reward if you do so.
  - knowsAboutScroll = true
- else if (returnedScroll == false && satchel.find(SCROLL) == satchel.end()) // if player does not have scroll
  - Print: Have you gotten that scroll from the teacher yet? Please go and get it from him as soon as you can. I want to close up for the day, but I will wait until you get that scroll back.
- else if (returnedScroll == false && satchel.find(SCROLL) != satchel.end()) // if player has scroll
  - Print: Thank you for bringing that scroll back from that teacher! He always keeps items checked out way too long.

    - Print: Here are 5 coins for your effort.
    - money += 5
    - satchel.erase(SCROLL)
    - returnedScroll = true
    - Print: The library is now closed for the day. Have a great day!
- else
    - Print: The library has closed for the day. Only those accompanied by the emperor can enter the library when it is closed.

DomusAurea.hpp / DomusAurea.cpp

Member Functions

DomusAurea() : Space("Domus Aurea", 2, "DomusAurea_Description.txt")

virtual ~DomusAurea()

virtual void interact(set<string>& satchel, int& money, int& stepsSinceBathing, bool& hasBathed, bool& knowsAboutScroll, bool& stillAlive, bool& withNero) override

- if (satchel.find(PERMIT) == satchel.end())   // if user does not have a permit to see Nero
    - Print: Halt! You must have a permit to see the emperor, which you can only get from his Praetorian Prefect. The prefect is currently at the Campus Martius training. Only come back if you have a permit!
- else if (!hasBathed)
    - Print: You smell like you have never bathed in your life! Maybe you're from out of town and have never used baths with the refinement of Rome's Thermae. Although you have a permit, I can't let you in smelling like this! Only come back when you have bathed.
- else if (stepsSinceBathing > 2)
    - Print: You stink! You say you already bathed today? I don't care! You must have been roaming around Rome too long since your bath. Although you have a permit, there's no way I'm letting you in smelling like this! Go get a bath, and come straight back here without making any unnecessary stops along the way.
- else
    - Print: I see that you have a permit and are freshly bathed. Welcome to the Domus Aurea!
    - withNero = true   // will signal to calling function to trigger ending sequence – the user has now won the game.

Ludus.hpp / Ludus.cpp

Constants:

- const int NUM_QUESTIONS = 50
- const int NUM_COICES = 4
- const int NUM_GAMES = 5

Private Data Members:

- struct Question
    - string questionText
    - vector<string> answerChoices(NUM_CHOICES)
    - int answerNum
- vector<Question> questions(NUM_QUESTIONS)
- vector<string> studentNames(NUM_GAMES)
- bool obtainedScroll
- int gamesPlayed
- int questionsAsked

Member Functions:

Ludus() : Space(“Ludus”, 3, “Ludus_Description.txt”)

- // Read questions from file into vector and randomly shuffle questions
- ifstream inputFile(“Ludus_Questions.txt”)
- for (int questionNum = 0; questionNum < NUM_QUESTIONS, questionNum++)
    - Question q
    - string text
    - getline(inputFile, text)
    - q.questionText = text
    - vector<string> choices(NUM_CHOICES)
    - for (int choiceNum = 0; choiceNum < NUM_CHOICES; choiceNum++)
        - getline(inputFile, text)
        - choices.push_back(text)
    - q.answerChoices = choices
    - getline(inputFile, text)
    - q.answerNum = stoi(text)
    - questions.push_back(q)
- inputFile.close()
- random_shuffle(questions.begin(), questions.end())
- studentNames.push_back(“Cornelius”)
- studentNames.push_back(“Marcus”)
- studentNames.push_back(“Julius”)
- studentNames.push_back(“Philemon”)
- studentNames.push_back(“Quintus”)
- obtainedScroll = false
- gamesPlayed = 0
- questionsAsked = 0

virtual ~Ludus()

virtual void interact(set<string>& satchel, int& money, int& stepsSinceBathing, bool& hasBathed, bool& knowsAboutScroll, bool& stillAlive, bool& withNero) override

- Print: Welcome to my School!

```
• // Check to see if the player knows about the scroll. If so, the magister (teacher) tries to give it to
  them.
• if (obtainedScroll == false && knowsAboutScroll == true)
    o Print: Magister (Teacher): Here is that scroll about which that grouchy librarian has been
      pestering me. Thanks for taking it back for me!
    o // Make sure satchel can hold the scroll
    o  if (satchel.size() < SATCHEL_CAPACITY)
        ▪ satchel.insert(SCROLL)
        ▪ obtainedScroll = true
    o else
        ▪ Print: “I see you do not have room in your satchel for the scroll.”
        ▪ Print: “The theater is collecting unneeded items as donations for the poor.
        ▪ I suggest you go there and then come back if you want that scroll.”
• if (gamesPlayed < NUM_GAMES)
    o Ask if player would like to play trivia against one of the students using menu function.
    o if (choice == 1)
        ▪ money += play_trivia()
• else
    o Print: Since my students have gone home for the day, there is no more trivia to play.
    o Print: Have a great day!
    o Have user press enter
```

