Assessment Cover Sheet

This Assessment Cover Sheet is only to be attached to hard copy submission of assessments.

ASSESSMENT DETAILS

Unit title	DATA STRUCTURES AND PATTERNS	Tutorial /Lab Group	THURS, 4-6PM	Office use only
Unit code	COS30008	Due date	01 NOV 2021 (EXTENDED SUBMISSION)	
Name of lecturer/tutor	DR. MARK TEE KIT TSUN			
Assignment title	PROGRAMMING PROJECT 01			Faculty or school date stamp

STUDENT(S) DETAILS

	Student Name(s)	Student ID Number(s)
(1)	KATHY WONG HUI YING	101212259
(2)		
(3)		
(4)		
(5)		
(6)		

DECLARATION AND STATEMENT OF AUTHORSHIP

1. I/we have not impersonated, or allowed myself/ourselves to be impersonated by any person for the purposes of this assessment.
2. This assessment is my/our original work and no part of it has been copied from any other source except where due acknowledgement is made.
3. No part of this assessment has been written for me/us by any other person except where such collaboration has been authorised by the lecturer/tutor concerned.
4. I/we have not previously submitted this work for this or any other course/unit.
5. I/we give permission for my/our assessment response to be reproduced, communicated, compared and archived for plagiarism detection, benchmarking or educational purposes.

I/we understand that:

6. Plagiarism is the presentation of the work, idea or creation of another person as though it is your own. It is a form of cheating and is a very serious academic offence that may lead to exclusion from the University. Plagiarised material can be drawn from, and presented in, written, graphic and visual form, including electronic data and oral presentations. Plagiarism occurs when the origin of the material used is not appropriately cited.

Student signature/s

I/we declare that I/we have read and understood the declaration and statement of authorship.

(1)	KATHY W. HY	(4)	
(2)		(5)	
(3)		(6)	

Further information relating to the penalties for plagiarism, which range from a formal caution to expulsion from the University is contained on the Current Students website at **https://www.swinburne.edu.my/current-students/manage-course/exams-results-assessment**

Copies of this form can be downloaded from the Student Forms web page at **https://www.swinburne.edu.my/current-students/manage-course/exams-results-assessment/how-to-submit-work.php**

Table of Contents

Introduction

The main object of this Programming Project is to showcase understanding regarding Data Structures in terms of both the design and implementation when it comes to actual applications. In accordance to the given requirements, a software prototype is to be developed with implementations that demonstrate certain Data Structures and Patterns concepts.

Out of the ten concept requirements, only eight of them have been implemented in this project.

The application that has been developed for this project is titled "The Maze". It is a text-based console game, so player interacts with the system by inputting text through the command line interface to navigate a maze. It also makes use of the external SFML library to play different audio at different regions in the maze.

The main aim of this application is simple: for the player to survive navigating the maze and find the exit point. Entities in the game include: game objects which are capable of inducing death, and non-friendly non-playable characters (NPCs) which can damage and kill off the player.

Game Entities	Description
Category: Characters	
Playable Character (PC)	Main player. Has an inventory which can store items (health potion or weapon). The game ends when the player dies.
Non-Playable Character (NPC)	Sphinx (enemy). The player can either approach it and answer a riddle, attack it or run away from it. Each action will deal some damage to the player, unless the riddle is answered correctly, or if the Sphinx dies.
Category: Objects	
Pit	Death-inducing game object. The player will die if they fall into a pit. The game will end.
Breeze	Non-death-inducing game object.

	A breeze indicates there is a pit nearby.
Fear	Non-death-inducing game object. The sense of fear indicates there is a Sphinx nearby.

OBJECT ORIENTED PROGRAMMING

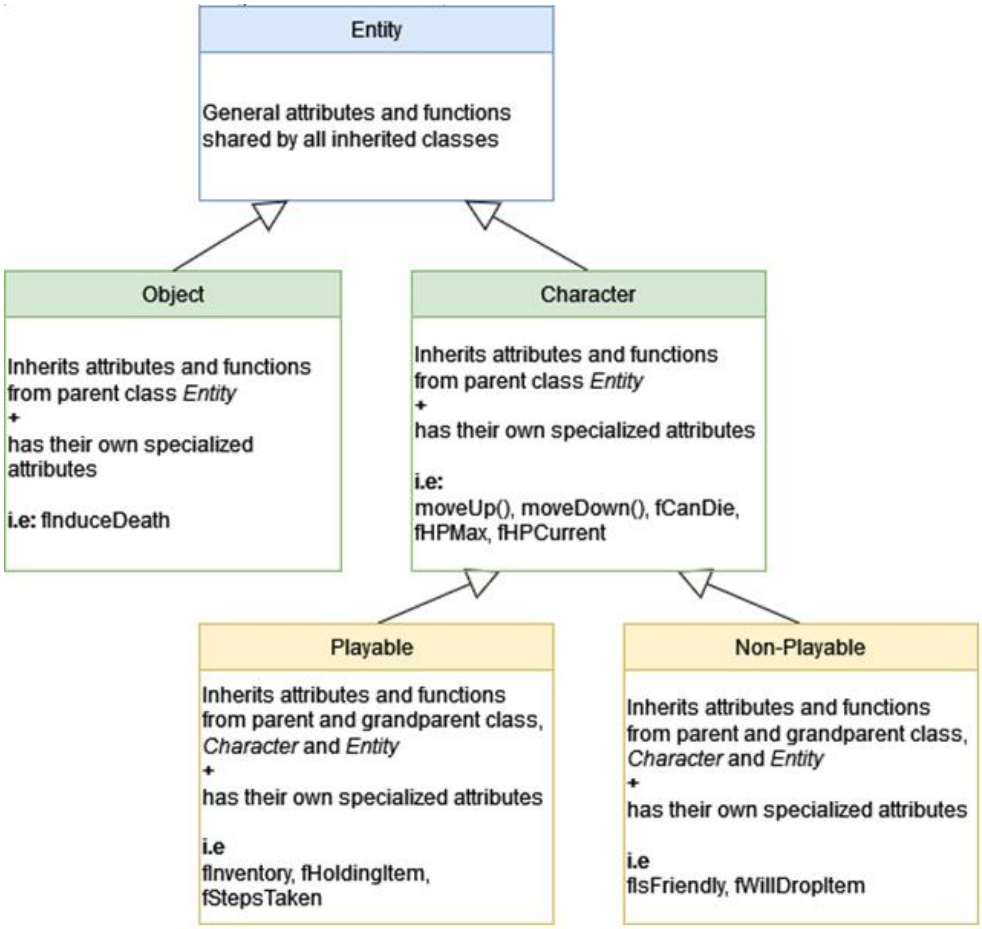
Inheritance of Derived Classes

Task Description

A generic game entity is created and named "Entity". This class would contain all the attributes common to each other (name, ID, position, and a message belonging to the entity). From here, the first two child classes are inherited from the parent class Entity: Object and Character.

Child class Object are game objects that can be interacted with (for example pits and breezes which indicate to the player what is nearby). Child class Character is used to indicate characters in a game, and are further specialized into two other classes: Playable and NonPlayable.

Concept



Inheritance is applied for the sake of convenience and minimizing errors. Declaring each entity as separate class but with the same attributes (plus some unique ones) would be a painstaking and error-prone process, and thus is not recommended.

