**CodeReuse.h**

#pragma once

#include <iostream>

#include "Tree.h"

#include "Room.h"

#include "RoomBuilder.h"

#include "FireSlime.h"

#include "WindSlime.h"

#include "WaterSlime.h"

#include "LinkedList.h"

#include "Party.h"

#include "windows.h"

#include "GachaMachine.h"

#include <typeinfo>

class CodeReuse

{

public:

void Overview(Party<Slime> slimes) {

cout << endl << "Party Overview" << endl;

slimes.listMembers();

cout << endl << "Inventory Overview" << endl;

slimes.inventory.display();

}

// Setup World

TreeNode<DoublyLinkedList<Room>>\* WorldSetup() {

//pointer to queue of chests

Queue<Item>\* chests = new Queue<Item>();

//get the location of buried Aztec Gold

int aztecGoldLocation = rand() % 18;

//fill in Item to the queue and Aztec Gold based on location

for (int i = 0; i < 18; i++) {

if (i == aztecGoldLocation) {

Item aztecGold("Aztec Gold", "", 1);

chests->enqueue(aztecGold);

}

else {

Item gachaKey("Gacha Key", "", 1);

chests->enqueue(gachaKey);

}

}

RoomBuilder roomBuilder;

// declare an array of 9 doubly linked list.

// There will be 2 rooms in each doubly linked list.

DoublyLinkedList<Room> rooms[9];

//build room and insert into doubly linked list

for (int i = 0; i < 9; i++) {

rooms[i].append(roomBuilder.buildRoom(1, chests));

rooms[i].append(roomBuilder.buildRoom(2, chests));

}

// put doubly linked list into array

DoublyLinkedList<Room> arr[9] = { rooms[0], rooms[1], rooms[2], rooms[3], rooms[4], rooms[5], rooms[6], rooms[7], rooms[8] };

int n = sizeof(arr) / sizeof(arr[0]);

// tree generator

Tree<DoublyLinkedList<Room>> tree;

// pointer to the root of tree

TreeNode<DoublyLinkedList<Room>>\* root = new TreeNode<DoublyLinkedList<Room>>();

// build the tree

root = tree.insertLevelOrder(arr, root, 0, n, NULL);

return root;

}

// Setup Party

Party<Slime>\* PartySetup() {

Party<Slime>\* slimes = new Party<Slime>();

FireSlime\* fireSlime = new FireSlime(80, 80, 5);

WaterSlime\* waterSlime = new WaterSlime(120, 120, 3);

WindSlime\* windSlime = new WindSlime(100, 100, 4);

cout << "Who are YOU ?" << endl << endl;

cout << "(1) Fire Slime..." << endl;

cout << "(2) Water Slime..." << endl;

cout << "(3) Wind Slime..." << endl;

int charSelection;

cout << "Input: ";

cin >> charSelection;

Slime\* protagonistSlime = fireSlime;

if (charSelection == 1) {

protagonistSlime = fireSlime;

}

else if (charSelection == 2) {

protagonistSlime = waterSlime;

}

else if (charSelection == 3) {

protagonistSlime = windSlime;

}

cout << endl << protagonistSlime->name << " gain ";

protagonistSlime->specialLuck();

cout << endl << "Please tell me your name..." << endl;

if (charSelection == 1) {

cin >> fireSlime->name;

system("cls");

cout << fireSlime->name << "," << endl << " " << " Water Slime and Wind Slime will be joining you for this adventure..." << endl;

slimes->addMember(\*fireSlime);

slimes->addMember(\*waterSlime);

slimes->addMember(\*windSlime);

}

else if (charSelection == 2) {

cin >> waterSlime->name;

system("cls");

cout << waterSlime->name << "," << endl << " " << " Fire Slime and Wind Slime will be joining you for this adventure..." << endl;

slimes->addMember(\*waterSlime);

slimes->addMember(\*fireSlime);

slimes->addMember(\*windSlime);

}

else if (charSelection == 3) {

cin >> windSlime->name;

system("cls");

cout << windSlime->name << "," << endl << " " << "Fire Slime and Water Slime will be joining you for this adventure..." << endl;

slimes->addMember(\*windSlime);

slimes->addMember(\*waterSlime);

slimes->addMember(\*fireSlime);

}

Item slimeBanner("Slime Party Banner", "", 1);

slimes->inventory.append(slimeBanner);

Item smallPotion("Small Potion", "", 1);

slimes->inventory.append(smallPotion);

Item gachaKey("Gacha Key", "", 1);

slimes->inventory.append(gachaKey);

Overview(\*slimes);

cout << endl << "Let the Adventure Begins !!!" << endl;

cout << endl << "Press any key to continue..." << endl;

string placeHolder;

cin >> placeHolder;

return slimes;

}

// battle phase

boolean BattlePhase(Room\* currRoom, Party<Slime>\* slimes) {

boolean fight = false;

// if stack of monster is not empty

while (!currRoom->monsters.isEmpty()) {

fight = true;

// get the top element as monster to fight

Monster monster = currRoom->monsters.peek();

cout << endl << "Monster appeared !!!" << endl;

::Sleep(100);

cout << endl << monster.monsterDes << endl;

::Sleep(100);

cout << endl << "============Battle Start============" << endl;

::Sleep(100);

for (int j = 1; monster.hP >= 0; j++) {

cout << endl << "Round " << j << endl;

for (int i = 0; i < slimes->size(); i++) {

::Sleep(100);

Slime& slime = slimes->getMembers(i);

int damageReceived = 0;

if (slime.type == "Fire Slime") {

damageReceived = monster.fireElem;

}

else if (slime.type == "Water Slime") {

damageReceived = monster.waterElem;

}

else if (slime.type == "Wind Slime") {

damageReceived = monster.windElem;

}

if (slime.status != "Dead") {

monster.hP -= slime.attack;

cout << slime.name << " dealed " << slime.attack << " damage to Monster, received " << damageReceived << " damage." << endl;

slime.setHP(slime.getHP() - damageReceived);

}

}

if (slimes->getMembers(0).status == "Dead" && slimes->getMembers(1).status == "Dead" && slimes->getMembers(2).status == "Dead") {

cout << endl << "All dead... all dead... all the slimes are dead..." << endl;

cout << endl << " GAME OVER" << endl;

exit(0);

}

}

// battle finished, remove monster from stack by pop operation

currRoom->monsters.pop();

::Sleep(100);

cout << endl << "============Battle End============" << endl;

::Sleep(100);

cout << endl << "Party Overview" << endl;

slimes->listMembers();