private int play_trivia()

```
• print that 5 questions will be asked, and whoever gets the most right wins. 1 point will be
  awarded for correct answers. There is no penalty for incorrect answers. If the game is tied after
  5 questions, then tie breaker questions will be asked until a definitive winner is chosen or 5 tie-
  breaker questions have been asked (whichever occurs first).
• int userScore = 0;
• int computerScore = 0;
• string computerName = students[gamesPlayed];
• int round
• for (round = 1; round <= 5; round++)
    o clear screen
    o Print Round Number
    o Question q = questions[questionsAsked]
    o Print q.questionText
    o int answerNum = q.answerNum
    o int userAnswer = menu(q.answerChoices)
    o if (userAnswer == answerNum)
        ▪ userScore++
        ▪ Print: Correct!
    o else
        ▪ Print: Incorrect
        ▪ Print: The correct answer was: q.answerChoices[(answerNum-1)]
    o computerScore += getRandomInt(0, 1)
    o questionsAsked++
    o display user and computer scores
    o have user press enter to continue
• // Use a while loop for a tie breaker
• while (userScore == computerScore && round <= 10)
    o Print: After (round) questions, it is a tie game.
    o clear screen
    o Print Round Number
    o Question q = questions[questionsAsked]
    o Print q.questionText
    o int answerNum = q.answerNum
    o int userAnswer = menu(q.answerChoices)
    o if (userAnswer == answerNum)
        ▪ userScore++
        ▪ Print: Correct!
    o else
        ▪ Print: Incorrect
        ▪ Print: The correct answer was: q.answerChoices[(answerNum-1)]
    o computerScore += getRandomInt(0, 1)
    o questionsAsked++
    o round++
    o display user and computer scores
    o have user press enter to continue
• gamesPlayed++
• int moneyWon = 0;
• if (userScore > computerScore)
    o Print: Congratulations! You have beaten (computerName)! You receive 4 coins.
    o moneyWon = 4
• else if (userScore < computerScore)
    o Print: Unfortunately, you have lost this game of trivia and have not earned any money.
• else
    o Print: Since the game has ended in a tie (even after 5 tie-breaking rounds), you have
      earned 2 coins
    o moneyWon = 2
• if (gamesPlayed < NUM_GAMES)
    o Print: Please come back here again if you want to play more trivia!
• else
    o Print: My students need to go home for the day, so that’s it for trivia. Thank you for
      playing!
• have user press enter
• return moneyWon
```

CircusMaximus.hpp / CircusMaximus.cpp

Enum Class

- enum class Color{RED, GREEN, BLUE}; // chariot team colors

Constants:

- const int LOW_BET = 2
- const int MEDIUM_BET = 4
- const int HIGH_BET = 6

Member Functions

CircusMaximus() : Space("Circus Maximus", 4, "CircusMaximus_Description.txt")

virtual ~CircusMaximus()

virtual void interact(set<string>& satchel, int& money, int& stepsSinceBathing, bool& hasBathed, bool& knowsAboutScroll, bool& stillAlive, bool& withNero) override

```
• (space description informs user that they will be betting if they enter the Circus Maximus)
• if (money < LOW_BET)
    o Print: "I'm sorry, but you must have at least LOW_BET coins to bet on the chariot races.
      Please come back again!
• else
    o int bet
    o bool validBet = false
    o do
        ▪ What would you like to bet?
        ▪ (menu with choices bet 2 coins, bet 4 coins, bet 6 coins)
        ▪ if (betChoice == 1)        // previous conditional already verified that player has
                                        at least 2 coins
            • bet = LOW_BET
            • validBet = true
        ▪ else if (betChoice == 2)
            • if (money >= MEDIUM_BET
                o bet = MED_BET
                o validBet = true
            • else
                o Print: "You don't have enough money for that bet. Please bet a
                  lower amount
        ▪ else if (betChoice == 3)
            • (same process as above)
    o … while(!validBet)
    o money += race(bet)
    o Have user press enter to continue
```

private int race(int bet)

```
• Print: On what team would you like to bet?
• int menuChoice = (menu with team choices in enum order: RED, GREEN, BLUE)
• // decrement menuChoice and static_cast to Color
• menuChoice--
• Color betColor = static_cast<Color>(menuChoice)
• Color winnerColor = static_cast<Color>(getRandomInt(0, 2))
• if (winner == Color::RED)
    o Print: Red wins!
• else if (same process for other 2 colors)
• int winnings = 0
• if (betColor == winningColor)
    o winnings = bet * 2
    o Print: Congratulations! Since you picked the winning team, you have earned (winnings)
      coins.
• else
    o Print: I’m sorry, but you did not pick the winning team, so you have lost your bet.
• Print: Please come back and play again!
• return winnings
```

Forum.hpp / Forum.cpp

Constants:

- const int EXPENSIVE_PRICE = 14
- const int CHEAP_PRICE = 8

Private Data Members

- struct Item
    - string itemName;
    - int price;
    - Item(string itemName, int price)
        - this->itemName = itemName
        - this->price = price
- vector<Item> goodsForSale(6)
- vector<string> purchaseMenu(7)

Member Functions

Forum() : Space("Forum", 5, "Forum_Description.txt")

- Item good1(OLIVE_OIL, EXPENSIVE_PRICE)
- Item good2(SANDALS, EXPENSIVE_PRICE)
- Item good3(WINE, EXPENSIVE_PRICE)
- Item good4(GRAIN, CHEAP_PRICE)
- Item good5(SALT, CHEAP_PRICE)
- Item good6(TUNIC, CHEAP_PRICE)
- goodsForSale.push_back(good1)
- (repeat for other 5 goods)
- make_purchase_menu()

private void make_purchase_menu()

- for (int index = 0; index < 6; index++)
    - string menuChoice = goodsForSale[index].itemName
    - menuChoice += " (" + std::to_string(goodsForSale[index].price) + ")"
    - purchaseMenu.push_back(menuChoice)
- purchaseMenu.push_back("Leave the Forum")

virtual ~Forum()

virtual void interact(set<string>& satchel, int& money, int& stepsSinceBathing, bool& hasBathed, bool& knowsAboutScroll, bool& stillAlive, bool& withNero) override

- Print: "Hello! What would you like to buy?
- int purchaseChoiceNum = 0
- do
    - purchaseChoiceNum = menu(purchaseMenu)
    - if (purchaseChoiceNum != 6)
        - int itemNum = (purchaseChoiceNum-1)
        - string itemName = goodsForSale[itemNum].itemName

```
        int price = goodsForSale[itemNum].price
        if (satchel.size() == SATCHEL_CAPACITY)
            Print: I’m sorry, but your satchel is at max capacity.
            Print: Please go to the theater. They are taking collections for the needy there. Once you have gotten rid of at least 1 item in your satchel, please come back here.
        else if (satchel.find(itemName) != satchel.end)
            Print: You already have this item. Please come back when you need more of this item, or choose a different item.
        else if (money < price)
            Print: You do not have enough money to purchase this item. Please choose a different item or come back later.
        else
            Print: Here is your (itemName)!
            satchel.insert(itemName)
            money -= price
… while (purchaseChoiceNum != 6)
Print: Have a great day!
```

Theatrum.hpp / Theatrum.cpp

Member Functions

Theatrum() : Space(“Theatrum”, 6, “Theatrum_Description.txt”)

virtual ~Theatrum()

virtual void interact(set<string>& satchel, int& money, int& stepsSinceBathing, bool& hasBathed, bool& knowsAboutScroll, bool& stillAlive, bool& withNero) override

```
if (satchel.empty())
    Print: You do not have any items to donate at this time
else
    bool wantsToExit = false;
    do
        vector<string> donationMenu
        set<string>::iterator satchelIter
        for (satchelIter = satchel.begin(); satchelIter != satchel.end(); satchelIter++)
            donationMenu.push_back(*satchelIter)
        donationMenu.push_back(“Leave the Theatrum”);
        int donationNumber = menu(donationMenu)
        if (donationNumber == donationMenu.size())
            wantsToExit = true
        else if (donationMenu[(donationNumber – 1)] == SCROLL || donationMenu[(donationNumber – 1)] == PERMIT)
            Print: I’m sorry, but we cannot accept that item for donations
        else
            string donationName = donationMenu[(donationNumber-1)]
```

        - satchel.erase(donationName)
        - Print: Thank you very much!
        - if (satchel.empty())
            - Print: It looks like you don’t have any items left in your satchel. Please come back when you have more that you want to donate!
    - … while (satchel.empty() == false && wantsToExit == false);
- Have user press enter

Colosseum.hpp / Colosseum.cpp

Enum Class

- enum class Move{SHIELD, NET, SWORD}

Private Data Member

- vector<string> moveMenu(3)

Member Functions

Colosseum() : Space(“Colosseum”, 7, “Colosseum_Description.txt”)

- moveMenu.push_back(“Shield”)
- moveMenu.push_back(“Net”)
- moveMenu.push_back(“Sword”)

virtual ~Colosseum()

virtual void interact(set<string>& satchel, int& money, int& stepsSinceBathing, bool& hasBathed, bool& knowsAboutScroll, bool& stillAlive, bool& withNero) override