Implementation and Output

In Entity.h

```
struct Position
{
    int X, Y;
};

class Entity
{

private:

protected:
    string fName; // Character name
    string fID; // Character ID
    Position fPosition; // (x, y) Position of character

    string fMessage; // Message dialogue

public:
    Entity(); //Default constructor
    Entity(string name, string id, int posx, int posy);


    //Declare Setters
    void setName(string name); // Setter for fName
    void setID(string id); // Setter for fIDS
    void setPositionX(int x); // Setter for x position
    void setPositionY(int y); // Setter for y position

    void setMessage(string msg); // Setter for fMessage

    //Declare Getters
    string getName(); // Getter for fName
    string getID(); // Getter for fID
    Position getPosition(); // Getter for x and y position

    string getMessage(); // Getter for fMessage

    //Friend Operator returns input stream
    friend istream& operator>>(istream& aIstream, Entity& aUnit);

    //Friend Operator returns output stream
    friend ostream& operator<<(ostream& aOstream, Entity& aUnit);

    void listen(string msg);
    void tell();


    //Destructor
    virtual ~Entity();
};
```

In Object.h (Parent class: Entity)

```
class Object : public Entity
{
private:
protected:
    bool fInduceDeath;

public:
    Object();
    Object(string name, string id, int posx, int posy, bool cancausedeath);

    //Declare Setters
    void setInduceDeath(bool choice);

    //Declare Getters
    bool getInduceDeath();
};
```

In Character.h (Parent class: Entity)

```
class Character : public Entity
{
private:
protected:
    int fMaxHP; // Max healthpoint
    int fCurrentHP; // Current healthpoint
    bool fCanDie; // Can the character die?
    bool fIsDead; // Is character dead? (will be = true if fCurrentHP = 0
                  // if fCanDie = false, fIsDead is permanently = false)

    //When char fight decrease hp by half

public:
    Character();
    Character(string name, string id, bool candie, int maxhp, int posx, int posy);


    //Declare Setters
    void setMaxHP(int maxhp); // Setter for fMaxHP
    void setCurrentHP(int currenthp); // Setter for fCurrentHP

    void setCanDie(bool candie); // Setter for fCanDie
    void setIsDead(bool isdead); // Setter for fIsDead


    //Declare Getters
    int getMaxHP(); // Getter for fMaxHP
    int getCurrentHP(); // Getter for fCurrentHP

    bool getCanDie(); // Getter for fCanDie
    bool getIsDead(); // Getter for fIsDead


    //Declare functions
    void increaseHP(int increaseby); // Function to increase HP
    void decreaseHP(int decreaseby); // Function to decrease HP

    virtual void Die();

    virtual void moveLeft(int x); // Move left by x amount
    virtual void moveRight(int x); // Move right by x amount
    virtual void moveUp(int y); // Move up by y amount
    virtual void moveDown(int y); // Move down by y amount

    //Destructor
    virtual ~Character();

};
```

In Playable.h (Parent class: Character, Grandparent class: Entity)

```
class Playable : public Character
{
private:
protected:

    string fInventory[3];   // has an accessable inventory

    bool fHoldingItem; // is PC holding an item
    string fItemHeld;

    int fStepsTaken; // Number of moves the character walked
    int fStepCounter; // Counts a 'cycle' of steps; reset at a certain value to carry out something
    //last step taken and number of steps

    Iterator1D* fInventoryPtr;

    StepRecordList* fStepListPointer;

public:
    Playable();
    Playable(string name, string id, bool candie, int maxhp, int posx, int posy);

    // Declare Setters
    void setHoldingItem(bool choice); // Setter for fHoldingItem
    void setItemHeld(string item);

    // Declare Getters
    int getStepsTaken(); // Getter for fStepsTaken
    int getStepCounter(); // Getter for fStepCounter
    bool getHoldingItem();
    string getItemHeld(); // Getter for fHoldingItem


    // Polymorphism
    void moveLeft(int x); // Move left by x amount
    void moveRight(int x); // Move right by x amount
    void moveUp(int y); // Move up by y amount
    void moveDown(int y); // Move down by y amount



    // Declaring functions
    void resetStepCount(int stepnumber); // Reset number of steps

    void showInventory(); // show inventory contents


    void inventoryNextItem(); //Increment inventory item using iterator
    void inventoryPrevItem(); //Decrement inventory item using iterator

    void addItem(string itemname); // Add item into inventory

    Iterator1D* getInventoryItem(); // Get inventory item (pointer to element)

    void takeItem(); // Get inventory item void GetItem(Iteraor* iterator ptr, return fItemHeld);
    void dropItem(); // Drop inventory item

    void viewLastSteps(int limit); // Cycle back from doubly linked list and obtain last steps

    //Destructor
    virtual ~Playable();
};
```

In NonPlayable.h (Parent class: Character, Grandparent class: Entity)

```
class NonPlayable : public Character
{
private:
protected:
    bool fWillDropItem;
    string fItemHeld;
    bool fIsFriendly; // is this NPC friendly? if false, will attack PC

public:
    NonPlayable();
    NonPlayable(string name, string id, bool candie, int maxhp, int posx, int posy, bool isfriendly)

    string DropItem(); // Return item dropped by the enemy

    // Declare Setters
    void setWillDropItem(bool choice); // Setter for fWillDropItem
    void setItemHeld(string itemname); // Setter for fItemHeld
    void setIsFriendly(bool choice); // Setter for fIsFriendly

    // Declare Getters
    bool getWillDropItem(); // Getter for fWillDropItem
    string getItemHeld(); // Getter for fItemHeld
    bool getIsFriendly(); // Getter for fIsFriendly

    // Declare functions
    void die();

    //Destructor
    virtual ~NonPlayable();
};
```

Troubleshooting

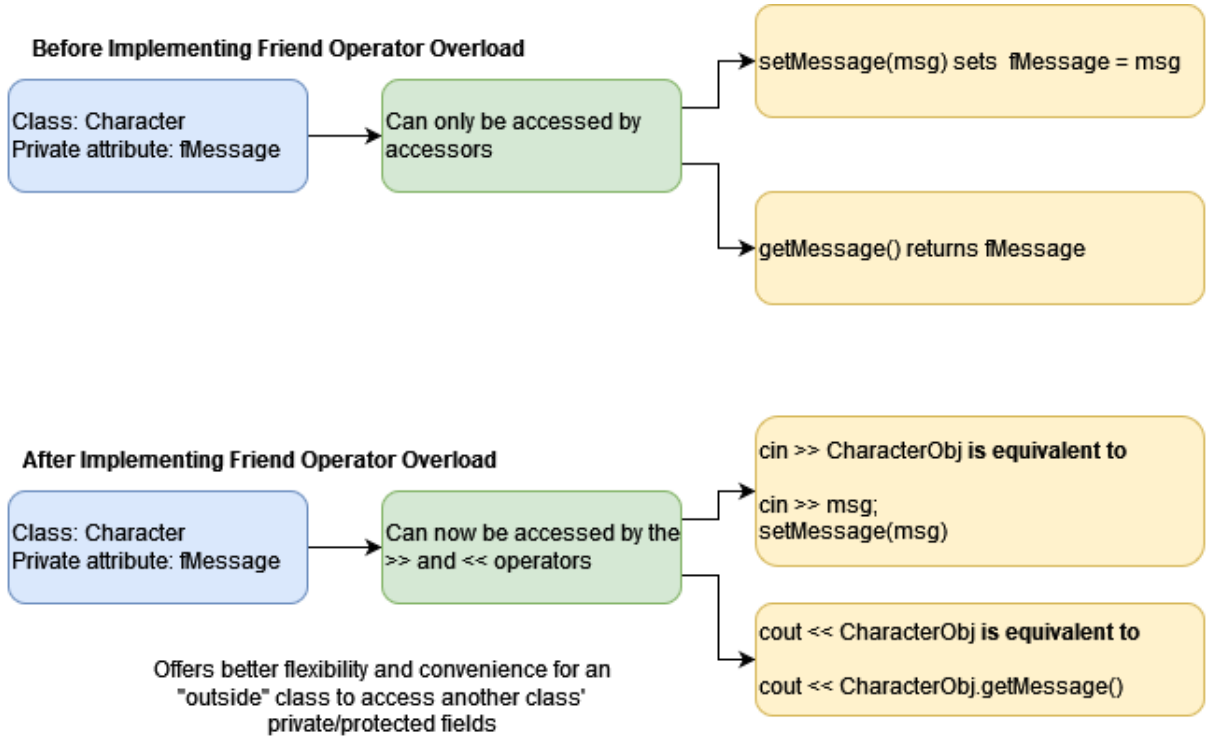
No errors were met during implementation of this segment.

Friend Operator Overloading

Task Description

Friend operator over-loadings are used in order to access a private or protected attribute of another class without violating the OOP Encapsulation concept. In this case, the insertion << operator and extraction >> operator are made to access the fMessage attribute of a Character class (or any class that has Character as its base class) and either output it to the console (via insertion) or change its value (via extraction).

Concept



It is possible to just make do with cout << CharacterObj.getMessage() or cin >> msg; amd CharacterObj.setMessage(msg).

However, making use of friend operator allows for better flexibility and convenience for an “outside” class to access the protected fMessage attribute, since there is no need to go through accessors but rather straight through the operators.