// if the popped monster reincarnate, then push a new reincarnated monster into the stack, and it will be the next monster to fight

if (monster.reincarnate) {

::Sleep(100);

cout << endl << "Monster reincarnate... Prepare for another battle..." << endl;

MonsterBuilder monsterBuilder;

currRoom->addMonster(monsterBuilder.buildMonster());

}

::Sleep(100);

// victory if there are no monsters left

if (currRoom->monsters.isEmpty()) {

cout << endl << " ";

for (int i = 0; i < 3; i++) {

::Sleep(100);

cout << " VICTORY";

}

cout << " !!!" << endl;

}

else {

cout << endl << "Press any key to continue..." << endl;

string wait;

cin >> wait;

}

::Sleep(100);

}

return fight;

}

//Character Selection

Slime\* CharSelection(Party<Slime>\* slimes) {

slimes->listMembers();

cout << "Input: ";

int charSelection;

cin >> charSelection;

return &(slimes->getMembers(charSelection - 1));

}

//Use Item

void UseItem(Party<Slime>\* slimes) {

cout << endl << "List of Items: " << endl;

slimes->inventory.display();

cout << "Input: ";

int itemUseSelection;

cin >> itemUseSelection;

if (slimes->inventory.getValue(itemUseSelection - 1).name == "Small Potion") {

cout << endl << "Which character you want to use potion on ?" << endl;

Slime\* healSlime;

healSlime = CharSelection(slimes);

healSlime->setHP(healSlime->getHP() + 10);

slimes->inventory.getValue(itemUseSelection - 1).amount--;

cout << endl << "1 Small Potion used." << endl;

if (slimes->inventory.getValue(itemUseSelection - 1).amount == 0) {

slimes->inventory.deleteAtIndex(itemUseSelection - 1);

}

cout << endl << healSlime->name << " recovered 10 hp." << endl;

}

else if (slimes->inventory.getValue(itemUseSelection - 1).name == "Gacha Key") {

GachaAction(slimes);

}

}

//Gacha Action

void GachaAction(Party<Slime>\* slimes) {

bool haveGachaKey = false;

// check if there is a Gacha Key available

for (int i = 0; i < slimes->inventory.size; i++) {

string test = slimes->inventory.getValue(i).name;

if (slimes->inventory.getValue(i).name == "Gacha Key") {

slimes->inventory.getValue(i).amount--;

cout << endl << "1 Gacha Key used." << endl;

if (slimes->inventory.getValue(i).amount == 0) {

slimes->inventory.deleteAtIndex(i);

}

haveGachaKey = true;

break;

}

}

// if Gache Key available

if (haveGachaKey) {

cout << endl << "Which character you want to use lottery on ?" << endl;

Slime\* lotterySlime;

lotterySlime = CharSelection(slimes);

// throw a dice

cout << endl << "Throwing Dice.";

for (int i = 0; i < 10; i++) {

::Sleep(100);

cout << ".";

}

int diceNumber = lotterySlime->throwDice();

cout << "You got " << diceNumber << " !" << endl;

GachaMachine gM;

//get lottery

Lottery lottery = gM.generateLottery();

// open number of slots based on dice number

for (int i = 1; i <= diceNumber; lottery.nextSlot(), i++) {

::Sleep(200);

string lotteryResult = lottery.getSlot();

// increase HP if slot prize is HP

if (lotteryResult == "HP") {

int amount = 1;

lotterySlime->setHP(lotterySlime->getHP() + amount);

lotterySlime->maxHP++;

cout << "(" << i << ") " << lottery.getSlot() << ": " << "Increased HP of " << lotterySlime->name << " by " << amount << endl;

}

// increase Attack if slot prize is Attack

else if (lotteryResult == "Attack") {

int amount = 1;

lotterySlime->attack++;

cout << "(" << i << ") " << lottery.getSlot() << ": " << "Increased Attack of " << lotterySlime->name << " by " << amount << endl;

}

// put small potion into inventory

else if (lotteryResult == "Small Potion") {

bool found = false;

int inventorySize = slimes->inventory.size;

for (int i = 0; i < inventorySize; i++) {

if (slimes->inventory.getValue(i).name == lotteryResult) {

slimes->inventory.getValue(i).amount++;

found = true;

break;

}

}

if (!found) {

Item smallPotion(lotteryResult, "", 1);

slimes->inventory.append(smallPotion);

}

cout << "(" << i << ") " << lottery.getSlot() << ": " << "Obtain 1 " << lotteryResult << "." << endl;

}

}

}

else {

cout << "No Gacha Key found." << endl;

}

::Sleep(100);

}

// Move Action

void MoveAction(TreeNode<DoublyLinkedList<Room>>\*\* currNode, DoublyLinkedNode<Room>\*\* currRoom) {

cout << endl << "Move direction available: " << endl;

// if there is next room, print front

if ((\*currRoom)->next) {

cout << " W(Front)" << endl;

}

// if there is left child node and the current room is the first room in the current node, print left

if ((\*currNode)->left && (\*currRoom)->data.id == 1) {

cout << " A(Left)" << endl;

}

// if there is parent node and the current room is the first room in the current node, print back

if (!((\*currNode)->parent == NULL && (\*currRoom)->data.id == 1)) {

cout << " S(Back)" << endl;

}

// if there is right child node and the current room is the first room in the current node, print right

if ((\*currNode)->right && (\*currRoom)->data.id == 1) {

cout << " D(Right)" << endl;

}

string movement;

cout << "Input: ";

cin >> movement;

// if player decided to move back

if (movement == "S") {

// if current room is the first room in current node and there is parent node for current node, then move to parent node and set current room as first room in parent node.

if ((\*currRoom)->data.id == 1 && (\*currNode)->parent) {

\*currNode = (\*currNode)->parent;

\*currRoom = (\*currNode)->data.getNode(0);

}

else {

// otherwise move to previous room in doubly linked list

\*currRoom = (\*currRoom)->prev;

}

}

// if there is next room for current room in doubly linked list

else if (movement == "W") {

\*currRoom = (\*currRoom)->next;

}

// the player select move right

else if (movement == "D") {

// if there is right child node and the current room is first room in current node, then move to right child node and set current room as first room in parent node.

if ((\*currNode)->right && (\*currRoom)->data.id == 1) {

\*currNode = (\*currNode)->right;

\*currRoom = (\*currNode)->data.getNode(0);

}

}

// the player select move left

else if (movement == "A") {

// if there is left child node and the current room is first room in current node, then move to left child node and set current room as first room in parent node.

if ((\*currNode)->left && (\*currRoom)->data.id == 1) {

\*currNode = (\*currNode)->left;

