**Objective:** You will design and implement a text-based game or puzzle where the player moves

through a series of rooms or compartments. Each space will be a class with (at least) four pointer

variables that link to other spaces. You must have at least 5 spaces of at least 3 different types. You will have a space abstract class that will have a special pure virtual function. Each type of space will have a special action. You will have at least 3 derived classes for different types of spaces. You will have at least 3 derived classes for different types of spaces.

You must have some way to keep track of which space the player is in. The player will have a container (backpack, knitting bag, or notebook) to carry “items”. The container must have some limit. One or more of these items will be required as part of the solution, such as a “key” to open the locked door. You should have a time limit to urge the player on. This does not mean a literal clock, just some way to prevent the ‘game’ from going on indefinitely. The player must interact with parts of the structure, and not just simply collect things.This can be throwing something at the monster, operating a light switch (or other control), opening doors, or singing to get the baby back to sleep.

**Theme:** Game of Life – Make it through with enough money for retirement. You will start off with a certain amount of funds before the game begins. Player will traverse through the different spaces (school, apartment, work, new house, lawyer’s office, retirement community) in order to perform the necessary tasks before going to the retirement community. Player must have a minimum amount of money in order to get into the retirement community. Tasks player must perform:

* Get a degree from school
* Obtain a house (not an apartment)
* Get married
* Get one bonus at work
* Obtain enough money to retire (set amount of money)

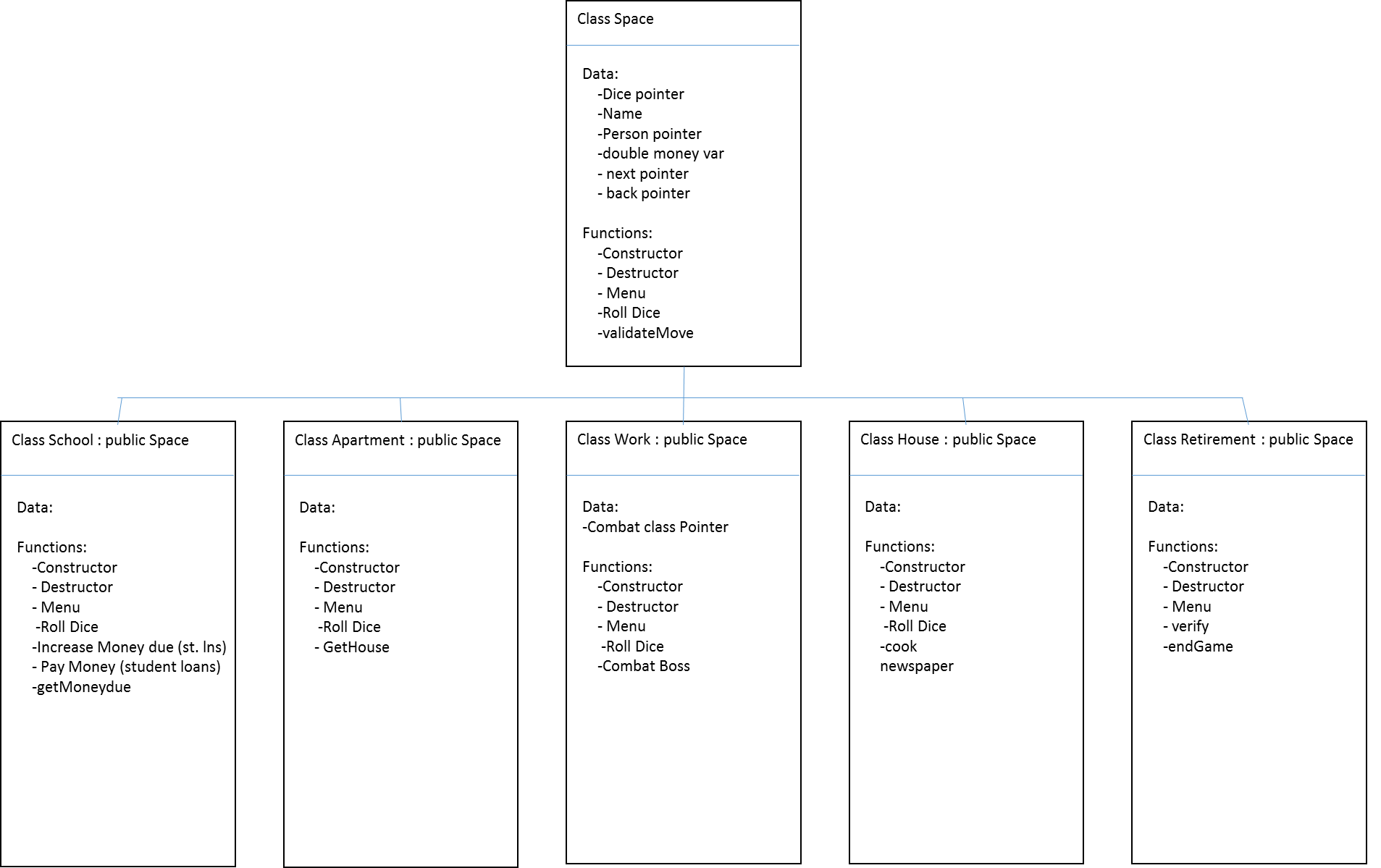
List of Rooms and games within rooms:

1. School:
   1. Answer random trivia questions to get your degree
   2. if you win you get your degree and have $500 worth of student loans
   3. if you lose, you must try again for your degree and will be charged another $500.
2. Apartment:
   1. Roll dice for your house type (trailer, rancher, mansion, etc.)
   2. Options range in price
   3. Once you get a house, your apartment will be deleted and your house space will be created.
3. Work:
   1. Combat game against boss
      1. Lose = must take a vacation to de-stress (losing money)
      2. Win = get bonus! (gain money)
      3. Job: random chance that when you go into the work space that you will receive your salary’s pay.
4. New House:
   1. Have option of cooking dinner or reading the newspaper
   2. Reading newspaper will allow you to donate plasma in order to get more money.
   3. Cooking will give you a random chance to lose or make money
5. Lawyer’s Office:
   1. Must go here in order to get married
   2. Have the option of fighting to erase your student loans.
6. Retirement Community:
   1. If you have enough money, and meet all the requirements, game over – you win!
   2. If you do not meet the requirements, you will be rejected from entering and must go back to work in order to get more money before the time runs out.

**Breaking Down the Code:**

|  |  |
| --- | --- |
| Data Needed | Actions |
| Person Class   * Money (double) * StudentLoans (double) * House * Bonus * Marriage * Degree * Item container | Person   * getStudentLoans * getMoney * get/set Bonus * get/set degree * get/set marriage * add to container * get container |
| Space (abstract class)   * Person object * Name of room * Next pointer * Back pointer * up pointer (ptr1) * down pointer (ptr2) | Space (abstract class)   * Menu * Validate move/links * getName of room |
| List   * head pointer * tail pointer * Person pointer | List   * addSpace * removeSpace * setPerson * setLocation * getLocation * move person |
| School   * Degree (will be a bool based on game results) * Physical degree | School   * Increase Student Loans (if trying to earn degree) * Pay Student Loans * Get physical degree * Get intellectual degree |
| Apartment   * Dice Class * House price (array of ints) | Apartment   * Increase house debt * Get house debt * Roll Dice |
| Work   * Combat class * Dice class * Salary (double) | Work   * Combat * Roll Dice * getSalary * Increase Salary (promotion) |
| New House   * Dice class * Interactive Item | Home   * Roll Dice * Newspaper * Cooking |
| Retirement Community   * Person pointer * Game over flag | Retirement Community   * Verify requirements * End game |
| Lawyer office   * Creature 1 * Creature 2 * Marriage bool | Lawyer Office   * Get married * Fight for no student loans |
| Main game class   * Timer for game * List object * Person object * Space object | Main game class   * Main menu * Room menu * Requirements list |

**Space Class Hierarchy:**



**Design/Implementation 1**: Creating a linked structure with 4 pointers per class, and moving through each structure.

**Pseudocode:**

*Space.hpp file:*

Class Space {

Protected:

String Name

Space \*next

Space \*back

Space \*ptr1

Space \*ptr2

Friend class List

Public:

Constructor()

Destructor()

Virtual String getName()

Virtual bool validMove(Space \*)

Virtual void menu()

}

*Space.cpp file:*

Constructor

SET Name = space

SET 2 ptr Pointers = NULL

SET next and back = NULL

Destructor {}

String getName()

RETURN name

Bool validMove(Space \*s)

CREATE bool vMove

SET vMove = false

IF (this->getName() = s’s getName())

DISPLAY “Re-entering current room”

SET

ELSE IF (this->ptr1 = s OR this->ptr2 = s)

DISPLAY “You are now going to s’s getname()”

SET vMove = true

ELSE IF (this->next = s OR this->back = s)

DISPLAY “you are not going to s’s getname()”

SET vMove = true

ELSE

DISPLAY “this move is illegal”

SET vMove = false

RETURN vMove

Void Menu() {}

*List.hpp file:*

Class List {

Private:

Space \*head

Int listLength

Public:

Constructor

Destructor

Void addSpace( Space \*, int)

Void removeSpace(int)

Void getNodeNames()

}

*List.cpp file:*

Constructor

SET head = NULL

SET listLength = 0

CREATE Game Layout

Destructor

CREATE Space \*prev

SET prev = head

CREATE Space \*ptr

SET ptr = head

WHILE (ptr != NULL)

SET prev = ptr

SET ptr = ptr’s next pointer

DELETE prev

Void addSpace (Space \*newSpace, int position)

CREATE Space \*ptr

SET ptr = head

CREATE Space \*prev

SET prev = head

CREATE int count

SET count = 0

IF (head is NULL)

SET head = newSpace

Set head’s next pointer = NULL

ELSE

WHILE (ptr is not NULL)

IF (position equals count)

SET prev’s next pointer = newSpace

SET newSpace’s next to ptr

INCREMENT listLength

SET prev = ptr

SET ptr = ptr’s next

INCREMENT count

IF (position equals count)

SET prev’s next pointer = newSpace

SET newSpace’s next to ptr

INCREMENT listLength

Void removeSpace(int position)

……

Void getNodeNames()

CREATE Space \*temp

SET temp = head

WHILE (temp is not NULL)

DISPLAY temp’s getName()

SET temp = temp’s next

All derived classes in phase 1 will follow the following design:

*School.hpp file:*

Class School : Public Space {

Public:

Constructor

Destructor

getName()

}

*School.cpp file:*

Constructor()

SET Name = School

Destructor()

IF school

DELETE school

String getName()

RETURN name

**Design/Implementation 1 Test Plan:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Input Value | Driver Function | Expected Outcome | Actual Outcome |
| Space pointer to School class calls School’s getName funct | Space \*sp2 = new School;  Sp2->getName() | Main()  getName() | “School” | “School” |
| List Class addSpace function adds node to beginning of list | L->addSpace(sp, 0);  L->getNodeNames(); | Main()  addSpace() | “space” | “space” |
| List Class addSpace function adds node to second spot of list | L->addSpace(sp, 0);  L->getNodeNames(); | Main()  addSpace() | “space”  “School” | “space”  “School” |

**Design/Implementation 2: Going from Singly linked to Doubly linked**

List.cpp file was updated with blue highlighted statements as follows:

List Constructor()

SET head = NULL

SET tail = NULL

SET listlength = NULL

List addSpace()

CREATE Space \*ptr

SET ptr = head

CREATE Space \*prev

SET prev = head

CREATE int count

SET count = 0

IF (head is NULL)

SET head = newSpace

SET head’s next pointer = NULL

SET head’s back pointer = NULL

ELSE

WHILE (ptr is not NULL)

IF (position equals count)

SET prev’s next pointer = newSpace

SET newSpace’s next to ptr

SET newSpace’s back to prev

INCREMENT listLength

SET prev = ptr

SET ptr = ptr’s next

INCREMENT count

IF (position equals count)

SET prev’s next pointer = newSpace

SET newSpace’s next to ptr

SET newSpace’s back to prev

INCREMENT listLength

List getNodeNames()

\*NOTE: new while loop only incremented for testing purposes. Will be commented or deleted out in finalized code.

CREATE Space \*temp

SET temp = head

CREATE Space \*bckwrds

SET bckwrds = head

WHILE (temp is not NULL)

DISPLAY temp’s getName()

SET temp = temp’s next

DISPLAY “traverse backwards”

SET temp = bcwrds

WHILE (temp is not NULL)

DISPLAY temp’s getName()

SET bckwrds = temp

SET temp = temp’s back

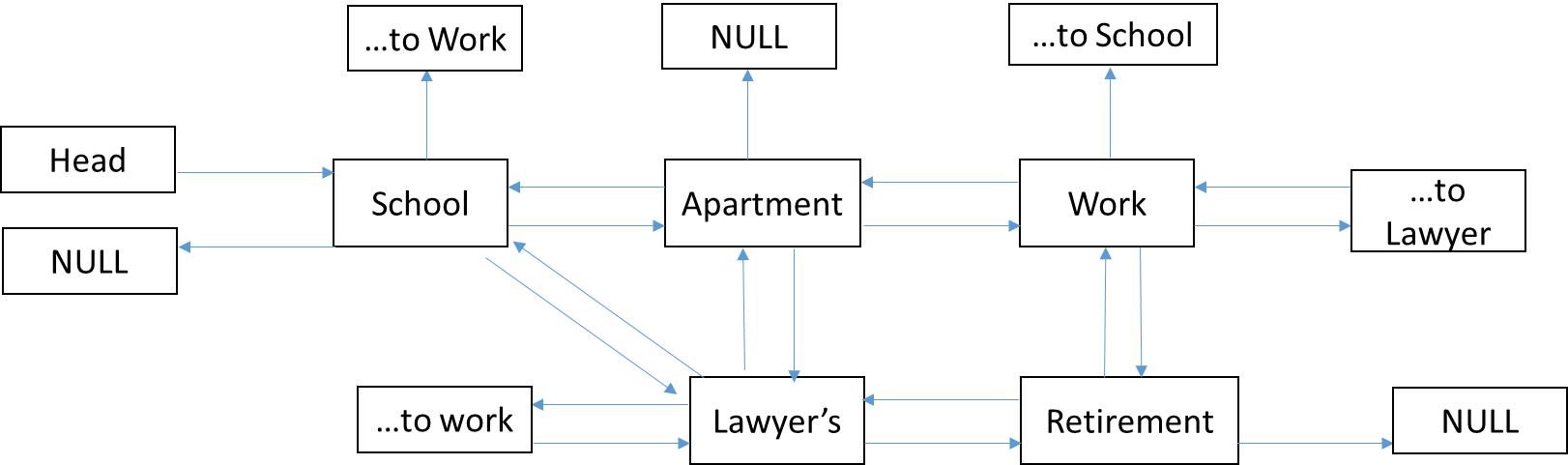
**Design/Implementation 2: Test Plan**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Input Value | Driver Function | Expected Outcome | Actual Outcome |
| Space object back pointers able to traverse backwards to previous node | L->getNodeNames() | Main()  GetNodeNames() | “space”  “School”  “Apartment”  “Work”  “Retirement”  “Traverse backwards: “  “Retirement”  “Work”  “Apartment”  “School”  “space” | “space”  “School”  “Apartment”  “Work”  “Retirement”  “Traverse backwards: “  “Retirement”  “Work”  “Apartment”  “School”  “space” |