- * make sure description file warns player that they may die in the battle and reminds them to review the rules before entering the Colosseum
- Print: The gladiator battle is about to begin. Are you ready?
- Have user press enter
- int userStrengthPoints = 5
- int computerStrengthPoints = 5
- while (userStrengthPoints > 0 && computerStrengthPoints > 0)
    - Clear screen
    - Print each player’s strengthPoints
    - Print: Select your move
    - int userMoveNum = menu(moveMenu)
    - userMoveNum—
    - Move userMove = static_cast<Move>(userMoveNum)
    - int computerMoveNum = getRandomInt(0, 2)
    - Print: Your opponent’s move: (moveMenu[computerMoveNum])
    - Move computerMove = static_cast<Move>(computerMoveNum)
    - if (userMove == Move::SHIELD)
        - if (computerMove == Move::SWORD)

```
            • Print: You win this round!
            • computerStrengthPoints—
        ▪ else if (computerMove == Move::NET)
            • Print: Your opponent wins this round.
            • userStrengthPoints—
        ▪ else
            • Print: This round is a draw.
    o else if (userMove == Move::NET)
        ▪ (repeat same process for other 2 user move possibilities)
    o Print updated scores
    o if (computerStrengthPoints == 0)
        ▪ Print: Congratulations, you win!
        ▪ bool computerDies = static_cast<bool>(getRandomInt(0, 1))
        ▪ if (!computerDies)
            • Print: The senator has ordered that you let your opponent live.
        ▪ else
            • Print: The senator has ordered that you kill your opponent.
        ▪ Print: Here are 10 coins for your victory.
        ▪ money += 10
    o if (userStrengthPoints == 0)
        ▪ Print: You have lost this match.
        ▪ bool userDies = static_cast<bool>(getRandomInt(0, 1))
        ▪ if (!userDies)
            • The senator has ordered your opponent to let you live.
            • You haven't won any money, but you leave with your life.
        ▪ else
            • The senator has ordered your opponent to kill you.
            • Thank you for your sacrifice for the entertainment of the Roman People.
            • stillAlive = false
        ▪ if (stillAlive)
            • Print: Please come back and play again!
• (end while)
• Have user press enter
```

Thermae.hpp / Thermae.cpp

Constants

- const int BATH_COST = 2

Member Functions

Thermae() : Space(“Thermae”, 8, “Thermae_Description.txt”)

virtual ~Thermae()

virtual void interact(set<string>& satchel, int& money, int& stepsSinceBathing, bool& hasBathed, bool& knowsAboutScroll, bool& stillAlive, bool& withNero) override

- Print: The cost of a bath is BATH_COST
- if (money < BATH_COST)
    - Print: You do not have enough money to bathe at this time. Please come back again when you have more money.
- else
    - (menu to ask user if they want a bath)
    - if (bathChoice == 1)
        - money -= BATH_COST
        - if (!hasBathed)
            - ifstream inputFile(“Thermae_Narration.txt”)
            - string text
            - while (getline(inputFile, text, DELIM))
                - Print: (text)
            - hasBathed = true
        - else
            - Print: Since you have already bathed today, you know how the routine goes. Now you are nice and clean again!
        - turnsSinceBathing = 0
    - else
        - Print: Come back later if you change your mind!

CampusMartius.hpp / CampusMartius.cpp

Private Data Members

- bool hasPassedTest
- bool hasObtainedPermit
- string expensiveChoice
- string cheapChoice
- bool hasGivenExpensive
- bool hasGivenCheap

Member Functions

CampusMartius() : Space("CampusMartius", 9, "Campus_Martius.txt")

- hasPassedTest = false
- hasObtainedPermit = false
- vector<string> expensiveItems = {OLIVE_OIL, SANDALS, WINE}
- vector<string> cheapItems = {GRAIN, SALT, TUNIC}
- expesniveChoice = expensiveItems[getRandomInt(0, 2)]
- cheapChoice = cheapItems[getRandomInt(0, 2)]
- hasGivenExpensive = false
- hasGivenCheap = false

virtual ~CampusMartius()

virtual void interact(set<string>& satchel, int& money, int& stepsSinceBathing, bool& hasBathed, bool& knowsAboutScroll, bool& stillAlive, bool& withNero) override

- if (!hasPassedTest)
    - stillAlive = test_user()
- else if(!hasObtainedPermit)
    - give_items(satchel)
- else
    - Print: "I've already given you permission to see Nero. What are you waiting for? Get to the Domus Aurea at once!"

private bool test_user()