\*currRoom = (\*currNode)->data.getNode(0);

}

}

}

};

**DoublyLinkedList.h**

#pragma once

#include <iostream>

#include "DoublyLinkedNode.h"

template<class DataType>

class DoublyLinkedList

{

public:

int size = 0;

DoublyLinkedNode<DataType>\* head = new DoublyLinkedNode<DataType>();

DoublyLinkedNode<DataType>\* tail = new DoublyLinkedNode<DataType>();

DoublyLinkedList() {}

// copy constructor

DoublyLinkedList(const DoublyLinkedList& src) {

DoublyLinkedNode<DataType>\* node = src.head;

while (node->next != NULL)

{

node = node->next;

append(node->data);

}

}

// assignment operator

DoublyLinkedList& operator=(DoublyLinkedList src)

{

swap(head, src.head);

swap(tail, src.tail);

size = src.size;

return \*this;

}

// destructor

~DoublyLinkedList()

{

DoublyLinkedNode<DataType>\* curr = head;

while (curr != NULL) {

DoublyLinkedNode<DataType>\* next = curr->next;

delete curr;

curr = next;

}

head = NULL;

}

// add at the start

void prepend(DataType val)

{

DoublyLinkedNode<DataType>\* node = new DoublyLinkedNode<DataType>(val);

if (head->next == NULL && tail->next == NULL) {

head->next = node;

tail->next = node;

node->prev = head;

}

else {

node->prev = head;

node->next = head->next;

head->next = node;

}

size++;

}

// add at the end

void append(DataType val)

{

DoublyLinkedNode<DataType>\* node = new DoublyLinkedNode<DataType>(val);

if (head->next == NULL && tail->next == NULL) {

head->next = node;

tail->next = node;

node->prev = head;

}

else {

node->prev = tail->next;

tail->next->next = node;

tail->next = node;

}

size++;

}

/\* Get the value of the index-th node in the linked list. If the index is invalid, throw error. \*/

DataType& get(int index) {

if (index > size) throw "Index Invalid";

DoublyLinkedNode<DataType>\* temp = head;

for (int i = 0; i <= index; i++) temp = temp->next;

return temp->val;

}

DoublyLinkedNode<DataType>\* getNode(int index) {

if (index > size) throw "Index Invalid";

DoublyLinkedNode<DataType>\* temp = head;

for (int i = 0; i <= index; i++) temp = temp->next;

return temp;

}

//should be correct

/\*\* Add a node of value val before the index-th node in the linked list. If index equals to the length of linked list, the node will be appended to the end of linked list. If index is greater than the length, the node will not be inserted. \*/

void addAtIndex(int index, DataType val) {

if (index > size) return;

DoublyLinkedNode<DataType>\* temp = head;

for (int i = 0; i < index; i++) {

temp = temp->next;

}

DoublyLinkedNode<DataType>\* node = new DoublyLinkedNode<DataType>(val);

if (head->next == NULL && tail->next == NULL) {

head->next = node;

tail->next = node;

node->prev = head;

}

else {

node->next = temp->next;

temp->next = node;

node->prev = temp;

}

size++;

}

//should be correct

/\*\* Delete the index-th node in the linked list, if the index is valid. \*/

void deleteAtIndex(int index) {

if (index >= size) return;

DoublyLinkedNode< DataType>\* temp = head;

for (int i = 0; i < index; i++) {

temp = temp->next;

}

DoublyLinkedNode< DataType>\* deletedNode = temp->next;

temp->next = deletedNode->next;

deletedNode->next->prev = temp;

size--;

delete deletedNode;

}

void display() {

DoublyLinkedNode<DataType>\* node = head;

while (node != NULL)

{

cout << node->data << " ";

node = node->next;

}

}

};

**DoublyLinkedNode.h**

#pragma once

#include <iostream>

using namespace std;

template<class DataType>

class DoublyLinkedNode

{

public:

DataType data;

DoublyLinkedNode<DataType>\* next;

DoublyLinkedNode<DataType>\* prev;

DoublyLinkedNode(DataType val) {

this->data = val;

next = NULL;

prev = NULL;

}

DoublyLinkedNode() {

next = NULL;

prev = NULL;

}

};

**FireSlime.h**

#pragma once

#include "Slime.h"

class FireSlime :

public Slime

{

public:

FireSlime(int MaxHP, int HP, int attack, string Name = "Fire Slime") : Slime(MaxHP, HP, attack, "Fire Slime", Name) {};

void specialLuck() {

attack = attack \* 2;

cout << "Enhanced Attack by 2x" << endl;

}

};

**GachaMachine.h**

#pragma once

#include <iostream>

#include <stdlib.h> /\* srand, rand \*/

#include "Iterator.h"

#include "Lottery.h"

using namespace std;

class GachaMachine

{

public:

GachaMachine() {

}

Lottery generateLottery() {

return Lottery();

}

};

**Item.h**

#pragma once

#include <string>

#include <iostream>

using namespace std;

class Item

{

public:

string name;

string description;

int amount;

Item() {};

Item(string Name, string Description, int Amount) :

name(Name), description(Description), amount(Amount) {};

friend ostream& operator<<(ostream& aOstream, Item item) {

aOstream << item.name << ": " << item.amount << endl;

return aOstream;

}

};

**Iterator.h**

#pragma once

#include <iostream>

using namespace std;

class Iterator

{

public:

const int fLength;

int fIndex;

const string\* arr;

Iterator(const std::string\* aArray, const int aLength, int aStart = 0) : arr(aArray), fLength(aLength), fIndex(aStart) {}

int getfIndex() const { return fIndex; }

Iterator& operator++() {

fIndex++;

return \*this;

}

Iterator operator++(int) {

Iterator temp = \*this;

fIndex++;

return temp;

}

Iterator& operator--() {

fIndex--;

return \*this;

}

Iterator operator--(int) {

Iterator temp = \*this;

fIndex--;

return temp;

}

const std::string& operator\*() const {

return arr[fIndex];

}

bool operator==(const Iterator& aOther) const {

return (fIndex == aOther.getfIndex()) && (arr == aOther.arr);

}

Iterator begin() const {

return Iterator(arr, fLength);

}

Iterator end() const {

return Iterator(arr, fLength, fLength);

}

};

**LinkedList.h**

#pragma once

#include "LinkedListNode.h"

#include "windows.h"

template <class DataType>

class LinkedList

{

public:

/\*\* Initialize your data structure here. \*/

int size = 0;

LinkedListNode<DataType>\* head = new LinkedListNode<DataType>();

LinkedListNode<DataType>\* tail = new LinkedListNode<DataType>();