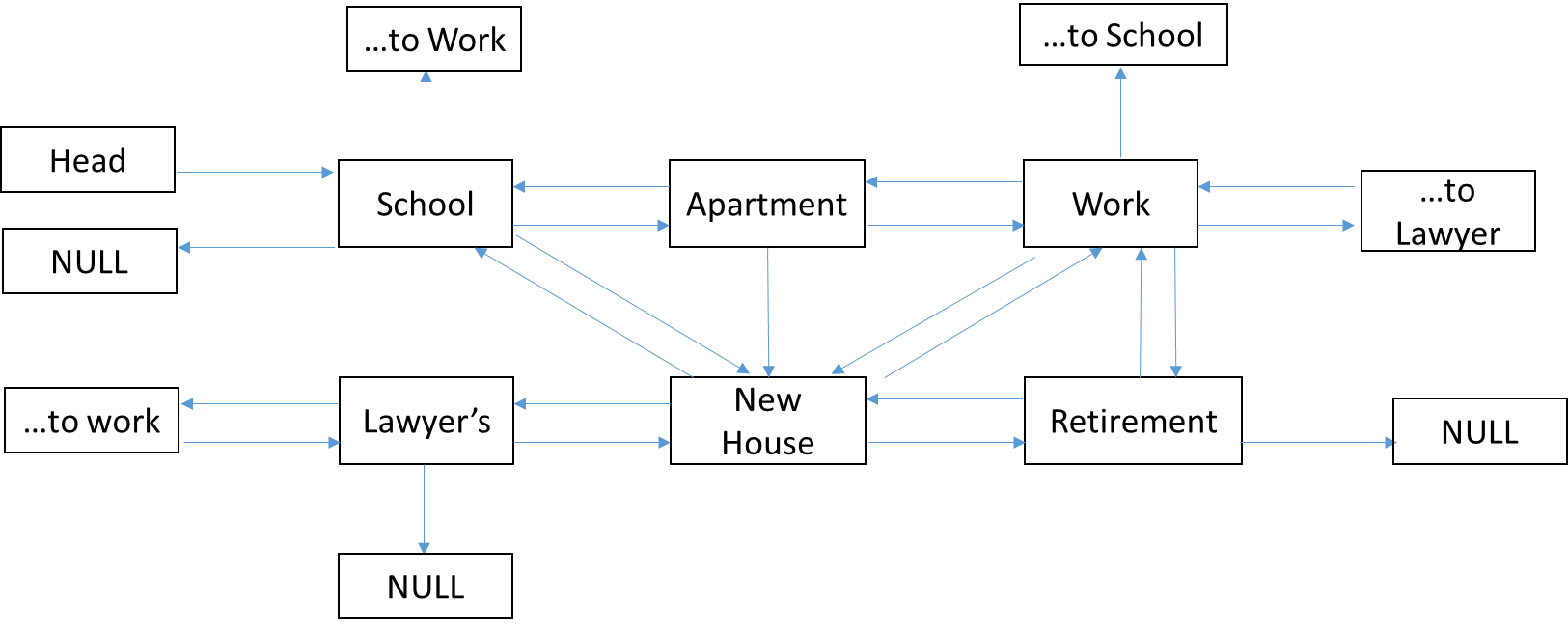
**Design/Implementation 3: Interconnecting Remaining Pointers**

**Connections:**

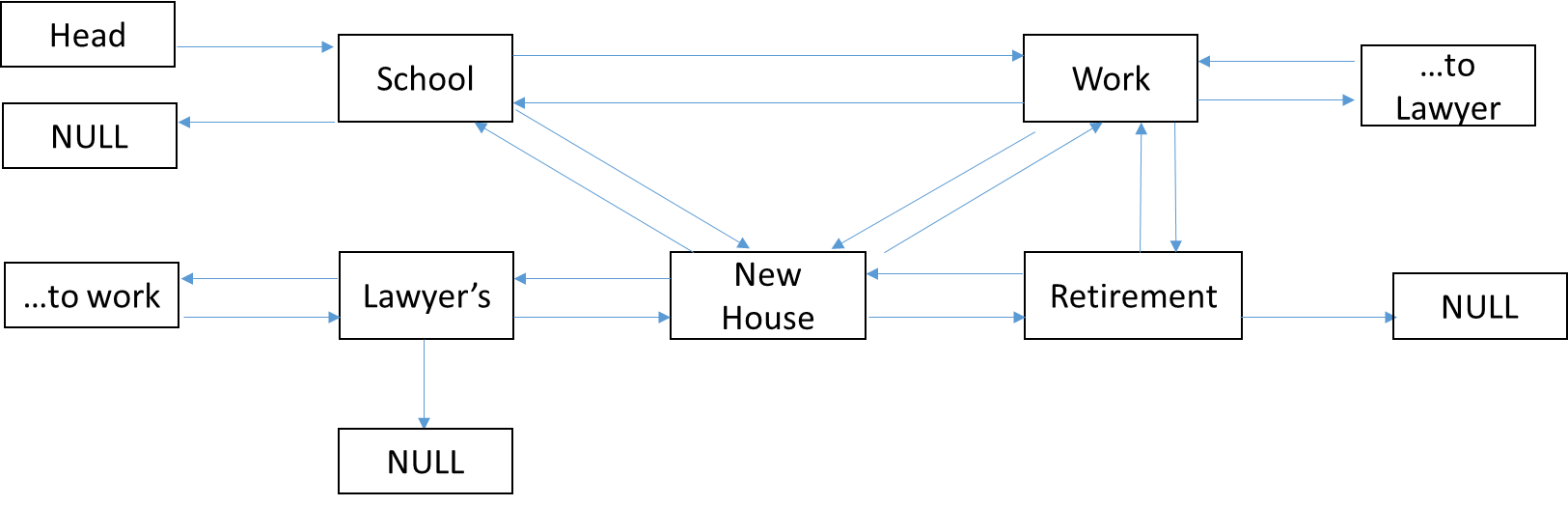
(*Before Apartment is deleted):*



*While house is being added:*



*After Apartment is deleted:*



**Pseudocode:**

Added in createGame() function to List class:

*List.cpp file:*

Constructor()

SET head = NULL

SET tail = NULL

SET listlength = NULL

SET player = NULL

Void List::createGame() {

CREATE Space \*sp

SET sp to each space type

CALL addSpace function for each space type

(This creates a doubly linked list)

CREATE Space \*temp

SET temp = head

//Assigning school pntrs to work and lawyer

SET head’s ptr1 = head next, next

SET head’s ptr2 = head next, next, next

SET temp = temp’s next

//Assigning Apartment to lawyer

SET temp’s ptr1 = head’s next, next, next

SET temp = temp’s next

//Assigning Work to school and retirement

SET temp’s ptr1 = head

SET temp’s ptr2 = temp’s next, next

SET temp = temp’s next

//Assigning Lawyer to apartment and school

SET temp’s ptr1 = head’s next

SET temp’s ptr2 = head

SET temp = temp’s next

//Assigning retirement to work

SET temp’s ptr1 = temp’s back, back

}

**Test Plan:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Input Value | Driver Function | Expected Outcome | Actual Outcome |
| Go from school to work node | 3 | L->move  School->validMove() | “You are now going to work” | “You are now going to work” |
| Go from school to Retirement | 5 | L->move  School->validMove() | “illegal move” | “illegal move” |

**Design/Implementation 4: Creating Person Class and Moving Through Structure**

*Person.hpp file:*

Class Person {

Private:

String pName

Double money

Double studentLoans

Int kidsNum

String container[2]

Bool house

Bool degree

Bool physDeg

Bool bonus

Bool end

Public:

Constructor(string)

Constructor()

Destructor()

String getName()

Double getStudentLoans()

Void addStudentLoans(double)

Double getMoney()

Void addMoney(double)

Bool getHouse()

Void setHouse(bool)

Bool getDegree()

Void setDegree(bool)

Bool getPhysDeg()

Void setPhysDeg(bool)

Void addContainer(string &)

String \* getContainer()

Void setBonus(bool)

Bool getBonus()

Void setEnd(bool)

Bool getEnd()

*Person.cpp file:*

Constructor (string nameIn)

SET pName = nameIn

SET money = 100

SET studenLoans = 0

SET kidsNum = 0

SET loc = NULL

SET house = false

SET degree = false

SET physDeg = false

SET end = false

FOR (int I = 0 to 1)

SET container[i] = “0”

Constructor ()

SET pName = “Player 2”

SET money = 100

SET studenLoans = 0

SET kidsNum = 0

SET loc = NULL

SET house = false

SET degree = false

SET physDeg = false

SET end = false

FOR (int I = 0 to 1)

SET container[i] = “0”

Destructor()

Space \* getName()

RETURN name

Double getMoney()

RETURN money

Void addMoney(double m)

SET money = money + m

Double getStudentLoans()

RETURN studentLoans

Void addStudentLoans(double s)

SET studentLoans = studentLoans + s

Void addContainer(string &s)

FOR (int I = 0 to 1)

IF (container[i] = “0”)

SET container[i] = s

NOTE: all remaining get/set functions will take the following form:

Void setHouse (bool h)

SET house = h

Bool getHouse()

RETURN houe

List.cpp file was updated with blue highlighted statements as follows:

Void addSpace (Space \*newSpace, int Position)

CREATE Space \*ptr

SET ptr = head

CREATE Space \*prev

SET prev = head

CREATE int count

SET count = 0

IF (head is NULL)

SET head = newSpace

SET head’s next pointer = NULL

SET head’s back pointer = NULL

CALL player’s setLoc()

SEND head as parameter

NOTE: move() and setPlayer() are new functions

Void move (int input)

CREATE Space \*nwRm

SET nwRm = head

IF (input = 1)

SET nwRm = head

ELSE IF (input = 2)

SET nwRm = head’s next

ELSE IF (input = 3)

SET nwRm = head’s next, next

ELSE IF (input = 4)

SET nwRm = nwRm’s next

SET nwRm = nwRm’s next, next

ELSE

SET nwRm = nwRm’s next, next

SET nwRm = nwRm’s next, next

CREATE Space \*temp

SET temp = players location

CREATE bool vMove

SET vMove = temp’s validMove()

SEND it to nwRm

IF (vMove)

CALL player’s location

SEND nwRm

Void removeSpace(int position)

CREATE Space \*ptr

SET ptr = head

CREATE Space \*prev

SET prev = head

IF (head is NULL)

DISPLAY “List empty”

ELSE IF (position is 0)

SET head = head’s next

SET ptr = head

DELETE prev

SET prev = ptr

DECREMENT listLength

ELSE

WHILE (ptr)