Implementation and Output

In Entity.h

```
//Friend Operator returns input stream
friend istream& operator>>(istream& aIstream, Entity& aUnit);

//Friend Operator returns output stream
friend ostream& operator<<(ostream& aOstream, Entity& aUnit);
```

In Entity.cpp

```cpp
// Friend Operator returns input stream
istream& operator>>(istream& aIstream, Entity& aUnit)
{
    getline(aIstream, aUnit.fMessage); //get a line of string
    return aIstream;
}

// Friend Operator returns output stream
ostream& operator<<(ostream& aOstream, Entity& aUnit)
{
    aOstream << aUnit.fMessage << endl;
    return aOstream;
}
```

In Main.cpp (Test code)

```cpp
int main()
{
    Playable* myPC = new Playable("Player01", "0001", true, 15, 0, 0); // Create a PC on the heap

    cout << "Using friend operators to modify and retrieve fMessage: " << endl;
    cout << "Input via cin>> *(myPC):";
    cin >> *(myPC);
    cout << "Output via cout<< *(myPC):" << * (myPC);
    cout << endl;


    cout << "Using accessors check and modify fMessage: " << endl;
    cout << "Output getMessage(): " << myPC->getMessage() << endl; // Obtain message using getter
    cout << "Store input to variable: ";
    string msg;
    cin >> msg;
    myPC->setMessage(msg); // Set message using setter
    cout << "Output via cout<< *(myPC): " << *(myPC);

    delete myPC;

    return 0;
}
```

Console Output (Test code)

```
Using friend operators to modify and retrieve fMessage:
Input via cin>> *(myPC):Hello
Output via cout<< *(myPC):Hello

Using accessors check and modify fMessage:
Output getMessage(): Hello
Store input to variable: Goodbye
Output via cout<< *(myPC): Goodbye
```

Troubleshooting

No errors were met during implementation of this segment.

Polymorphism

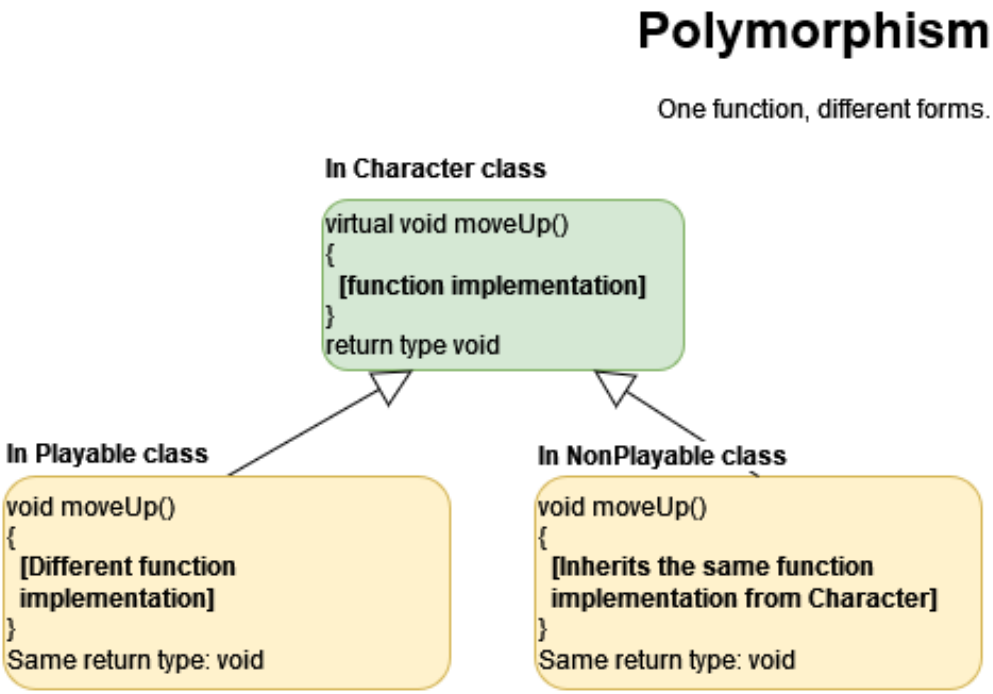
Task Description

The functions moveUp(), moveDown(), moveLeft() and moveRight() in Character class determine how many steps are taken by a character in one direction. These functions will then be inherited by child classes Playable and NonPlayable so their objects can move around.

Due to specialization, there are certainly some attributes that are declared unique to each of the inherited classes. The same functions inherited might be expected to carry out certain operations a little differently, despite having the same name, input parameters and return type value. This is where polymorphism comes in.

Therefore, the abovementioned functions are declared as virtual. The NonPlayable class will just inherit them as is, while the Playable class will have a slightly different implementation to the function itself.

Concept



Implementation and Output

In Character.h	In Character.cpp

```
virtual void moveLeft(int x); /
virtual void moveRight(int x);
virtual void moveUp(int y); //
virtual void moveDown(int y); /
```

```
void Character::moveLeft(int x) // Move left by x amount
{
    fPosition.X = fPosition.X - x;
}

void Character::moveRight(int x) // Move right by x amount
{
    fPosition.X = fPosition.X + x;
}

void Character::moveUp(int y) // Move up by y amount
{
    fPosition.Y = fPosition.Y + y;
}

void Character::moveDown(int y) // Move down by y amount
{
    fPosition.Y = fPosition.Y - y;

}
```

In Playable.h	In Playable.cpp

`// Polymorphism`
`void moveLeft(int x);`
`void moveRight(int x);`
`void moveUp(int y); //`
`void moveDown(int y);`
	`void Playable::moveLeft(int x) // Move left by x amount`
	`{`
	`fPosition.X = fPosition.X - x;`
	`fStepsTaken++;`
	`fStepCounter++;`
	`fStepListPointer->addStep("Left", fStepsTaken, fPosition.X, fPosition.Y);`
	`}`
	`void Playable::moveRight(int x) // Move right by x amount`
	`{`
	`fPosition.X = fPosition.X + x;`
	`fStepsTaken++;`
	`fStepCounter++;`
	`fStepListPointer->addStep("Right", fStepsTaken, fPosition.X, fPosition.Y);`
	`}`
	`void Playable::moveUp(int y) // Move up by y amount`
	`{`
	`fPosition.Y = fPosition.Y + y;`
	`fStepsTaken++;`
	`fStepCounter++;`
	`fStepListPointer->addStep("Up", fStepsTaken, fPosition.X, fPosition.Y);`
	`}`
	`void Playable::moveDown(int y) // Move down by y amount`
	`{`
	`fPosition.Y = fPosition.Y - y;`
	`fStepsTaken++;`
	`fStepCounter++;`
	`fStepListPointer->addStep("Down", fStepsTaken, fPosition.X, fPosition.Y);`
	`}`

Troubleshooting

No errors were met during implementation of this segment.

COMPOSITE DATA STRUCTURES

Struct

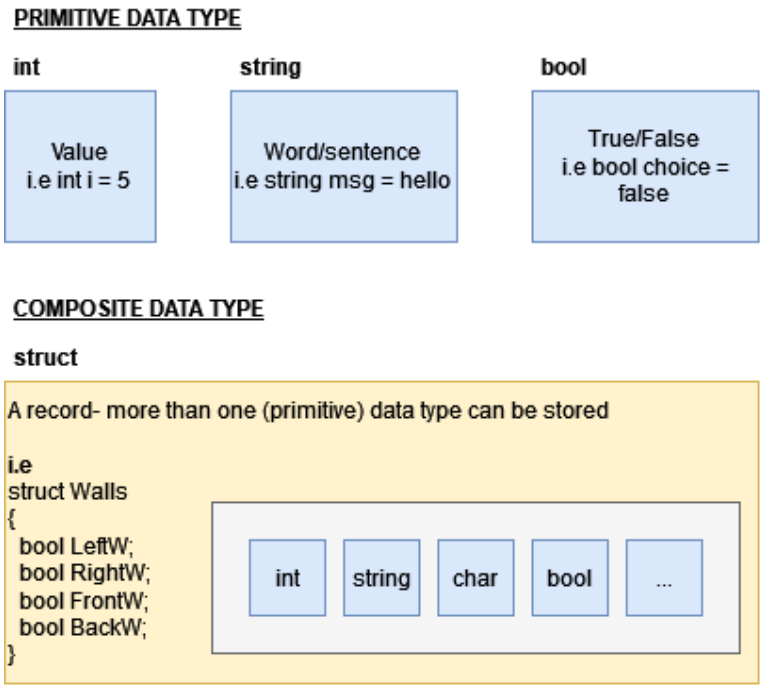
Task Description

In the Entity class, a struct is used to represent the position of a game entity, Position with co-ordinates X and Y.