LinkedList() {}

// copy constructor

LinkedList(const LinkedList& src) {

LinkedListNode<DataType>\* node = src.head;

while (node->next != NULL)

{

node = node->next;

append(node->val);

}

}

// assignment operator

LinkedList& operator=(LinkedList src)

{

swap(head, src.head);

size = src.size;

return \*this;

}

//destructor

~LinkedList()

{

LinkedListNode<DataType>\* curr = head;

while (curr != NULL) {

LinkedListNode<DataType>\* next = curr->next;

delete curr;

curr = next;

}

head = NULL;

}

/\*\* Get the value of the index-th node in the linked list. If the index is invalid, throw error. \*/

DataType& getValue(int index) {

if (index >= size) throw "Index Invalid";

LinkedListNode<DataType>\* temp = head;

for (int i = 0; i <= index; i++) temp = temp->next;

return temp->val;

}

/\*\* Get the value of the index-th node in the linked list. If the index is invalid, throw error. \*/

DataType get(int index) {

if (index >= size) throw "Index Invalid";

LinkedListNode<DataType>\* temp = head;

for (int i = 0; i <= index; i++) temp = temp->next;

return temp->val;

}

//correct

/\*\* Add a node of value val before the first element of the linked list. After the insertion, the new node will be the first node of the linked list. \*/

void prepend(DataType val) {

LinkedListNode<DataType>\* node = new LinkedListNode<DataType>(val);

if (head->next == NULL && tail->next == NULL) {

head->next = node;

tail->next = node;

}

else {

node->next = head->next;

head->next = node;

}

size++;

}

//correct

/\*\* Append a node of value val to the last element of the linked list. \*/

void append(DataType val) {

LinkedListNode<DataType>\* node = new LinkedListNode<DataType>(val);

if (head->next == NULL && tail->next == NULL) {

head->next = node;

tail->next = node;

}

else {

tail->next->next = node;

tail->next = node;

}

size++;

}

void display() {

LinkedListNode<DataType>\* temp = head;

int counter = 0;

while (temp->next != NULL) {

Sleep(100);

counter++;

temp = temp->next;

//will call the overloaded operator of the data type.

cout << "(" << counter << ")" << temp->val ;

}

}

//should be correct

/\*\* Add a node of value val before the index-th node in the linked list. If index equals to the length of linked list, the node will be appended to the end of linked list. If index is greater than the length, the node will not be inserted. \*/

void addAtIndex(int index, DataType val) {

if (index > size) return;

LinkedListNode<DataType>\* temp = head;

for (int i = 0; i < index; i++) {

temp = temp->next;

}

LinkedListNode<DataType>\* node = new LinkedListNode<DataType>(val);

if (head->next == NULL && tail->next == NULL) {

head->next = node;

tail->next = node;

}

else {

node->next = temp->next;

temp->next = node;

}

size++;

}

//correct

/\*\* Delete the index-th node in the linked list, if the index is valid. \*/

void deleteAtIndex(int index) {

if (index >= size) return;

LinkedListNode< DataType>\* temp = head;

for (int i = 0; i < index; i++) {

temp = temp->next;

}

LinkedListNode< DataType>\* deletedNode = temp->next;

temp->next = deletedNode->next;

size--;

delete deletedNode;

}

};

**LinkedListNode.h**

#pragma once

#include "LinkedListNode.h"

#include "windows.h"

template <class DataType>

class LinkedList

{

public:

/\*\* Initialize your data structure here. \*/

int size = 0;

LinkedListNode<DataType>\* head = new LinkedListNode<DataType>();

LinkedListNode<DataType>\* tail = new LinkedListNode<DataType>();

LinkedList() {}

// copy constructor

LinkedList(const LinkedList& src) {

LinkedListNode<DataType>\* node = src.head;

while (node->next != NULL)

{

node = node->next;

append(node->val);

}

}

// assignment operator

LinkedList& operator=(LinkedList src)

{

swap(head, src.head);

size = src.size;

return \*this;

}

//destructor

~LinkedList()

{

LinkedListNode<DataType>\* curr = head;

while (curr != NULL) {

LinkedListNode<DataType>\* next = curr->next;

delete curr;

curr = next;

}

head = NULL;

}

/\*\* Get the value of the index-th node in the linked list. If the index is invalid, throw error. \*/

DataType& getValue(int index) {

if (index >= size) throw "Index Invalid";

LinkedListNode<DataType>\* temp = head;

for (int i = 0; i <= index; i++) temp = temp->next;

return temp->val;

}

/\*\* Get the value of the index-th node in the linked list. If the index is invalid, throw error. \*/

DataType get(int index) {

if (index >= size) throw "Index Invalid";

LinkedListNode<DataType>\* temp = head;

for (int i = 0; i <= index; i++) temp = temp->next;

return temp->val;

}

//correct

/\*\* Add a node of value val before the first element of the linked list. After the insertion, the new node will be the first node of the linked list. \*/

void prepend(DataType val) {

LinkedListNode<DataType>\* node = new LinkedListNode<DataType>(val);

if (head->next == NULL && tail->next == NULL) {

head->next = node;

tail->next = node;

}

else {

node->next = head->next;

head->next = node;

}

size++;

}

//correct

/\*\* Append a node of value val to the last element of the linked list. \*/

void append(DataType val) {

LinkedListNode<DataType>\* node = new LinkedListNode<DataType>(val);

if (head->next == NULL && tail->next == NULL) {

head->next = node;

tail->next = node;

}

else {

tail->next->next = node;

tail->next = node;

}

size++;

}

void display() {

LinkedListNode<DataType>\* temp = head;

int counter = 0;

while (temp->next != NULL) {

Sleep(100);

counter++;

temp = temp->next;

//will call the overloaded operator of the data type.

cout << "(" << counter << ")" << temp->val ;

}

}

//should be correct

/\*\* Add a node of value val before the index-th node in the linked list. If index equals to the length of linked list, the node will be appended to the end of linked list. If index is greater than the length, the node will not be inserted. \*/

void addAtIndex(int index, DataType val) {

if (index > size) return;

LinkedListNode<DataType>\* temp = head;

for (int i = 0; i < index; i++) {

temp = temp->next;

}

LinkedListNode<DataType>\* node = new LinkedListNode<DataType>(val);

if (head->next == NULL && tail->next == NULL) {

head->next = node;

tail->next = node;

}

else {

node->next = temp->next;

temp->next = node;

}

size++;

}

//correct

/\*\* Delete the index-th node in the linked list, if the index is valid. \*/

void deleteAtIndex(int index) {

if (index >= size) return;