IF (position = count)

SET prev’s next = ptr’s next

DELETE ptr

SET ptr = prev’s next

SET ptr’s back = prev

DECREMENT listLength

SET prev = ptr

SET ptr = ptr’s next

INCREMENT count

IF (position = listLength)

SET prev’s next = NULL

DELETE ptr

SET ptr = prev’s next

SET tail = prev

DECREMENT listLength

Void setPlayer (Person \*pIn)

SET player = pIn

**Test Plan:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Input Value | Driver Function | Expected Outcome | Actual Outcome |
| Move player’s location to same room | 1 (School) | Main()  Move()  validMove()  setLoc() | “Original location: 0x7013c0” (for example)  “New location:  0x7013c0” | “Original location: 0x7013c0” (for example)  “New location:  0x7013c0” |
| Move player’s location to acceptable choice (link to space) | 2 (Apartment) | Main()  Move()  validMove()  setLoc() | “Original location: 0x7013c0” (for example)  “New location:  0xfc1400” (for example) | “Original location: 0x7013c0” (for example)  “New location:  0xfc1400” (for example) |
| Move player’s location to unacceptable choice (no link to space) | 5 (Retirement) | Main()  Move()  validMove()  setLoc() | “That move is not legal. Pick again. “ | “That move is not legal. Pick again. “ |

**Design/Implementation 5: Converting Space Class to Pure Virtual**

* Set all functions equal to zero.
* Copied all functions (getName & validMove) to derived classes
* Added Menu function to Space class and all derived classes (empty placeholder function so far).

**Test Plan:** compiler run.

**Design/Implementation 6: Creating the Work Class:**

* Added in the combat class and two creature pointers, and double therapyMoney and double bonus
* Added in setPerson function, and Person pointer to Space abstract class
* Developed Menu function

*Work.cpp file:*

Bool combat()

CREATE new BlueMenYou object with you pointer

CREATE new MedusaBoss object with boss pointer

CREATE double attackRoll for fight

CREATE bool dead

CREATE bool youDied

SET equal to false

CREATE double lifeStrength for fight

DO {

DISPLAY strength of each character by using character’s getStrength()

DISPLAY name of each character by using character’s getName()

CALL boss’s attack function

CALL your defense function

SEND boss’s attack value

GET your strength after attacks

CALL your dead function

IF (dead)

DISPLAY “You lost. Pay $$ for therapy. “

SET youdied = true

ELSE

CALL your attack function

CALL boss’s defense function

SEND your attack value

IF (dead)

DISPLAY “Boss lost the fight. You get raise!”

SET youdied = false

} WHILE (not dead)

RETURN youdied

Void setPerson(Person \*pIn)

SET player1 = pIn

Void menu()

CREATE bool payMoney

DISPLAY story about boss wanting to fight

DISPLAY “Hit enter to begin your fight”

IGNORE leftover buffer newline

GET next input

SET payMoney = combat() call

IF (payMoney)

CALL person’s addMoney() function

SEND therapyMoneuy

ELSE

CALL person’s addMoney() function

SEND bonus

**Test Plan 6:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Input Value | Driver Function | Expected Outcome | Actual Outcome |
| Work menu Displays when menu function is called | nwRm->menu() | List::move()  Work::menu() | “You are at work….” | “You are at work….” |
| Work combat function produces winner | None | Work::combat() | “Boss lost” or “Boss wins” | “Boss lost” or “Boss wins” |
| Person’s money increases by X amount when player wins battle | none | Work::combat()  Work::Menu()  Person::addMoney() | “Player’s original money: 100 “  “player’s new money: 300” | “Player’s original money: 100 “  “player’s new money: 300” |
| Person’s money decreases by X amount when player loses battle | none | Work::combat()  Work::Menu()  Person::addMoney() | “Player’s original money: 100 “  “player’s new money: -900” | “Player’s original money: 100 “  “player’s new money: -900” |

**Design/Implementation 7: Creating the Apartment Class:**

* Developed Menu
* Added in rollHouseDie() which rolls a die to select a house type and cost for player to buy

*Apartment.cpp file:*

Int Menu()

CREATE char house

DISPLAY “you are in your apt would you like to look for a house?”

SET house to input

IF (house == yes)

CALL rollHouseDice function

ELSE

DISPLAY “please come back when you are ready to buy a house”

Void rollHouseDie()

CREATE string houseType

CREATE double cost

CREATE char buy

CREATE bool bought

SET = false

CREATE new DiceMod from DiceMod pointer

CALL die1’s setSideCount ()

Send 3 to function

CREATE double result

CREATE int result2

SET result = die1’s rollDie()

SET result2 = type-casted result

DISPLAY “Press enter to see your house type”

IGNORE input

GET input

SWITCH (result2)

CASE 1:

DISPLAY “trailer. Cost = 800. Want to buy?”

SET buy = input

SET houseType = trailer

SET cost = -800

Break

CASE 2:

DISPLAY “split level. Cost = 4000. Want to buy?”

SET buy = input

SET houseType = split level

SET cost = -4000

Break

CASE 3:

DISPLAY “Mansion. Cost = 10,000. Want to buy?”