In the MapPiece class, a struct is used to keep track whether there are walls in that certain area. The struct tag name is Walls, and has the member definitions LeftW, RightW, FrontW, and BackW, all boolean, to indicate if there are walls on the left, right, front and back respectively.

In this case, struct is used instead of just declaring them as individual attributes of a class as it provides a better performance due to it being a value type. Apparently, this means a lightening the load of garbage collection when objects are instantiated on the heap.

Concept



Accessing primitive data types:

i;

msg;

choice;

Accessing struct variables:

structureTag. MemberName

Walls.LeftW

Walls.RightW

Walls.FrontW

Walls.BackW

The main reason for using struct is because for Position, X and Y make up the coordinates of position and thus should be grouped together. The same goes for walls.

Initially, arrays were considered to be used instead of struct. However, for each element of the array, one getter would need to be declared. This is an absolute hassle, and doing it so does not make it any different than from declaring each elements as a different class attribute.

Therefore, struct is used, there only needs to be one getter declared for the entire struct. Each individual members can then be accessed by declaring the same getter, but adding on the member name using dot operator.

Implementation and Output

For struct Walls

In MapPiece.h

```h
struct Walls
{
    bool LeftW, RightW, FrontW, BackW;
};


class MapPiece
{
private:

protected:
    Walls fWalls;
    Object* ObjectPtr;
    NonPlayable* NPCPointer;

public:
    MapPiece();

    MapPiece(Walls walls);

    // Declare Getters
    Walls getWalls(); // Getter for fWalls
```

In MapPiece.cpp

```cpp
MapPiece::MapPiece() // Default constructor
{
    fWalls.LeftW = false;
    fWalls.RightW = false;
    fWalls.FrontW = false;
    fWalls.BackW = false;

    ObjectPtr = nullptr;
    NPCPointer = nullptr;
}

MapPiece::MapPiece(Walls walls)
{
    fWalls = walls;

    ObjectPtr = nullptr;
    NPCPointer = nullptr;
}

Walls MapPiece::getWalls() // Getter for fWalls
{
    return fWalls;
}
```

In Main.cpp

```cpp
int main()
{
    // Defining the walls for MapPiece object
    Walls CorridorH; // Horizontal corridor
    {
        CorridorH.LeftW = false;
        CorridorH.RightW = false;
        CorridorH.FrontW = true;
        CorridorH.BackW = true;
    }

    MapPiece* myMap = new MapPiece(CorridorH);
    cout << "1 = true; 0 = false" << endl;
    cout << "Get Left Wall: " << myMap->getWalls().LeftW << endl;
    cout << "Get Right Wall: " << myMap->getWalls().RightW << endl;
    cout << "Get Front Wall: " << myMap->getWalls().FrontW << endl;
    cout << "Get Back Wall: " << myMap->getWalls().BackW << endl;

    delete myMap;

    return 0;
}
```

Console Output (Test Code)

```
1 = true; 0 = false
Get Left Wall: 0
Get Right Wall: 0
Get Front Wall: 1
Get Back Wall: 1
```

For struct Position

In Entity.h

```
struct Position
{
    int X, Y;
};

class Entity
{

private:

protected:
    string fName; // Character name
    string fID; // Character ID
    Position fPosition; // (x, y) Position of character

void setPositionX(int x); // Setter for x position
void setPositionY(int y); // Setter for y position

Position getPosition(); // Getter for x and y position
```

In Entity.cpp

```
void Entity::setPositionX(int x) // Setter
{
    fPosition.X = x;
}

void Entity::setPositionY(int y) // Setter
{
    fPosition.Y = y;
}

Position Entity::getPosition() // Getter
{
    return fPosition;
}
```

In Main.cpp

```
cout << endl << "What is your name?:" << endl;
cout << "Input: ";
string myname = "";
cin >> myname;

myPC = new Playable(myname, "0001", true, 15, 0, 0); // Create a PC on the heap


myPC->moveUp(1); // Proceed to move up by one step

cout << "Current position :" << myPC->getPosition().X << ", " << myPC->getPosition().Y << endl; // Cout where the PC
```

Console Output

```
Current position :0, 1
```

Troubleshooting

References used:

StackOverflow, 2011, *creating an array of structs in c++*, StackOverflow, viewed 31 October 2021 < https://stackoverflow.com/questions/6810656/creating-an-array-of-structs-in-c>.

C# Corner, 2017, *What Is Struct And When To Use Struct In C#*, C# Corner, viewed 31 October 2021 <https://www.c-sharpcorner.com/article/what-is-structure-and-when-to-use-in-c-sharp/>.

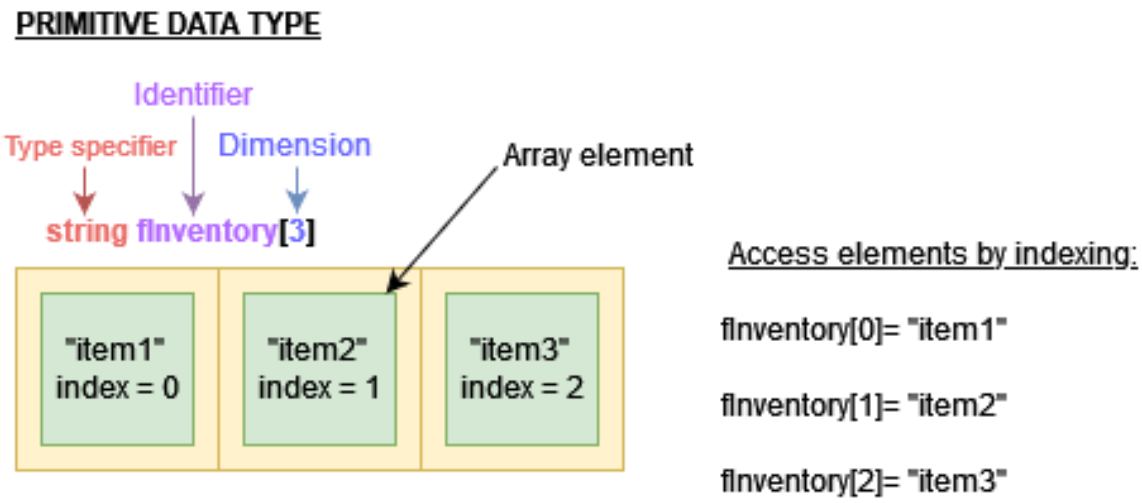
Array

Task Description

The Playable character has an inventory that it can use to store items dropped by the Sphinx. Thus, a one-dimensional array of size three is declared to store up to a maximum of three items throughout the course of the game.

In the main function, a two-dimensional array of MapPiece pointers is also declared- this is used to emulate the maze for the player to navigate. Each element of this array (named myMap is a pointer which will be pointing to a single "piece" of map.

Concept



Other choices considered to be used for the applications mentioned above are: linked-lists and doubly-linked lists. However, since the prototype being coded is relatively small-scale (as compared to actual game applications), and it is still relatively easy to build a 10x10 map, so arrays are chosen due to familiarity of usage and ease of access.

Linked-lists will be applied for another role (to be discussed in the sections below).

Implementation and Output

For fInventory

In Playable.h

```
class Playable : public Character
{
private:
protected:

    string fInventory[3];   // has an accessable inventory

public:
    Playable();

 void showInventory(); // show inventory contents

void addItem(string itemname); // Add item into inventory
```

In Playable.cpp

```
Playable::Playable()
{
    fInventory[0] = " ";
    fInventory[1] = " ";
    fInventory[2] = " ";

    fHoldingItem = false;
    fItemHeld  = ""; // no value, no item currently
    fInventoryPtr = new Iterator1D(fInventory, 3);
    fStepsTaken = 0;
    fStepCounter = 0;

    fStepListPointer = new StepRecordList();
}

void Playable::showInventory() // show inventory contents
{
    for (int i = 0; i < 3; i++)
    {
        cout << " " << fInventory[i] << " ";
    }
}

void Playable::addItem(string itemname) // Add item into inventory
{
    bool itemadded = false;


    for (int i = 0; i < 3; i++)
    {
        if ((fInventory[i] == "") & (itemadded == false))
        {
            fInventory[i] = itemname;
            itemadded = true;
            cout << itemname << " added into inventory!" << endl;
        }
    }

    if (itemadded == false) // Means the inventory is full
    {
        cout << "Inventory is full! Unable to take." << endl;
    }

    if (fHoldingItem == true)
    {
        fHoldingItem = false;
        fItemHeld = "";
    }

}
```

In Main.cpp (Test code)

```
int main()
{
    Playable* myPC = new Playable();
    myPC->addItem("item1");
    myPC->addItem("item2");
    myPC->addItem("item3");

    cout << endl << "Showing inventory items: " << endl;
    myPC->showInventory();

    delete myPC;

    return 0;
}
```

Console Output (Test code)

```
item1 added into inventory!
item2 added into inventory!
item3 added into inventory!

Showing inventory items:
 item1  item2  item3
```

Troubleshooting

No errors were met during implementation of this segment.