LinkedListNode< DataType>\* temp = head;

for (int i = 0; i < index; i++) {

temp = temp->next;

}

LinkedListNode< DataType>\* deletedNode = temp->next;

temp->next = deletedNode->next;

size--;

delete deletedNode;

}

};

**Lottery.h**

#pragma once

#include "LinkedListNode.h"

#include "windows.h"

template <class DataType>

class LinkedList

{

public:

/\*\* Initialize your data structure here. \*/

int size = 0;

LinkedListNode<DataType>\* head = new LinkedListNode<DataType>();

LinkedListNode<DataType>\* tail = new LinkedListNode<DataType>();

LinkedList() {}

// copy constructor

LinkedList(const LinkedList& src) {

LinkedListNode<DataType>\* node = src.head;

while (node->next != NULL)

{

node = node->next;

append(node->val);

}

}

// assignment operator

LinkedList& operator=(LinkedList src)

{

swap(head, src.head);

size = src.size;

return \*this;

}

//destructor

~LinkedList()

{

LinkedListNode<DataType>\* curr = head;

while (curr != NULL) {

LinkedListNode<DataType>\* next = curr->next;

delete curr;

curr = next;

}

head = NULL;

}

/\*\* Get the value of the index-th node in the linked list. If the index is invalid, throw error. \*/

DataType& getValue(int index) {

if (index >= size) throw "Index Invalid";

LinkedListNode<DataType>\* temp = head;

for (int i = 0; i <= index; i++) temp = temp->next;

return temp->val;

}

/\*\* Get the value of the index-th node in the linked list. If the index is invalid, throw error. \*/

DataType get(int index) {

if (index >= size) throw "Index Invalid";

LinkedListNode<DataType>\* temp = head;

for (int i = 0; i <= index; i++) temp = temp->next;

return temp->val;

}

//correct

/\*\* Add a node of value val before the first element of the linked list. After the insertion, the new node will be the first node of the linked list. \*/

void prepend(DataType val) {

LinkedListNode<DataType>\* node = new LinkedListNode<DataType>(val);

if (head->next == NULL && tail->next == NULL) {

head->next = node;

tail->next = node;

}

else {

node->next = head->next;

head->next = node;

}

size++;

}

//correct

/\*\* Append a node of value val to the last element of the linked list. \*/

void append(DataType val) {

LinkedListNode<DataType>\* node = new LinkedListNode<DataType>(val);

if (head->next == NULL && tail->next == NULL) {

head->next = node;

tail->next = node;

}

else {

tail->next->next = node;

tail->next = node;

}

size++;

}

void display() {

LinkedListNode<DataType>\* temp = head;

int counter = 0;

while (temp->next != NULL) {

Sleep(100);

counter++;

temp = temp->next;

//will call the overloaded operator of the data type.

cout << "(" << counter << ")" << temp->val ;

}

}

//should be correct

/\*\* Add a node of value val before the index-th node in the linked list. If index equals to the length of linked list, the node will be appended to the end of linked list. If index is greater than the length, the node will not be inserted. \*/

void addAtIndex(int index, DataType val) {

if (index > size) return;

LinkedListNode<DataType>\* temp = head;

for (int i = 0; i < index; i++) {

temp = temp->next;

}

LinkedListNode<DataType>\* node = new LinkedListNode<DataType>(val);

if (head->next == NULL && tail->next == NULL) {

head->next = node;

tail->next = node;

}

else {

node->next = temp->next;

temp->next = node;

}

size++;

}

//correct

/\*\* Delete the index-th node in the linked list, if the index is valid. \*/

void deleteAtIndex(int index) {

if (index >= size) return;

LinkedListNode< DataType>\* temp = head;

for (int i = 0; i < index; i++) {

temp = temp->next;

}

LinkedListNode< DataType>\* deletedNode = temp->next;

temp->next = deletedNode->next;

size--;

delete deletedNode;

}

};

**Monster.h**

#pragma once

#include "LinkedListNode.h"

#include "windows.h"

template <class DataType>

class LinkedList

{

public:

/\*\* Initialize your data structure here. \*/

int size = 0;

LinkedListNode<DataType>\* head = new LinkedListNode<DataType>();

LinkedListNode<DataType>\* tail = new LinkedListNode<DataType>();

LinkedList() {}

// copy constructor

LinkedList(const LinkedList& src) {

LinkedListNode<DataType>\* node = src.head;

while (node->next != NULL)

{

node = node->next;

append(node->val);

}

}

// assignment operator

LinkedList& operator=(LinkedList src)

{

swap(head, src.head);

size = src.size;

return \*this;

}

//destructor

~LinkedList()

{

LinkedListNode<DataType>\* curr = head;

while (curr != NULL) {

LinkedListNode<DataType>\* next = curr->next;

delete curr;

curr = next;

}

head = NULL;

}

/\*\* Get the value of the index-th node in the linked list. If the index is invalid, throw error. \*/

DataType& getValue(int index) {

if (index >= size) throw "Index Invalid";

LinkedListNode<DataType>\* temp = head;

for (int i = 0; i <= index; i++) temp = temp->next;

return temp->val;

}

/\*\* Get the value of the index-th node in the linked list. If the index is invalid, throw error. \*/

DataType get(int index) {

if (index >= size) throw "Index Invalid";

LinkedListNode<DataType>\* temp = head;

for (int i = 0; i <= index; i++) temp = temp->next;

return temp->val;

}

//correct

/\*\* Add a node of value val before the first element of the linked list. After the insertion, the new node will be the first node of the linked list. \*/

void prepend(DataType val) {

LinkedListNode<DataType>\* node = new LinkedListNode<DataType>(val);

if (head->next == NULL && tail->next == NULL) {

head->next = node;

tail->next = node;

}

else {

node->next = head->next;

head->next = node;

}

size++;

}

//correct

/\*\* Append a node of value val to the last element of the linked list. \*/

void append(DataType val) {

LinkedListNode<DataType>\* node = new LinkedListNode<DataType>(val);

if (head->next == NULL && tail->next == NULL) {

head->next = node;

tail->next = node;

}

else {

tail->next->next = node;

tail->next = node;

}

size++;

}

void display() {

LinkedListNode<DataType>\* temp = head;

int counter = 0;

while (temp->next != NULL) {

Sleep(100);

counter++;

temp = temp->next;

//will call the overloaded operator of the data type.

cout << "(" << counter << ")" << temp->val ;

}

}

//should be correct

/\*\* Add a node of value val before the index-th node in the linked list. If index equals to the length of linked list, the node will be appended to the end of linked list. If index is greater than the length, the node will not be inserted. \*/

void addAtIndex(int index, DataType val) {

if (index > size) return;