SET buy = input

SET houseType = mansion

SET cost = -10000

Break

IF (buy = y or Y)

DISPLAY “Congrats! You can’t go back to the apt but go to your house instead”

SET bought = true

CALL player’s addMoney()

SEND cost

CALL player’s setHouse()

SEND bought

ELSE

DISPLAY “Visit apt again to roll for another house”

**Test Plan 7:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Input Value | Driver Function | Expected Outcome | Actual Outcome |
| Apartment menu Displays when menu function is called | nwRm->menu() | List::move()  Apartment::menu() | “You are at your apartment….” | “You are at your apartment….” |
| Apartment rollHouseDie function randomly generates numbers | None | Apartment::rollHouseDie() | New house type appears each time | New house type appears each time |
| If person buys house, money removed from their account | ‘Y’ or ‘y’ | Apartment::rollHouseDie()  Person::addMoney() | “Player’s original money: 100 “  “player’s new money: -XXX” | “Player’s original money: 100 “  “player’s new money: -XXX” |
| If person buys house, their house bool is changed | ‘Y’ or ‘y’ | Apartment::rollHouseDie()  Person::addMoney() | “Player’s house: 0 “  “player’s house: 1” | “Player’s house: 0 “  “player’s house: 1” |
| If person does not buy house, their house bool is not changed | ‘N’ or ‘n’ | Apartment::rollHouseDie()  Person::addMoney() | “Player’s house: 0 “  “player’s house: 0” | “Player’s house: 0 “  “player’s house: 0” |
| If person does not buy house, their money is not changed | ‘N’ or ‘n’ | Apartment::rollHouseDie()  Person::addMoney() | “Player’s original money: 100 “  “player’s new money: 100” | “Player’s original money: 100 “  “player’s new money: 100” |

**Design/Implementation 8: Creating the School Space**

* Developed Menu
* Added in degree() which allows a player to earn their degree
* Added in physDeg() which allows a player to pick up the physical degree and keep it in their container
* Added in payLoans() which allows a player to pay back their loans incurred by going to school

*School.cpp file:*

Int Menu()

CREATE int input

CREATE double loans

DISPLAY “Please choose option:

1) get degree

2) pay student loans off

3) pick up physical degree

4) exit

SET input = cin

IF (input is 1)

CALL degree()

IF (input is 2)

CALL payLoans()

IF (input is 3)

CALL physDeg()

ELSE

DISPLAY “now leaving school”

Void degree()

CREATE char ans

CREATE bool answered

CREATE double result

CREATE int result2

CREATE new diceMod

CALL die1’s settSideCount()

SEND 3

CREATE array of char

SET equal to ‘b’, ‘a’, ‘a’

DISPLAY “Student Loans will be $500. This is added to your account”

CALL player1’s addStudentLoans()

SEND 500

SET result = die1’s rollDie()

SET result2 = static cast of result

SWITCH (result2)

CASES 1 through 3:

DISPLAY Question

SET ans = cin

IF (ans = array[0])

DISPLAY “Correct! “

SET answered = true

ELSE

DISPLAY “Sorry, that’s wrong”

SET answered = false

Break

IF (answered is true)

DISPLAY “Congrats! You earned your degree. Pick it up by choosing option 3 in school menu”

SEND answered to player1’s setDegree()

SEND answered to player1’s setPhysDeg()

ELSE

DISPLAY “You did not earn your degree. Please try again but note that another student loan will be charged to your account”

Void physDeg()

IF (player’s getPhysDeg() is true)

DISPLAY “your degree has been added to your container of items”

CALL player1’s addContainer()

SEND “Physical Degree”

ELSE

DISPLAY “you have not earned your degree yet”

Void payLoans()

CREATE double amount

CREATE double loans

SET loans = player1’s getStudentLoans()

IF (loans >= 0.01)

DISPLAY “you have && left to pay. Enter amount you want to pay: “

SET amount = cin

SET amount = (-1)\*amount

CALL player1’s addStudentLoans()

SEND amount

DISPLAY “thank you”

ELSE

DISPLAY “you do not have any loans”

**Test Plan 8:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Input Value | Driver Function | Expected Outcome | Actual Outcome |
| School menu Displays when menu function is called | nwRm->menu() | List::move()  School::menu() | “You are at school….” | “You are at school….” |
| Degree Menu calls roll die function | 1 | School::degree()  DiceMode::rollDie() | “You rolled a X” | “You rolled a X” |
| Degree recognizes correct answers | Q1: b | School::degree() | “Correct!” | “Correct!” |
| Correct answer results in degree | Q1: b | School::degree() | “Congrats!...” | “Congrats!...” |
| Wrong answer does not result in degree | Q1: c | School::degree() | “Sorry…” | “Sorry…” |
| physDeg recognizes if you’ve earned your degree | 3 | School::physDeg() | “Degree has been added…” | “Degree has been added…” |
| physDeg successfully adds degree to Person container | 3 | School::physDeg()  Person::getContainer() | “Item 0: Physical Degree” | “Item 0: Physical Degree” |
| payLoans displays correct balance | 2 | School::payLoans()  Person::getStudentLoans() | “you have X left to pay” | “you have X left to pay” |
| payLoans removes desired amount from student loans | 500 | School::payLoans()  Person::addStdntLns() | “500 has been discounted…” | “500 has been discounted…” |
| payLoans removes paid amount from bank account | 500 | School::payLoans()  Person::addMoney() | Money is 500 less | Money is 500 less |

**Design/Implementation 9: Retirement Community**

* Developed Menu
* Added in verify() which checks the person’s requirements to make sure they were met.
* Added in endGame() which ends the game if the player met requirements

Retire.cpp file:

Void Menu()

DISPLAY “Welcome. Let us make sure you’ve met our reqs”

CALL verify()

Void verify()

IF (player1’s getDegree() = false)

DISPLAY “You did not receive a degree. Go back.”