- Print: Halt! Who goes there?
- You say you want to see Nero?
- Well, I, Gaius Silius, am the Prefect of the Praetorian Guard.
- Emperor Nero is very busy, so in order to see him, you'll need my permission.
- I will ask you 3 questions to see if you are worthy of seeing the emperor.
- Print: Of what country are you a citizen?
- vector<string> question1Choices = {"The United States", "Rome", "Germany"}
- int answerChoice = menu(question1Choices)
- if (answerChoice == 2)
    - Print: OK, so you are a Roman. Good thing, because I do not trust non-Romans.
- else
    - Print: I don't trust non-Romans, especially those who want an audience with the emperor.
    - Print: I must kill you now for the protection of the emperor.
    - return false
- Print: Who is the best emperor?
- vector<string> question2Choices = {"Nero", "Caligula", "Augustus"}
- answerChoice = menu(question2Choices)
- if (answerChoice == 1)
    - Print: I agree! Nero is the best emperor! Long live Nero!
- else

```
        This is treason! How dare you be such a traitor to your country? Clearly, Nero is the best emperor ever.
        You must be plotting to overthrow him … I know some scoundrles have talked of such plans, and I am to immediately execute anyone on the spot who even hints at conspiracy.
        Time to die, traitor!
        return false
    Print: What was Nero's role in the Great Fire?
    vector<string> question3Choices = {"He started it", "He played the lyre while Rome burned," "He made improvements to Rome after the Great Fire to prevent future fires."}
    answerChoice = menu(question3Choices)
    if (answerChoice == 3)
        Print: I am glad you acknowledge just how much our great emperor has done to protect Rome from future fires! I can't believe those traitors who claim he started the fire or played his lyre while Rome burned.
        Print: Clearly, you are a supporter of Nero who is worthy to see the emperor.
        Print: I do have a favor to ask before I give you permission to visit Nero, though.
        Print: Could you pick some items up for me in the Forum?
        Print: I am on duty the rest of the day but could use the following items:
        Print: "\t- " << expensiveItem
        Print: "\t- " << cheapItem
        Print: Return here with those, and I will be happy to give you a permit to see Nero.
        hasPassedTest = true
        return true
    else
        How dare you insult our dear emperor like that! All honest, trustworthy Roman citizens know that Nero was deeply grieved by the Great Fire did everything in his power to prevent future fires.
        You may have survived the Great Fire, but you will not survive my wrath.
        For Nero!
        return false
```

private void give_items(set<string>& satchel)

```
    // Try to get whichever items are missing from user, and then respond with appropriate message when user has given items
    if (!hasGivenExpensive)
        hasGivenExpensive = check_for_item(satchel, expensiveChoice)
    if (!hasGivenCheap)
        hasGivenCheap = check_for_item(satchel, cheapChoice)
    // Now that the user has given each item if they have it, print appropriate messages depending on what the soldier still needs
    if (hasGivenExpensive == false && hasGIvenCheap == false)
        Print: I am still waiting on the (expensiveChoice) and (cheapChoice) from you. Bring them to me as soon as you have them!
```

- else if (!hasGivenExpensive)
    - Print: Although you have brought me the (cheapChoice), I am still waiting on the (expensiveChoice) from you. Bring it to me right away!
- else if (!hasGivenCheap)
    - Print: Although you have brought me the (expensiveChoice), I am still waiting on the (cheapChoice) from you. If you can afford to buy me (expensiveCHoice), you can afford to buy me (cheapChoice)! Come back as soon as you have it!
- else
    - Print: Since you have brought me the (expensiveItem) and the (cheapItem) like I asked, I will give you permission to see the Great Emperor Nero.
    - satchel.insert(PERMIT)
    - hasObtainedPermit = true

private bool check_for_item(set<string>& satchel, string itemName)

- if (satchel.empty())
    - return false
- else if (satchel.find(itemName) == satchel.end)
    - return false
- else
    - Print: you give the soldier the [itemName]
    - satchel.erase(itemName)
    - Have user press enter
    - return true

Board.hpp / Board.cpp

Constants:

- const int NUM_BOARD_IMAGES = 9 // indicates how many board images there are

Enum class:

- enum class Direction{N, NE, E, SE, S, SW, W, NW};

Private data members:

- vector<string> boardImages(NUM_BOARD_IMAGES)
- Space* space1 // these space pointers will be used to help construct and destruct the board
- Space* space2
- Space* space3
- Space* space4
- Space* space5
- Space* space6
- Space* space7
- Space* space8
- Space* space9

- Space* playerLocation

Member functions:

Board()

- space1 = new Bibliotheca()
- (assign each other space pointer to the type of space to which it corresponds on the board layout)
- playerLocation = space5 // player starts in forum
- Set each space's directional pointers (if a space does not have another space in a given direction, that direction's pointer will be left as the default value of nullptr)
- ifstream inputFile("Board_Images.txt")
- string image
- for(int index = 0; index < NUM_BOARD_IMAGES; index++)
    - getline(inputFile, image, DELIM)
    - boardImages.push_back(image)
- inputFile.close()

~Board()

- delete space1
- space1 = nullptr
- (repeat for other 8 spaces)
- playerLocation = nullptr

Space get_player_location() const*

- return this->playerLocation;

void print_board()

- // Determine which board image to print depending on the player's location.
- int boardIndex = (playerLocation->get_num() – 1);
- cout << boardImages[boardIndex]

void move()

- vector<string> moveChoices(8);
- set_move_menu(moveChoices);
- Print: "In what direction would you like to move?";
- // declare variables for use in do-while loop
- bool validMove = false;
- Direction dir;
- do // use do-while loop to read in and process move choice. Repeat until a valid move is selected.
    - int moveChoiceNum = menu(moveChoices)
    - moveChoiceNum-- // decrement moveChoiceNum since it will be static cast to the corresponding direction
    - dir = static_cast<Direction>(moveChoiceNum);
    - validMove = is_valid_move(dir);