Singly-Linked List

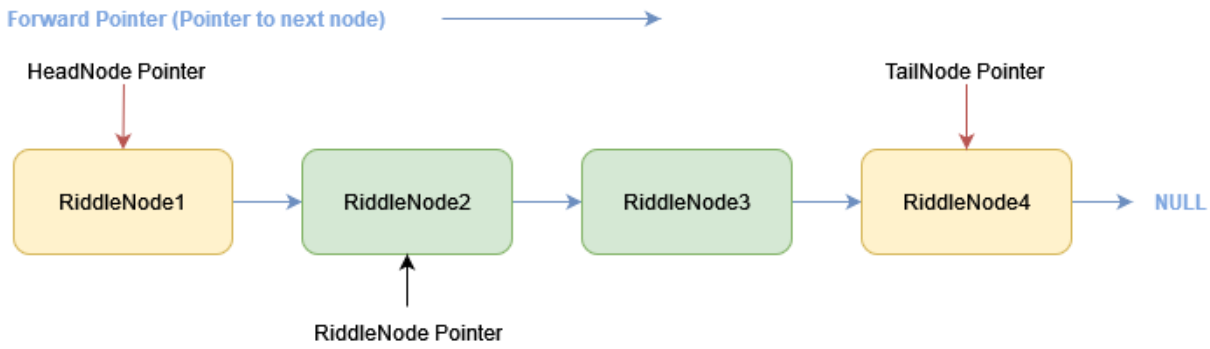
Task Description

In the game, there is a linked list known as RiddleNodeList. This is used to store a riddle in each node, and the riddle question and answer choices will be input as a message for each Sphinx the player meets along the way.

Therefore, this is like a "riddle reserve" for Sphinxes to retrieve their questions from, instead of having to declare a unique riddle for each NonPlayable Sphinx object. This also means that the same Sphinx can give different riddles each time (if the same object is approached by the player again and again).

If a riddle has already been given, the riddle node pointer will be cycled to point at the next immediate node. If the end of the list has been reached, the RiddleNode will once again made to point at the head of the list.

Concept



Singly-linked list is most suitable for this application, since it only requires the next riddle to be obtained. There is no need to go backwards through the list but rather just in one direction (therefore doubly-linked list is not used).

A stack was also considered, since it technically acts in the same way as a singly-linked list. However, stack operations are 'push' and 'pop', and riddles that have been 'popped' are released from memory. This will lead to no more riddles to give to the Sphinxes.

Implementation and Output

In RiddleRecordList.h

```
class RiddleRecordList
{

private:
    // Linked List of skill nodes
    RiddleNode* headNode; // Pointer to head of list of skills
    RiddleNode* tailNode; // Pointer to tail of list of skills
    RiddleNode* riddlePtr; // Pointer to any skill in the list

public:

    RiddleRecordList();
    void addRiddle(string question, string choice1, string choice2, string choice3, int correctchoice);

    // Declare functions to manage skill list
    void removeTop(); // Remove Riddle (unlearn) at the front of list

    string showRiddleAndChoices(); // Output Riddle and its choices

    int correctAns(); // Obtain Integer for correct answer

    void NextRiddle(); // Point at next riddle

    // Declare Getters
    RiddleNode* getRiddlePtr();

    ~RiddleRecordList();
};
```

In RiddleRecordList.cpp

```cpp
RiddleRecordList::RiddleRecordList()
{
    tailNode = new RiddleNode("", "", "", "", 0); // Create sentinel tail node on heap
    headNode = tailNode; // also poiint at end

    riddlePtr = NULL; // Don't point at anything yet
}

void RiddleRecordList::addRiddle(string question, string choice1, string choice2, string choice3, int correctchoice)
{
    riddlePtr = new RiddleNode(question, choice1, choice2, choice3, correctchoice);

    riddlePtr->prependRiddle(*headNode);
    headNode = riddlePtr; // point at beginning of list
}

 // Declare functions to manage skill list
void RiddleRecordList::removeTop() // Remove Riddle at the front of list
{
    riddlePtr = headNode->getNextNode(); // Point at next node
    headNode->removeRiddle(); // Remove riddle
    headNode = riddlePtr; // Point at new head of list
}

// Declare functions to manage skill list
void RiddleRecordList::removeTop() // Remove Riddle at the front of list
{
    riddlePtr = headNode->getNextNode(); // Point at next node
    headNode->removeRiddle(); // Remove riddle
    headNode = riddlePtr; // Point at new head of list
}

string RiddleRecordList::showRiddleAndChoices() // Output Riddle and its choices
{
    //and delete riddles already asked
    string qna = "";

    qna = riddlePtr->getQuestion() + '\n' + riddlePtr->getAnsChoices();

    return qna;
}

void RiddleRecordList::NextRiddle() // Obtain Integer for correct answer
{
    //After adding riddle riddlePtr will definitely be at the head of list

    riddlePtr = riddlePtr->getNextNode(); // Point at head of list
    // Note that skillPtr is now pointing at our current node that we may want to delete

    cout << "next riddle!" << endl;
    if (riddlePtr == tailNode)
    {
        riddlePtr = headNode; // Go back to the head node
    }
}
```

```cpp
// Declare getters
RiddleNode* RiddleRecordList::getRiddlePtr() // Getter for riddlePtr
{
    return riddlePtr;
}


int RiddleRecordList::correctAns()
{
    int ansindex = riddlePtr->getCorrectAnsIndex(); // Point at next riddle

    return ansindex;
}



RiddleRecordList::~RiddleRecordList()
{
    riddlePtr = headNode; // Point at head of list
    // Note that skillPtr is now pointing at our current node that we may want to delete

    while (riddlePtr != tailNode) // While not at end of the skills list
    {
        headNode = headNode->getNextNode();

        riddlePtr->removeRiddle(); //Start deleting from beginning of list
        delete riddlePtr;

        riddlePtr = headNode;
    } // Iterate process when end of list is not reached

    delete tailNode;
}
```

In Main.cpp (Test code)

```cpp
int main()
{
    // Generating Riddle List- this list will be cycled through in the game-> NPCs will access this list to obtain their riddles
    RiddleRecordList* RiddleRecordPtr = new RiddleRecordList();
    RiddleRecordPtr->addRiddle("What do you call a fish with no eyes?", "Myxine Glutinosa", "Blind", "I dunno... fsh?", 2);
    RiddleRecordPtr->addRiddle("At night they come without being fetched, and by day they are lost without being stolen. What are
    RiddleRecordPtr->addRiddle("What is the meaning of life?", "Whatever which we choose to give it", "There's none", "E", 2);
    RiddleRecordPtr->addRiddle("Why is a raven like a writing desk?", "They're both not made of cheese", "Because Poe wrote on bo
    RiddleRecordPtr->addRiddle("Why did the chicken cross the road?", "To get to the other side", "There was a car coming", "I do

    cout << RiddleRecordPtr->showRiddleAndChoices();
    cout << endl;
    cout << endl;
    RiddleRecordPtr->NextRiddle();
    cout << RiddleRecordPtr->showRiddleAndChoices();
    cout << endl;
    cout << endl;
    RiddleRecordPtr->NextRiddle();
    cout << RiddleRecordPtr->showRiddleAndChoices();
    cout << endl;
    cout << endl;

    delete RiddleRecordPtr;
    return 0;
}
```

Console Output (Test code)

```
Why did the chicken cross the road?
1) To get to the other side
2) There was a car coming
3) I don't know. Why?


next riddle!
Why is a raven like a writing desk?
1) They're both not made of cheese
2) Because Poe wrote on both
3) I haven't the slightest idea


next riddle!
What is the meaning of life?
1) Whatever which we choose to give it
2) There's none
3) E
```

Troubleshooting

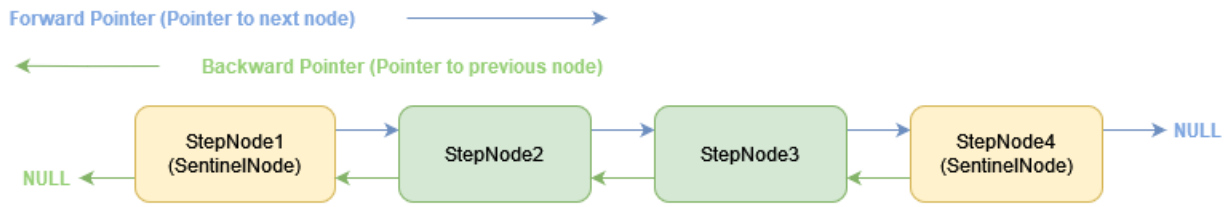
No errors were met during implementation of this segment.