LinkedListNode<DataType>\* temp = head;

for (int i = 0; i < index; i++) {

temp = temp->next;

}

LinkedListNode<DataType>\* node = new LinkedListNode<DataType>(val);

if (head->next == NULL && tail->next == NULL) {

head->next = node;

tail->next = node;

}

else {

node->next = temp->next;

temp->next = node;

}

size++;

}

//correct

/\*\* Delete the index-th node in the linked list, if the index is valid. \*/

void deleteAtIndex(int index) {

if (index >= size) return;

LinkedListNode< DataType>\* temp = head;

for (int i = 0; i < index; i++) {

temp = temp->next;

}

LinkedListNode< DataType>\* deletedNode = temp->next;

temp->next = deletedNode->next;

size--;

delete deletedNode;

}

};

**MonsterBuilder.h**

#pragma once

#include <stdlib.h> /\* srand, rand \*/

#include <time.h> /\* time \*/

#include "Monster.h"

#include <string>

#include <iostream>

using namespace std;

class MonsterBuilder

{

public:

string monsterDes[15] = {

"This bulky feline monster lairs in icy climes. It lures its prey, which includes mundane beasts, monstrous humanoids, other predators, and small creatures. It attacks with a tail striker, venom and debilitating effects. Legend holds that they were created by an experiment gone wrong.",

"This towering, insect-like, ethereal monster can be found in marshes. It attacks with spikes, noxious fumes and cold. It fears a certain metal.",

"This medium-sized, unnatural, bird-like beast makes its home in deserts. It tracks its prey, which includes monstrous humanoids and mundane beasts. It attacks with spines, psionics and creeping darkness. They travel in tribes of 5-16. Some legends say that they have a hidden, primitive civilization.",

"This towering chimeric beast lives in deep lakes. It attacks with spines, a hypnotic song and creeping darkness. Folktales say that various body parts are much sought after as ritual components.",

"This large fiendish monster dwells in underground caverns. It stalks its prey, which includes medium-sized creatures, small creatures, and other predators. It attacks with dizzying blows, electricity, projectile weapons and necrotic energy. They live in packs of 1-5.",

"This small insect-like beast dwells in coastal areas. It attacks with a tail striker, an eldritch aura and thrown weapons. It dislikes bright light. They lair in groups of 3-9. According to myth, they cooperate with other monsters.",

"This dog-sized chimeric beast makes its home in mountains. It lies in wait for its prey, which includes other predators, large creatures, magical beasts, and mundane beasts. It attacks with spikes, acid, projectile weapons and obscuring fog. It is weak against ash. Rumor has it that they do most of their hunting at twilight.",

"This runty reptilian creature dwells in mountains. It lies in wait for its prey, which includes other monsters of the same type, humans, and mundane beasts. It attacks with slashing claws, debilitating effects and psionics. They lair in flocks of 3-10. According to folktales, various body parts are much sought after as ritual components.",

"This bear-sized bird-like beast lives in city sewers. It leaps upon its prey, which includes other predators, medium-sized creatures, other monsters of the same type, and monstrous humanoids. It attacks with piercing claws, obscuring fog and necrotic energy.",

"This human-sized snake-like beast lairs in swamps. It lies in wait for its prey, which includes other predators, mundane beasts, large creatures, and other monsters of the same type. It attacks with rapid blows, toxic bites, fire and necrotic energy. It dislikes certain symbols. According to myth, they keep hordes of treasure.",

"This hulking oozing monster can be found in deserts. It attacks with crushing blows and toxic bites. It dislikes certain herbs. They hunt alone. Rumor has it that they keep trophies from slain prey.",

"This horse-sized, phantasmal, canine monster can be found in city sewers. It stalks its prey, which includes humans, other predators, and monstrous humanoids. It attacks with piercing claws, obscuring fog and entangling webs.",

"This horse-sized, mongrel, bird-like monster lives in untamed grasslands. It lies in wait for its prey, which includes large creatures and mundane beasts. It attacks with a tail striker and a hypnotic gaze.",

"This towering, fiendish, phantasmal beast lives in deep forests. It stalks its prey, which includes other monsters of the same type and mundane beasts. It attacks with piercing claws, choking smoke and grasping tentacles. It is difficult to harm without bright light. They hunt in bands of 5-12.",

"This medium-sized mongrel beast lives in wooded areas. It lures its prey, which includes other predators, mundane beasts, medium-sized creatures, and other monsters of the same type. It attacks with crushing blows, projectile weapons and sound. They live in tribes of 2-5."

};

Monster buildMonster() {

int startingHP = rand() % 20 + 15;

bool reincarnate = false;

if (rand() % 10 == 9) {

reincarnate = true;

}

Monster monster(startingHP, startingHP, rand() % 5, rand() % 5, rand() % 5, monsterDes[rand() % 15], reincarnate);

return monster;

}

};

**Party.h**

#pragma once

#include "Slime.h"

template <class DataType>

class Party

{

public:

LinkedList<DataType> slimes;

LinkedList<Item> inventory;

Party() {}

void addMember(DataType newMember) {

slimes.append(newMember);

}

void listMembers() {

slimes.display();

}

DataType& getMembers(int index) {

return slimes.getValue(index);

}

int size() {

return slimes.size;

}

};

**Queue.h**

#pragma once

#include "LinkedList.h"

template<class DataType>

class Queue

{

private:

LinkedList<DataType> fElements;

public:

bool isEmpty() {

return fElements.size == 0;

}

int size() {

return fElements.size();

}

void enqueue(DataType val) {

fElements.append(val);

}

DataType dequeue() {

if (!isEmpty()) {

DataType removed = fElements.get(0);

fElements.deleteAtIndex(0);

return removed;

}

else

throw std::underflow\_error("Queue is empty!");

}

};

**Room.h**

#pragma once

#include "Stack.h"

#include "Monster.h"

#include "DoublyLinkedList.h"

#include "DoublyLinkedNode.h"

#include "LinkedList.h"

#include "Item.h"

#include "Queue.h"

class Room

{

public:

int id;

string roomIntro;

Stack<Monster> monsters;

boolean chest = true;

Queue<Item>\* chests;

Room() {};

Room(int ID, string RoomIntro, Queue<Item>\* Chests): id(ID), roomIntro(RoomIntro), chests(Chests) {

};

friend ostream& operator<<(ostream& aOstream, Room room) {

aOstream << room.roomIntro << endl;

return aOstream;

}

Item openChest() {

chest = false;

return chests->dequeue();