    - if (!validMove)
        - Print: you cannot move in that direction; please pick a different direction.
- … while (!validMove);
- Now that the move has been validated, move the player to the new space.
- if (dir == Direction::N)
    - playerLocation = playerLocation->get_north();
- (repeat for other 7 directions)

private void set_move_menu(vector<string>& moveChoices)

- To keep consistency with what each direction's number is in the menu, all directions will be printed as choices, even if they player cannot move in that direction. After the name of the direction, the name of the space that is in that direction will be printed or the message "cannot move in this direction" will be printed if applicable.
- string cannotMove = "(cannot move in this direction)"
- string directionMenuOption = "North: ";
- if (playerLocation->get_north != nullptr)
    - directionMenuOption += playerLocation->get_north()->get_name();
- else
    - directionMenuOption += cannotMove;
- moveChoices.push_back(directionMenuOption);
- directionMenuOption = "Northeast: "
- (repeat same process used for north for the other 7 directions)

private bool is_valid_move(Direction dir)

- if (dir == Direction::N)
    - if (playerLocation->get_north == nullptr)
        - return false
    - else
        - return true
- (repeat same process for other 7 directions)

Game.hpp / Game.cpp

Constant:

- const int MAX_STEPS = 75

Private Data Members:

- Board gameBoard
- set<string> satchel
- int money
- int stepsTaken
- int stepsSinceBathing
- bool hasBathed
- bool knowsAboutScroll
- bool stillAlive
- bool withNero

- bool gameOver

Member Functions:

Game()

- money = 0
- stepsTaken = 0
- stepsSinceBathing = 0
- hasBathed = false
- knowsAboutScroll = false
- stillAlive = true
- withNero = false
- gameOver = false

void take_turn()

- print steps taken of steps allowed
- print coins
- print board
- print_satchel_contents()
- Space* currentSpace = gameBoard.get_player_location()
- print currentSpace->get_name()
- print currentSpace->get_description()
- menu: “enter currentSpace->get_name()”, “continue moving”
- if (choice == 1)
    - currentSpace->interact(satchel, money, stepsSinceBathing, hasBathed, knowsAboutScroll, stillAlive, withNero)
        - (clear screen at beginning of interact functions; still print coins, satchel, and space name)
    - if (withNero)
        - ending_sequence()
    - else if (stillAlive)
        - if (stepsTaken == MAX_STEPS)
            - Print: You have reached the maximum number of steps, and you are not with Nero. We are now going to bring you back to the present since we don’t want you alone in the city of Rome without the emperor’s protection at night.
            - gameOver = true
        - else
            - clear screen and print same info as at beginning of turn
            - gameBoard.move()
            - stepsTaken++
            - if (hasBathed)
                - stepsSinceBathing++
    - else
        - gameOver = true

```
• else if (choice == 2)
    o if (stepsTaken == MAX_STEPS)
        ▪ Print: You have reached the maximum number of steps, and you are not with
          Nero. We are now going to bring you back to the present since we don’t want
          you alone in the city of Rome without the emperor’s protection at night.
        ▪ gameOver = true
    o else
        ▪ gameBoard.move()
        ▪ stepsTaken++
        ▪ if (hasBathed)
            • stepsSinceBathing++
```

private void print_satchel_contents()

```
• if (satchel.empty())
    o cout << “Your satchel is currently empty” << endl
• else
    o set<string>::iterator iter
    o cout << “Satchel Contents: “
    o for (iter = satchel.begin(); iter != satchel.end(); iter++)
        ▪ cout << *iter
        ▪ // test to see if this is the last element by incrementing iter, seeing if it equals
          satchel.end(), and then decrementing it to get it back to current value
        ▪ iter++
        ▪ if (iter != satchel.end())
            • cout << “, “
        ▪ iter--
    o cout << endl
```

private void ending_sequence()

```
• ifstream inputFile(“Ending_Sequence.txt”)
• string text
• while (getline(inputFile, text, DELIM)
    o Print: text
• inputFile.close()
• gameOver = true
• Have user press enter
```

bool game_over() const

```
• return this->gameOver
```

enterValidInt.hpp / enterValidInt.cpp:

- Utility function implemented as part of CS 162 Lab 1 and updated for Lab 3.
- Function does not have any parameters and returns a valid integer.
- User input is read into a string using getline(cin, input)

- A while loop begins that does not terminate until the user has entered a valid integer with no additional “garbage input.”
- If the user only presses enter and the input string remains empty, a nested while loop keeps prompting the user to enter an integer until at least one character (other than the newline character) is entered.
- Else, every character in the string is processed. First, the character at position 0 is checked to verify that it is either a – or a digit.
- All other characters in the string are checked through a for loop to ensure that they are all digits.
- If any of the characters in the string are invalid according to the rules above, the user is prompted to enter an integer again.
- Before exiting the loop, the stoi function is called within a try-catch block. If any exceptions are thrown, the catch block sets the validInt flag back to false, calling the while loop to iterate again to collect new user input. This ensures that no integers too large or too small to be stored as an int cause the stoi function to throw an exception and the program to crash.
- In addition to ensuring that the user enters a valid integer, the function ensures that only one integer is entered at a time.