Doubly-Linked List

Task Description

A doubly-linked list is used to keep track of all steps taken by the player. With each new step, a new node object is added to the list StepNodeList. Then, through this, the player is allowed to review the last five steps taken by cycling backwards from the end of the list.

Concept



A doubly-linked list is used because it is dynamic (unlike static containers such as arrays- there is no fixed size for how many steps a player should take), and has both a forward and backward pointer pointing to the next and previous nodes, so the step nodes can be accessed from both directions.

Another choice to be considered would be the use of vectors in the place of this doubly-linked list. Vectors allow for quick insertion at the end, the same as doubly-linked lists. (Although doubly-linked lists also have an added advantage of allowing quick insert and erase in the front and back, and also easier to insert and delete a node from the middle of the list compared to

vector—however, these characteristics are not required for the current implementation of StepNodeList.)

Since the concept requirements do not ask for vectors, it was thus not used, and doubly-linked list was chosen instead.

Implementation and Output

In StepRecordList.h

```
class StepRecordList
{

private:
    // Linked List of skill nodes
    StepNode* headNode; // Pointer to head of list of skills
    StepNode* tailNode; // Pointer to tail of list of skills
    StepNode* stepPtr; // Pointer to any skill in the list

public:

    StepRecordList();

     void addStep(string direction, int stepnumber, int x, int y); // Add skill (learn new skill

     string showLastSteps(int limit); // View character steps-> print last limit steps?


     ~StepRecordList();
};
```

In StepRecordList.cpp

```cpp
StepRecordList::StepRecordList()
{

    headNode = new StepNode("SentinelStart", NULL, NULL, NULL); // Create sentinel head node on heap
    tailNode = new StepNode("SentinelEnd", NULL, NULL, NULL); // Create sentinel tail node on heap

    headNode->appendStep(*tailNode); // Link the head and tail sentinel nodes together

    stepPtr = headNode; // Point at head of list
}



void StepRecordList::addStep(string direction, int stepnumber, int posx, int posy) // Add step
{
    stepPtr = new StepNode(direction, stepnumber, posx, posy); // Create new step on the heap

    tailNode->prependStep(*stepPtr); // Prepend the newly added step to the list (add in before sentinel tail node
    stepPtr = tailNode->getPrevNode(); // Point at current (newly added) step
}

string StepRecordList::showLastSteps(int limit) // View character steps
{
    string outputlist = "Showing the last " + to_string(limit) + " steps:" + '\n';
    int n = 0;

    stepPtr = tailNode; // Point to beginning of step list
    // Point to next step. Since we start at headNode, we want to point at the next one

    stepPtr = stepPtr->getPrevNode();

    for (int i = 0; i < limit; i++)
    {
        if (stepPtr == headNode)
        {
            break;
        }

        outputlist = outputlist + to_string(i+1) + ") " + stepPtr->getDirection() + '\n';
        stepPtr = stepPtr->getPrevNode();
    }

    return outputlist;
}

StepRecordList::~StepRecordList()
{
    StepNode* temp; // Create a temporary pointer
    stepPtr = headNode; // Point at head of step list

    stepPtr = headNode; // Point to beginning of step list
    stepPtr = stepPtr->getNextNode(); // Point to next step. Since we start at headNode, we want to point at the next on

    // Note that stepPtr is now pointing at our current node that we may want to delete

    while (stepPtr != tailNode) // While not at end of the steps list
    {
        stepPtr = stepPtr->getNextNode(); // Point stepPtr at the next node
        temp = stepPtr->getPrevNode();  // Point temp at the 'current' node

        delete temp; // Destruct the step node
    } // Iterate process when end of step list is not reached

    delete headNode;
    delete tailNode;
}
```

In Main.cpp (snippet inside Game Loop)

```
if (userInput == "w")  // Go UP
{
    if (myMap[myPC->getPosition().X][myPC->getPosition().Y]->getWalls().FrontW == false) // If there is no wall in front
    {
        myPC->moveUp(1); // Proceed to move up by one step
    }
    else
    {
        cout << "Can't go up. There's a wall blocking me." << endl;
    }

    cout << "Current position :" << myPC->getPosition().X << ", " << myPC->getPosition().Y << endl; // Cout where the PC is
    userInput = "run"; // Force user input to continue in the loop
}

if (userInput == "a") // Go LEFT
{
    if (myMap[myPC->getPosition().X][myPC->getPosition().Y]->getWalls().LeftW == false) // If there is no wall on left
    {
        myPC->moveLeft(1); // Proceed to move left by one step
    }
    else
    {
        cout << "Can't go left. There's a wall blocking me." << endl;
    }

    cout << "Current position :" << myPC->getPosition().X << ", " << myPC->getPosition().Y << endl; // Cout where the PC is
    userInput = "run";
}


if (userInput == "d") // Go RIGHT
{
    if (myMap[myPC->getPosition().X][myPC->getPosition().Y]->getWalls().RightW == false) // If there is no wall on = right
    {
        myPC->moveRight(1); // Proceed to move right by one step
    }
    else
    {
        cout << "Can't go right. There's a wall blocking me." << endl;
    }

    cout << "Current position :" << myPC->getPosition().X << ", " << myPC->getPosition().Y << endl; // Cout where the PC i
    userInput = "run";
}

if (userInput == "s") // Go DOWN
{
    if (myMap[myPC->getPosition().X][myPC->getPosition().Y]->getWalls().BackW == false) // If there is no wall behind
    {
        myPC->moveDown(1); // Proceed to move down one step
    }
    else
    {
        if ((myPC->getPosition().X == 0) & (myPC->getPosition().Y == 0)) // If at starting position
        {
            myPC->setMessage("This is where I started from... I musn't go backwards."); // Do not move out of map
            cout << *(myPC) << endl;
        }
        else
        {
            cout << "Can't go down. There's a wall blocking me." << endl;
        }

    }
    cout << "Current position :" << myPC->getPosition().X << ", " << myPC->getPosition().Y << endl; // Cout where the PC i
    userInput = "run";
}

if (userInput == "trace") // Trace back the ast five steps made by PC
{
    cout << "====================" << endl;
    myPC->viewLastSteps(5);
    cout << "====================" << endl;
    cout << endl;
}
```

Console Output (snippet)

```
Press 'w' to take your first step!
Input: w
Current position :0, 1
Input: d
Current position :1, 1
Input: w
Current position :1, 2
Something feels off.
Input: s
Current position :1, 1
Input: a
Current position :0, 1
Input: s
Current position :0, 0
Input: d
Can't go right. There's a wall blocking me.
Current position :0, 0
Input:
w
Current position :0, 1
Input: trace
====================
Showing the last 5 steps:
1) Up
2) Down
3) Left
4) Down
5) Up
====================
```

Troubleshooting

References used:

StackOverflow, 2013, *Linked List vs Vector*, StackOverflow, viewed 31 October 2021 <https://stackoverflow.com/questions/19039972/linked-list-vs-vector>.