}

void addMonster(Monster monster) {

monsters.push(monster);

}

~Room() {};

};

**RoomBuilder.h**

#pragma once

#include <stdlib.h> /\* srand, rand \*/

#include <time.h> /\* time \*/

#include "Room.h"

#include "MonsterBuilder.h"

#include <string>

#include <iostream>

using namespace std;

class RoomBuilder

{

public:

MonsterBuilder monsterBuilder;

RoomBuilder() {

/\* initialize random seed: \*/

srand(time(0));

};

string roomDesc[20] = {

"A crack in the ceiling above the middle of the north wall allows a trickle of water to flow down to the floor. The water pools near the base of the wall, and a rivulet runs along the wall an out into the hall. The water smells fresh.",

"Thick cobwebs fill the corners of the room, and wisps of webbing hang from the ceiling and waver in a wind you can barely feel. One corner of the ceiling has a particularly large clot of webbing within which a goblin's bones are tangled.",

"Tapestries decorate the walls of this room. Although they may once have been brilliant in hue, they now hang in graying tatters. Despite the damage of time and neglect, you can perceive once-grand images of wizards' towers, magical beasts, and symbols of spellcasting. The tapestry that is in the best condition bulges out weirdly, as though someone stands behind it (an armless statue of a female human spellcaster).",

"Rats inside the room shriek when they hear the door open, then they run in all directions from a putrid corpse lying in the center of the floor. As these creatures crowd around the edges of the room, seeking to crawl through a hole in one corner, they fight one another. The stinking corpse in the middle of the room looks human, but the damage both time and the rats have wrought are enough to make determining its race by appearance an extremely difficult task at best.",

"Neither light nor darkvision can penetrate the gloom in this chamber. An unnatural shade fills it, and the room's farthest reaches are barely visible. Near the room's center, you can just barely perceive a lump about the size of a human lying on the floor. (It might be a dead body, a pile of rags, or a sleeping monster that can take advantage of the room's darkness.)",

"Burning torches in iron sconces line the walls of this room, lighting it brilliantly. At the room's center lies a squat stone altar, its top covered in recently spilled blood. A channel in the altar funnels the blood down its side to the floor where it fills grooves in the floor that trace some kind of pattern or symbol around the altar. Unfortunately, you can't tell what it is from your vantage point.",

"A liquid-filled pit extends to every wall of this chamber. The liquid lies about 10 feet below your feet and is so murky that you can't see its bottom. The room smells sour. A rope bridge extends from your door to the room's other exit.",

"Fire crackles and pops in a small cooking fire set in the center of the room. The smoke from a burning rat on a spit curls up through a hole in the ceiling. Around the fire lie several fur blankets and a bag. It looks like someone camped here until not long ago, but then left in a hurry.",

"A flurry of bats suddenly flaps through the doorway, their screeching barely audible as they careen past your heads. They flap past you into the rooms and halls beyond. The room from which they came seems barren at first glance.",

"Rusting spikes line the walls and ceiling of this chamber. The dusty floor shows no sign that the walls move over it, but you can see the skeleton of some humanoid impaled on some wall spikes nearby.",

"You open the door, and the reek of garbage assaults your nose. Looking inside, you see a pile of refuse and offal that nearly reaches the ceiling. In the ceiling above it is a small hole that is roughly as wide as two human hands. No doubt some city dweller high above disposes of his rubbish without ever thinking about where it goes.",

"You open the door, and the room comes alive with light and music. A sourceless, warm glow suffuses the chamber, and a harp you cannot see plays soothing sounds. Unfortunately, the rest of the chamber isn't so inviting. The floor is strewn with the smashed remains of rotting furniture. It looks like the room once held a bed, a desk, a chest, and a chair.",

"A skeleton dressed in moth-eaten garb lies before a large open chest in the rear of this chamber. The chest is empty, but you note two needles projecting from the now-open lock. Dust coats something sticky on the needles' points.",

"Rounded green stones set in the floor form a snake's head that points in the direction of the doorway you stand in. The body of the snake flows back and toward the wall to go round about the room in ever smaller circles, creating a spiral pattern on the floor. Similar green-stone snakes wend along the walls, seemingly at random heights, and their long bodies make wave shapes.",

"The manacles set into the walls of this room give you the distinct impression that it was used as a prison and torture chamber, although you can see no evidence of torture devices. One particularly large set of manacles -- big enough for an ogre -- have been broken open.",

"You gaze into the room and hundreds of skulls gaze coldly back at you. They're set in niches in the walls in a checkerboard pattern, each skull bearing a half-melted candle on its head. The grinning bones stare vacantly into the room, which otherwise seems empty.",

"Unlike the flagstone common throughout the dungeon, this room is walled and floored with black marble veined with white. The ceiling is similarly marbled, but the thick pillars that hold it up are white. A brown stain drips down one side of a nearby pillar.",

"A huge iron cage lies on its side in this room, and its gate rests open on the floor. A broken chain lies under the door, and the cage is on a rotting corpse that looks to be a hobgoblin. Another corpse lies a short distance away from the cage. It lacks a head.",

"This room is a tomb. Stone sarcophagi stand in five rows of three, each carved with the visage of a warrior lying in state. In their center, one sarcophagus stands taller than the rest. Held up by six squat pillars, its stone bears the carving of a beautiful woman who seems more asleep than dead. The carving of the warriors is skillful but seems perfunctory compared to the love a sculptor must have lavished upon the lifelike carving of the woman.",

"A dim bluish light suffuses this chamber, its source obvious at a glance. Blue-glowing lichen and violet-glowing moss cling to the ceiling and spread across the floor. It even creeps down and up each wall, as if the colonies on the floor and ceiling are growing to meet each other. Their source seems to be a glowing, narrow crack in the ceiling, the extent of which you cannot gauge from your position. The air in the room smells fresh and damp." };

Room buildRoom(int id, Queue<Item>\* chests) {

Room room(id, roomDesc[rand() % 20], chests);

room.addMonster(monsterBuilder.buildMonster());

room.addMonster(monsterBuilder.buildMonster());

return room;

};

};

**Slime.h**

#pragma once

#include <string>

#include <iostream>

#include "LinkedList.h"

#include "Item.h"

#include <stdlib.h> /\* srand, rand \*/

using namespace std;

class Slime

{

protected:

int hP;

public:

string name;

int maxHP;

int attack;

string type;

string status;

Slime(int MaxHP, int HP, int Attack, string Type, string Name) :

maxHP(MaxHP), hP(HP), attack(Attack), type(Type), name(Name) {

status = "Normal";

};

Slime() {};

void setHP(int HP) {

hP = HP;

if (hP < 0) {

hP = 0;