getRandomInt.hpp / getRandomInt.cpp:

- Utility function developed as part of CS 162 Project 1
- Function parameters are the lower and upper bounds of the randomized integer (the returned integer can be greater than or equal to the lower bound and less than or equal to the upper bound).
- Time function is called to produce a seed for rand function.
- Srand function is passed the seed.
- Random number is generated by rand.
- Restrict the random number to the specified range in function paramaters:
  - Use the mod operator to find the remainder when the random number is divided by (upperBound – lowerBound + 1) and add the lowerBound to the result
- Return the random number within the specified range to the calling function.

menu.hpp / menu.cpp:

- More general-purpose easy-to-reuse menu utility function that I developed as part of CS 162 Lab 4
- Function receives parameter of vector of strings containing menu choices
- Prints stars for top of menu
- Uses for loop to auto-number (with value of index+1) and display each menu choice on its own line
- Prints stars for bottom of menu
- Prompts user for choice using enterValidInt utility function that I created
- Uses while loop to continually reprompt user for choice using enterValidInt if choice number is less than 1 or greater than size of menu choices vector

- Returns validated menu choice to calling function

pressEnter.hpp / pressEnter.cpp:

- Utility function developed during Week 1 of class to cause output to pause until user presses enter.
- Void function with no paramaters.
- Validates that user only presses enter and ensures that no extra “garbage input” causes the program to crash.
- Input is read in as a string using getline(cin, input).
- If the string read in is not an empty string, user is prompted to simply press enter and told that no other input is allowed.
- This not only takes care of garbage input that is stored in a local string variable to the pressEnter function and then destroyed, but it also ensures user understands that anything entered before pressing enter when prompted to simply press enter to continue will be not be stored in memory for later use by the running program.

finalProjMain.cpp

```
bool playAgain = false
do
    Clear screen
    // print game instructions
    ifstream inputFile(“Game_Instructions.txt”)
    string text
    while (getline(inputFile, text, DELIM)
        Print: text
    Have user press enter
    Game myGame
    do
        myGame.take_turn()
    … while (myGame.game_over() == false)
    vector<string> mainMenu = {“Play again”, “Exit”}
    int mainMenuChoice = menu(mainMenu)
    if (mainMenuChoice == 1)
        playAgain = true
    else
        playAgain = false
… while (playAgain)
return 0
```

Testing Plan

** Please Note: The utility functions (enterValidInt, getRandomInt, menu, and pressEnter) will not be specifically tested since these have been tested during the implementation of previous labs and projects in this course and verified to work correctly. **

Test Description	Expected Results	Actual Results
Test 1: This game will be tested incrementally as it is developed. The first test will be run once the Space parent class has been created and stubs have been created for the child classes (to test making spaces of each different type before their specific interact functions are fully implemented). The Board and Game classes as well as the main function will be fully developed. This first test is to ensure that: • the map displays properly and indicates the player's location accurately. • Each space points to the correct surrounding spaces, with unused pointers pointing to nullptr. • The menu only allows the user to move to adjacent spaces. • When the step limit has been reached (which will be set to a much lower limit of 20 steps for testing purposes), the game ends. For the purposes of this test, any time the player reaches a new space, they will just be	The game will allow the player to move around according to game rules, only allowing the player to go to another space touching the current space (either on an edge or a diagonal). The game will not let the player go out-of-bounds. When the step limit has been reached, the game will end.	After fixing glitch where board wasn't printing due to incorrect file name for file of board images, everything worked as expected. No memory leaks or segmentation faults.

prompted to move to another space. In subsequent tests, the interact functions of specific spaces will be tested.		
Test 2: Test Circus Maximus. Since the Circus Maximus will be the first space whose interact function is implemented, code will be changed so that the user starts with 20 coins instead of 0 coins for testing purposes.	Winning and losing will function correctly (user doubles their bet if they win and lose it if they lose). Space will properly handle user having too little money for a given bet or no money at all.	Once I added in a statement setting validMove to true if the user chose the lowest bet in CircusMaximus::race, everything worked as expected.
Test 3: Test Forum. Test to ensure that the player can purchase items. Satchel set up as set of strings functions properly. Starting money amount is 30 coins for testing purposes.	Game ensures that user has enough money to purchase a given item. If their satchel is full, they cannot purchase anything. If they already have a given item, they cannot purchase a duplicate item until they have given the first of that item away.	Everything functioned as expected.
Test 4: Theatrum Test to ensure the user can donate items. For testing purposes, the user has the scroll and the permit in their satchel. Then, in the next running of this test, their satchel will be empty.	The donation menu will be constructed properly. If the player has no items to donate, that situation will be handled properly. The player will not be allowed to donate key items (scroll or permit).	Everything worked as expected.
Test 5: Domus Aurea Test to ensure the game correctly determines whether or not the user has won. In different executions of test, will place permit in satchel, and will initialize hasBathed to true.	Correctly responds to whether or not user has permit, whether or not user has bathed at any point, and whether the user has bathed within the past 2 turns.	Everything worked as expected.
Test 6: Thermae Test to ensure game allows user to pay 2 coins to bathe and	Once the user bathes for the first time, hasBathed will be set to true, and the game will begin counting how many steps the user has taken since their last	Everything worked as expected.