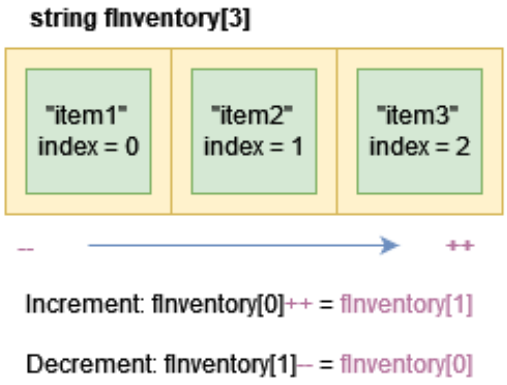
DESIGN PATTERNS

Iterator

Task Description

Arrays are accessed through indexing by default. In this case, for the array that acts as the player's inventory, it is accessed differently by implementing an iterator design pattern. By overloading the "++" and "--" operators, the inventory can be accessed as if the player is "scrolling" through them.

Concept



Implementation and Output

In Iterator.h

```
class Iterator
{
protected:

    string* fArrayElements;
    int fIndex;

public:
    Iterator();

    Iterator(string aArray[], int aIndex = 0);

    string operator*();

    Iterator& operator++(); //prefix operator

    Iterator operator++(int); //postfix operator

    Iterator& operator--(); //prefix operator

    Iterator operator--(int); //postfix operator

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

    bool operator!=(const Iterator& aOther) const;

    virtual Iterator begin() const;

    virtual Iterator end() const;


    virtual int getIndex(); //Added new

   virtual int getIndex();

   virtual ~Iterator();
};
```

Iterator.cpp

```cpp
Iterator::Iterator()
{
    fArrayElements = NULL;
    fIndex = 0;
}

Iterator::Iterator(string aArray[], int aIndex)
{
    fArrayElements = aArray;
    fIndex = aIndex;
}

string Iterator::operator*()
{
    return fArrayElements[fIndex];
}

Iterator& Iterator::operator++()
{
    //prefix operator
    fIndex++;
    return *this;
}

Iterator& Iterator::operator--()
{
    //prefix operator
    fIndex--;
    return *this;
}

Iterator Iterator::operator--(int)
{
    //postfix operator
    Iterator temp = *this;
    fIndex--;
    return temp;
}

bool Iterator::operator==(const Iterator& aOther) const
{
    return (fIndex == aOther.fIndex); //Return true or false
}

bool Iterator::operator!=(const Iterator& aOther) const
{
    return (fIndex != aOther.fIndex); //Return true or false
}

Iterator Iterator::begin() const
{
    return Iterator();
}

Iterator Iterator::end() const
{
    return Iterator();
}


int Iterator::getIndex()
{
    return fIndex;
}


 //Destructor
Iterator::~Iterator()
{
}
```

Iterator1D.h

```
class Iterator1D :
    public Iterator
{

private:
    int fEndIndex;

public:
    Iterator1D();

    Iterator1D(string aArray[], int aArrSize, int aIndex = 0);

    Iterator begin() const;

    Iterator end() const;

    ~Iterator1D();
};
```

Iterator1D.cpp

```
Iterator1D::Iterator1D()
{
    fArrayElements = NULL;
    fIndex = 0;
    fEndIndex = 0;
}


Iterator1D::Iterator1D(string aArray[], int aArrSize, int aIndex) : Iterator(aArray, aIndex)
{
    fArrayElements = aArray;
    fEndIndex = aArrSize - 1; //Show end of Array
}

Iterator Iterator1D::begin() const
{
    int lArrSize = fEndIndex + 1;
    return Iterator1D(fArrayElements, lArrSize);

}

Iterator Iterator1D::end() const
{
    int lArrSize = fEndIndex + 1;
    return Iterator1D(fArrayElements, lArrSize, fEndIndex);

}

 //Destructor
Iterator1D::~Iterator1D()
{
}
```

In Playable.h

```
    void inventoryNextItem(); //Increment inventory item using iterator
    void inventoryPrevItem(); //Decrement inventory item using iterator
```

In Playable.cpp

```
void Playable::inventoryPrevItem() //Decrement iterator to previous item
{
    if ((*fInventoryPtr) == (fInventoryPtr->begin()))
    {
        fInventoryPtr->begin();
        //Don't do anything if at min beginning of inventory
    }

    else
    {
        --(*fInventoryPtr);
    }

    for (int i = 0; i < 3; i++)
    {
        if (fInventoryPtr->getIndex() == i)
        {
            cout << " [" << fInventory[i] << "] ";
        }

        else
        {
            cout << " " << fInventory[i] << " ";
        }
    }

}


void Playable::inventoryNextItem() //Increment iterator to next item
{
    if ((*fInventoryPtr) == (fInventoryPtr->end()))
    {
        fInventoryPtr->end();

        //Don't do anything if at max end of inventory
    }

    else
    {
        ++(*fInventoryPtr);
    }

    for (int i = 0; i < 3; i++)
    {
        if (fInventoryPtr->getIndex() == i)
        {
            cout << " [" << fInventory[i] << "] ";
        }

        else
        {
            cout << " " << fInventory[i] << " ";
        }
    }
}
```

In Main.cpp (Test code)

```
int main()
{
    Playable* myPC = new Playable();
    myPC->addItem("item1");
    myPC->addItem("item2");
    myPC->addItem("item3");

    cout << "Showing inventory items: " << endl;
    myPC->showInventory(); // Show iventory items

    cout << endl << endl;
    cout << "Using inventoryPrevItem(): ";
    myPC->inventoryPrevItem();
    cout << endl << endl << "Using inventoryNextItem(): ";
    myPC->inventoryNextItem();
    cout << endl << endl << "Using inventoryNextItem(): ";
    myPC->inventoryNextItem();
    cout << endl << endl << "Using inventoryNextItem(): ";
    myPC->inventoryNextItem();
    cout << endl << endl << "Using inventoryPrevItem(): ";
    myPC->inventoryPrevItem();
    cout << endl << endl << "Using inventoryPrevItem(): ";
    myPC->inventoryPrevItem();


    return 0;
}
```

Console Output (Test code)

```
item1 added into inventory!
item2 added into inventory!
item3 added into inventory!
Showing inventory items:
 item1  item2  item3

Using inventoryPrevItem():  [item1]  item2  item3

Using inventoryNextItem():  item1  [item2]  item3

Using inventoryNextItem():  item1  item2  [item3]

Using inventoryNextItem():  item1  item2  [item3]

Using inventoryPrevItem():  item1  [item2]  item3
```

Troubleshooting

No errors were met during implementation of this segment.

References

StackOverflow, 2011, *creating an array of structs in c++*, StackOverflow, viewed 31 October 2021 < https://stackoverflow.com/questions/6810656/creating-an-array-of-structs-in-c>.

C# Corner, 2017, *What Is Struct And When To Use Struct In C#*, C# Corner, viewed 31 October 2021 <https://www.c-sharpcorner.com/article/what-is-structure-and-when-to-use-in-c-sharp/>.

StackOverflow, 2013, *Linked List vs Vector*, StackOverflow, viewed 31 October 2021 <https://stackoverflow.com/questions/19039972/linked-list-vs-vector>.