}

if (hP > maxHP) {

hP = maxHP;

}

if (hP == 0 && status == "Normal") {

status = "Dead";

cout << name << " is dead." << endl;

}

}

int getHP() {

return hP;

}

int throwDice() {

return rand() % 10 + 1;

}

virtual void specialLuck() {

return ;

}

friend ostream& operator<<(ostream& aOstream, Slime slime) {

aOstream << slime.name << " -> " << "HP: " << slime.hP << "/" << slime.maxHP << " Attack: " << slime.attack << endl;

return aOstream;

}

};

**Stack.h**

#pragma once

#include "LinkedList.h"

#include <iostream>

using namespace std;

template <class DataType>

class Stack

{

public:

LinkedList<DataType> fElements;

Stack() {}

// add data at the end

void push(DataType data)

{

fElements.append(data);

}

// check is empty or not

int isEmpty()

{

return fElements.size == 0;

}

// Utility function to return top element in a stack

DataType& peek()

{

// Check for empty stack

if (!isEmpty())

return fElements.getValue(fElements.size - 1);

else

exit(1);

}

// Utility function to pop top element from the stack

DataType pop()

{

if (!isEmpty()) {

DataType removed = fElements.get(fElements.size - 1);

fElements.deleteAtIndex(fElements.size - 1);

return removed;

}

else

throw std::underflow\_error("Queue is empty!");

}

// Function to print all the elements of the stack

void display()

{

fElements.display();

}

};

**Tree.h**

#pragma once

#include "TreeNode.h"

#include <iostream>

using namespace std;

template <class DataType>

class Tree

{

public:

// Function to insert nodes in level order

TreeNode<DataType>\* insertLevelOrder(DataType arr[], TreeNode<DataType>\* root,

int i, int n, TreeNode<DataType>\* parent)

{

// Base case for recursion

if (i < n)

{

// create a node

TreeNode<DataType>\* node = new TreeNode<DataType>();

// initialise with data in arrat

node->data = arr[i];

node->left = node->right = NULL;

root = node;

// se this node to point to its parent

root->parent = parent;

// insert left child

root->left = insertLevelOrder(arr,

root->left, 2 \* i + 1, n, root);

// insert right child

root->right = insertLevelOrder(arr,

root->right, 2 \* i + 2, n, root);

}

return root;

}

// Function to print tree nodes in

// InOrder fashion

void inOrder(TreeNode<DataType>\* root)

{

if (root != NULL)

{

cout << root->data << " ";

inOrder(root->left);

inOrder(root->right);

}

}

};

**TreeNode.h**

#pragma once

#include <stdio.h>

template <class DataType>

class TreeNode

{

public:

DataType data;

TreeNode\* parent;

TreeNode\* left;

TreeNode\* right;

};

**WaterSlime.h**

#pragma once

#include "Slime.h"

class WaterSlime :

public Slime

{

public:

WaterSlime(int MaxHP, int HP, int attack, string Name = "Water Slime") : Slime(MaxHP, HP, attack, "Water Slime", Name) {};

void specialLuck() {

hP += 40;

maxHP += 40;

cout << "Increased HP by 40" << endl;

}

};

**WindSLime.h**

#pragma once

#include "Slime.h"

class WindSlime :

public Slime

{

public:

WindSlime(int MaxHP, int HP, int attack, string Name = "Wind Slime") : Slime(MaxHP, HP, attack, "Wind Slime", Name) {};

void specialLuck() {

attack += 3;

hP += 20;

maxHP += 20;

cout << "Increased Attack by 3 and HP by 20" << endl;

}

};

**Programming Project 1.cpp**

// Programming Project 1.cpp : This file contains the 'main' function. Program execution begins and ends there.

//

#include <iostream>

#include "CodeReuse.h"

using namespace std;

// Starting Point

int main()

{

/\* initialize random seed: \*/

srand(time(0));

CodeReuse codeReuse;

// get pointer to current node

TreeNode<DoublyLinkedList<Room>>\* currentNode = codeReuse.WorldSetup();

// a pointer to pointer and initialise it with the pointer to current node

TreeNode<DoublyLinkedList<Room>>\*\* currentNodePtr = &currentNode;

// get pointer to the first room in current node

DoublyLinkedNode<Room>\* currentRoom = (\*currentNodePtr)->data.getNode(0);

// a pointer to pointer and initialise it with the pointer to current room

DoublyLinkedNode<Room>\*\* currentRoomPtr = &currentRoom;

Party<Slime>\* slimes = codeReuse.PartySetup();

while (true) {

if (currentNodePtr) {

system("cls");

// Entered room

cout << "Entered room..." << endl;

// print current room description

cout << endl << (\*currentRoomPtr)->data.roomIntro << endl;

::Sleep(100);

// Battle if there is any monster

boolean fight = codeReuse.BattlePhase(&((\*currentRoomPtr)->data), slimes);

while (true) {

cout << endl << "Resting";

for (int i = 0; i < 10; i++) {

::Sleep(100);

cout << ".";

}

cout << endl;

codeReuse.Overview(\*slimes);

cout << endl << "Action Available:" << endl;

cout << "(1) Check Surrounding" << endl;

cout << "(2) Use Item" << endl;

cout << "(3) Gacha!!!" << endl;

cout << "(4) Move..." << endl;

string chooice;

cout << "Input: ";

cin >> chooice;

// if check surrounding

if (chooice == "1") {

if ((\*currentRoomPtr)->data.chest && !((\*currentRoomPtr)->data.chests->isEmpty())) {

Item chestItem = (\*currentRoomPtr)->data.openChest();

if (chestItem.name == "Aztec Gold") {

cout << endl << "You had found Aztec Gold... You will be cursed for a thousand years..." << endl;

cout << endl << " GAME OVER";

exit(0);

}

bool found = false;

for (int i = 0; i < slimes->inventory.size; i++) {

if (slimes->inventory.getValue(i).name == chestItem.name) {

slimes->inventory.getValue(i).amount++;

found = true;

break;

}

}

if (!found) {

Item smallPotion(chestItem.name, "", 1);

slimes->inventory.append(smallPotion);

}

cout << endl << "Obtain 1 " << chestItem.name << "." << endl;

}

else {

cout << endl << "Nothing found..." << endl;

}

}

else if (chooice == "2") {

codeReuse.UseItem(slimes);

}

else if (chooice == "3") {

codeReuse.GachaAction(slimes);

}

// move action

else if (chooice == "4") {

codeReuse.MoveAction(currentNodePtr, currentRoomPtr);

break;

}

}

}

}

}