changes relevant variables accordingly.	bath. The stepsSinceBathed will then be reset to 0 anytime the user bathes again.	
Test 7: Campus Martius (Part 1) After implementing the test_user private member function (called by the interact function), test it to make sure it handles correct and incorrect answers as expected.	If the player answers a question incorrectly, they immediately die. If they answer all 3 questions correctly, they pass the test, and the soldier tells them which 2 items he wants from the Forum. When subsequently returning to the CampusMartius, they are not tested again within the same game.	Everything worked as expected.
Test 8: Campus Martius (Part 2) Test the give_items and check_for_item functions once implemented. For purposes of this test, user starts with 30 coins so that they can buy requested items.	The soldier will only accept the requested items (randomly selected by CampusMartius constructor). The game will properly respond to the user returning with neither item, the user returning with both items at once, returning with just one item, returning again without the second item, and returning after giving both items.	Everything worked as expected.
Test 9: Ludus Test to ensure that trivia works as expected. During first run of test, change code so that computer always gets a point. This will test tying / going through all 50 questions.	Wins and losses in regular play, wins and losses in tie-breaker rounds, and ties will be handled properly. Only 5 games will be allowed. Game will be able to ask all 50 trivia questions in question vector if necessary.	Everything worked as expected
Test 10: Bibliotheca / Ludus Test to ensure user can be told about scroll and return it. The interaction of schoolteacher with user when they ask for scroll will also be tested.	Game will allow user to learn about scroll upon first visit, prompt the user for the scroll again if they return without it, reward them when they get it from the schoolteacher, and tell them library is closed any subsequent times they visit. Schoolteacher will try to give scroll but keep it if user has too many items in satchel until they come back with room in the satchel. Then, teacher will give	Everything worked as expected. Now that game is nearly complete (except for Colosseum space, which is optional space), tested meeting game objectives to win the game. Won the game successfully.

	scroll. After user turns in scroll to librarian, teacher will not try to give it again.	
Test 11: Colosseum Test fighting gladiatorial battles.	The game will properly handle who wins / loses the match as well as what move beats what other move. If the player loses and the senator chooses for them to die, dying will work successfully.	Works as expected.

Reflection:

During implementation of the code, I made the following changes:

- Deleted bool hasObtainedPermit from CampusMartius.hpp because playerkeeps permit until they win the game. Therefore, the prefect knows whether or not it is in their satchel.
- Realized that return type of Board::get_player_location() needed changed from Board* to Space* since it returns a pointer to the space where the player is currently located.
- In Game constructor, added in line initializing money to 0 (had unintentionally omitted this line from pseudocode but caught the mistake when implementing the Game class).
- In various class hpp files (such as Ludus.hpp), removed size specifier from vector declarations since they caused a compiler error.
- Wrote incorrect file name in Board.hpp for Board_Images.txt. Made correction so that Board would print.
- CircusMaximus::race: added statement setting validBet to true if the user picks the low bet (previously had forgotten that statement, resulting in the loop continuing to iterate if the user chose the low bet with potential for endless loop if that was the only bet they could do).
- Forum::interact: changed conditional testing for whether or not user wants to leave forum to check for purchase choice equaling 7. 7 indicates they want to exit, not 6 (there are 6 items they can buy + the choice to exit, and menu numbering starts at 1).
- Slightly revised schoolmaster's statements in Ludus::interact so that it is clear whether or not he is giving the scroll to the user. (Previously, he said "Here is the scroll" and then later said that he saw the user didn't have room for the scroll in the satchel. New wording of message makes it clearer.)
- Ludus::play_trivia – instead of randomly generating 0 or 1 and adding it directly to computer's score, changed logic slightly so that whether computer got question right or wrong is reported. Now, 0 or 1 is randomly generated, static cast to a bool, and then it is reported whether they computer got the question right or wrong. A point is added to their score if they got it right.
- Since random shuffle seemed to be shuffling questions in the same order in Ludus constructor, added in code to generate a random number between 1 and 50 and shuffle the questions that many times to make their order more truly random.

This was a very fun project to complete. As a Latin teacher, I am very passionate about Ancient Rome, which is the basis of this game. This project allowed me to review important concepts from throughout this class, such as dynamic memory allocation, STL containers and algorithms, and file I/O. Participating in the Weeks 9-10 discussion was helpful for creating my test plans, as it helped my tests be more focused on key special cases to test for while also being more efficient.