IF (player1’s getHouse() = false)

DISPLAY “You did not buy a house. Go back”

IF (player1’s getBonus() = false)

DISPLAY “You did not get a bonus. Go back”

ELSE IF (player1’s getMoney() < 1000)

DISPLAY “You do not have enough money. Go back”

ELSE

DISPLAY “Congrats! You meet the reqs”

CALL endGame()

Void endGame()

CALL player1’s setEnd()

SEND true

**Test Plan 9:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Input Value | Driver Function | Expected Outcome | Actual Outcome |
| Recognizes no house | House = false | Verify() | “you do not have a house” | “you do not have a house” |
| Recognizes no marriage | Marriage = false | Verify() | “you did not get married” | “you did not get married” |
| Recognizes no degree | Degree = false | Verify() | “you did not get a degree” | “you did not get a degree” |
| Recognizes no bonus | Bonus = false | Verify() | “you did not get a bonus” | “you did not get a bonus” |
| Recognizes when all flags are true | House = marriage = degree = bonus = true | Verify() | “all reqs met” | “all reqs met” |

**Design/Implementation 10: Creating House Space**

* Developed Menu()
* Added in Cook() which provides 3 different scenarios
* Added in newspaper() which allows you to pick up the newspaper to read.

House.cpp

Void Menu()

Do {

DISPLAY “choose between cooking, reading newspaper, leaving”

SET input = cin

IF (input is 1)

CALL cook()

IF (input is 2)

CALL newspaper()

} while (input is not 3)

Void cook()

CREATE new dice object

CALL dice’s setSideCount()

SEND 3

CREATE double result

CREATE double result2

SET result = die1’s rollDie()

SET result2 = static cast of result

SWITCH (result2)

CASE 1:

DISPLAY “spaghetti incident…cleaning supply…. Find $5”

CALL player1’s addMoney()

SEND 5

break

CASE 2:

DISPLAY “must order pizza. Pay $10”

CALL player1’s addMoney()

SEND -$10

Break

CASE 3:

DISPLAY “spaghetti cooking went great!”

Break

Void newspaper()

CREATE int input

CREATE string \*s = new string

INCREMENT visitCount

DISPLAY “Newspaper has been added to your container. Classified wants you to sell plasma for $300. Interested? Can only do every 4th house visit“

SET input = cin

SET s = player1’s getContainer()

FOR (int I = 0:1)

IF (s[i] = “newspaper”

DISPLAY “newspaper already in container”

ELSE

CALL player1’s addContainer()

SEND newspaper

IF (input = yes)

IF (visitCount <= 3)

DISPLAY “can only donate every 4th visit”

ELSE

DISPLAY “great! $300 added to account”

CALL player1’s addMoney()

SEND 300

SET visitCount = 0;

ELSE

DISPLAY “You have decided not to donate”

**Test Plan 10:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Input Value | Driver Function | Expected Outcome | Actual Outcome |
| Money added to account when donating plasma | Y | newspaper() | Money = +300 | Money = +300 |
| No plasma option returns to house menu | N | newspaper() | House menu displays | House menu displays |
| Cooking option randomly generates different scenarios | Cook() | Cook() | Diff scenarios appear | Diff scenarios appear |

**Design/Implementatioin 11: Lawyer Office**

* Developed Menu()
* Added Marriage() which marries the player and their significant other
* Added loanfight() which gives the player a chance to fight to erase their debt

*Lawyer.cpp file:*

Void Menu()

CREATE int input

DO

DISPLAY “enter 1 to get married” “enter 2 to fight to remove student loans” “enter 3 to leave”

SET inupt = cin

IF (input is 1)

CALL marriage()

IF (input is 2)

CALL loanfight()

WHILE (input is not 3)

Marriage()

DISPLAY “you are now married. Congrats!”

CALL player1’s setMarriage()

SEND true

loanFight()

DISPLAY “you will battle computer to remove loans. Must pay $50 to pay”

CALL player1’s addMoney()

SEND -50

SET you = new BlueMenYou

SET loanShark = new ReptilePeople

CREATE double attackRoll

CREATE bool dead

CREATE bool youDied

SET = false

CREATE double lifeStrength

DO

DISPLAY strength of each character by using character’s getStrength()

DISPLAY name of each character by using character’s getName()

CALL boss’s attack function

CALL your defense function

SEND boss’s attack value

GET your strength after attacks

CALL your dead function

IF (dead)

DISPLAY “You lost. Pay $$ for therapy. “

SET youdied = true

ELSE

CALL your attack function

CALL boss’s defense function

SEND your attack value

IF (dead)

DISPLAY “Boss lost the fight. You get raise!”

SET youdied = false

} WHILE (not dead)

IF (youdied)

DISPLAY “your loans are not erased”

ELSE

CREATE double m

DISPLAY “loans are gone”

SET m = player1’s getStudentLoans()

CALL player1’s setStudentLoans()

SEND m