Appendix

// Main.cpp

```
#include <iostream>
#include <SFML/Audio.hpp>
#include "Entity.h"
#include "Character.h"
#include "Object.h"
#include "NonPlayable.h"
#include "Iterator.h"
#include "MapPiece.h"
#include "RiddleRecordList.h"

using namespace std;

int main()
{
	string userInput = "run";

	// String variables defined
	string msgPit = "Into the pit I fall! Die I shall.";
	string msgFear = "Something feels off.";
	string msgBreeze = "I feel a breeze...";
	string msgEnemy = "Oh no. There's a... sphinx.";

	string instruction = "Welcome to the MAZE, where all dreamers enter but few survive. Navigate through these corridors-- find the exit, and we'll let you go unharmed. Though of course, we love that if you could stay. But know that you are not alone within these walls... \n A note of advice: Try not to run, but face your fears head-on. Who knows, you might be going in the right direction.";
	string instruction2 = "\n GAME INSTRUCTIONS : \n 1) Navigate this maze using w, a, s, d keys to move up, left, down, right. \n 2) If you feel a breeze, there is a pit nearby. Don't fall into it, you'll die!";
	string instruction3 = " 3) If something feels off, there is an Sphinx nearby. You can choose to approach and answer it, attack or run. Some of them drop useful items when they die, be sure to pick them up.";
	string instruction4 = " 4) Other game controls, type: ";
	string instruction5 = " 'menu' to access the game menu\n 'stat' to view player stats \n 'bag' to access your inventory \n 'trace' to review your last five steps";
	string hint = " Remember: Try not to run, but face your fears head-on. Who knows, you might be going in the right direction.";

	string itemDubiousWeapon = "weapon";
	string itemHealthPotion = "potion";


	// Defining the walls in the maze, for each piece of the ma
	Walls CorridorH; // Horizontal corridor
	{
		CorridorH.LeftW = false;
		CorridorH.RightW = false;
		CorridorH.FrontW = true;
		CorridorH.BackW = true;
	}

	Walls CorridorV; // Vertical corridor
	{
		CorridorV.LeftW = true;
		CorridorV.RightW = true;
		CorridorV.FrontW = false;
		CorridorV.BackW = false;
	}

	Walls TopRightC; // Corner at top right
	{
		TopRightC.LeftW = false;
```

```
        TopRightC.RightW = true;
        TopRightC.FrontW = true;
        TopRightC.BackW = false;
}

Walls TopLeftC; // Corner at top left
{
        TopLeftC.LeftW = true;
        TopLeftC.RightW = false;
        TopLeftC.FrontW = true;
        TopLeftC.BackW = false;
}

Walls BottomRightC; // Corner at bottom right
{
        BottomRightC.LeftW = false;
        BottomRightC.RightW = true;
        BottomRightC.FrontW = false;
        BottomRightC.BackW = true;
}

Walls BottomLeftC; // Corner at bottom left
{
        BottomLeftC.LeftW = true;
        BottomLeftC.RightW = false;
        BottomLeftC.FrontW = false;
        BottomLeftC.BackW = true;
}

Walls LeftDeadEnd; // Dead end on the left
{
        LeftDeadEnd.LeftW = true;
        LeftDeadEnd.RightW = false;
        LeftDeadEnd.FrontW = true;
        LeftDeadEnd.BackW = true;
}

Walls RightDeadEnd; // Dead end on the right
{
        RightDeadEnd.LeftW = false;
        RightDeadEnd.RightW = true;
        RightDeadEnd.FrontW = true;
        RightDeadEnd.BackW = true;
}

Walls FrontDeadEnd; // Dead end ahead in front
{
        FrontDeadEnd.LeftW = true;
        FrontDeadEnd.RightW = true;
        FrontDeadEnd.FrontW = true;
        FrontDeadEnd.BackW = false;
}

Walls BackDeadEnd; // Dead end behind
{
        BackDeadEnd.LeftW = true;
        BackDeadEnd.RightW = true;
        BackDeadEnd.FrontW = false;
        BackDeadEnd.BackW = true;
}

Walls OnlyLeftW; // Wall on the left
{
        OnlyLeftW.LeftW = true;
        OnlyLeftW.RightW = false;
```

```
                OnlyLeftW.FrontW = false;
                OnlyLeftW.BackW = false;
        }

        Walls OnlyRightW; // Wall on the right
        {
                OnlyRightW.LeftW = false;
                OnlyRightW.RightW = true;
                OnlyRightW.FrontW = false;
                OnlyRightW.BackW = false;
        }

Walls OnlyFrontW; // Wall ahead in front
        {
                OnlyFrontW.LeftW = false;
                OnlyFrontW.RightW = false;
                OnlyFrontW.FrontW = true;
                OnlyFrontW.BackW = false;
        }

        Walls OnlyBackW; // Wall behind
        {
                OnlyBackW.LeftW = false;
                OnlyBackW.RightW = false;
                OnlyBackW.FrontW = false;
                OnlyBackW.BackW = true;
        }

        Walls NoWalls; // No walls
        {
                NoWalls.LeftW = false;
                NoWalls.RightW = false;
                NoWalls.FrontW = false;
                NoWalls.BackW = false;
        }

        // Objecct pointers defined
        Object* objectPtr = nullptr; // Pointer to game objects
        NonPlayable* NPCPtr = nullptr; // Pointer to game NPCS
        Playable* myPC = nullptr; // Pointer to PC
        RiddleRecordList* RiddleRecordPtr = nullptr; // Pointer to the Singly-linked Riddle list

// Generating Game Map
        MapPiece* myMap[10][10]; // Creating an array of pointers to "MapPieces"- forms the game map

// Declaring each element of the map array- forming each part of the map, and declaring what they contain (what kind
of walls, what kind of game object, what kind of NPC)
        // Column 0
        {
                myMap[0][0] = new MapPiece(BackDeadEnd);
                myMap[0][1] = new MapPiece(TopLeftC);
                myMap[0][2] = new MapPiece(BottomLeftC);
                myMap[0][3] = new MapPiece(CorridorV);
                myMap[0][4] = new MapPiece(FrontDeadEnd);

                objectPtr = new Object("Pit", "Pit0,5", 0, 5, true);
                objectPtr->setMessage(msgPit);
                myMap[0][5] = new MapPiece(BackDeadEnd, *objectPtr);

                objectPtr = new Object("Breeze", "Breeze0,6", 0, 6, false);
                objectPtr->setMessage(msgBreeze);
                myMap[0][6] = new MapPiece(CorridorV, *objectPtr);

                myMap[0][7] = new MapPiece(OnlyLeftW);
```

```
		myMap[0][8] = new MapPiece(OnlyLeftW);
		myMap[0][9] = new MapPiece(TopLeftC);
}

// Column 1
{
		objectPtr = new Object("Pit", "Pit1,0", 1, 0, true);
		objectPtr->setMessage(msgPit);
		myMap[1][0] = new MapPiece(LeftDeadEnd, *objectPtr);

		myMap[1][1] = new MapPiece(BottomRightC);

		objectPtr = new Object("Fear", "Fear1,2", 1, 2, false);
		objectPtr->setMessage(msgFear);
		myMap[1][2] = new MapPiece(NoWalls, *objectPtr);

		NPCPtr = new NonPlayable("Sphinx", "Sphinx1,3", true, 10, 1, 3, false);
		NPCPtr->setMessage(msgEnemy);
		NPCPtr->setWillDropItem(true);
		NPCPtr->setItemHeld(itemDubiousWeapon);
		myMap[1][3] = new MapPiece(CorridorV, *NPCPtr);

		objectPtr = new Object("Fear", "Fear1,4", 1, 4, false);
		objectPtr->setMessage(msgFear);
		myMap[1][4] = new MapPiece(OnlyLeftW, *objectPtr);

		objectPtr = new Object("Breeze", "Breeze1,5", 1, 5, false);
		objectPtr->setMessage(msgBreeze);
		myMap[1][5] = new MapPiece(OnlyLeftW, *objectPtr);

		myMap[1][6] = new MapPiece(FrontDeadEnd);
		myMap[1][7] = new MapPiece(CorridorH);
		myMap[1][8] = new MapPiece(CorridorH);
		myMap[1][9] = new MapPiece(CorridorH);
}

// Column 2
{
		objectPtr = new Object("Breeze", "Breeze2,0", 2, 0, false);
		objectPtr->setMessage(msgBreeze);
		myMap[2][0] = new MapPiece(OnlyBackW, *objectPtr);

		myMap[2][1] = new MapPiece(CorridorV);
		myMap[2][2] = new MapPiece(OnlyRightW);
		myMap[2][3] = new MapPiece(TopLeftC);
		myMap[2][4] = new MapPiece(CorridorH);

		objectPtr = new Object("Pit", "Pit2,5", 2, 5, true);
		objectPtr->setMessage(msgPit);
		myMap[2][5] = new MapPiece(RightDeadEnd, *objectPtr);

		myMap[2][6] = new MapPiece(BottomLeftC);

		objectPtr = new Object("Fear", "Fear2,7", 2, 7, false);
		objectPtr->setMessage(msgFear);
		myMap[2][7] = new MapPiece(OnlyFrontW, *objectPtr);

		objectPtr = new Object("Breeze", "Breeze2,8", 2, 8, false);
		objectPtr->setMessage(msgBreeze);
		myMap[2][8] = new MapPiece(CorridorH, *objectPtr);

		myMap[2][9] = new MapPiece(CorridorH);
}

// Column 3
```

```
{
	myMap[3][0] = new MapPiece(CorridorH);

	NPCPtr = new NonPlayable("Sphinx", "Sphinx3,1", true, 10, 3, 1, false);
	NPCPtr->setMessage(msgEnemy);
	myMap[3][1] = new MapPiece(BottomLeftC, *NPCPtr);

	objectPtr = new Object("Fear", "Fear3,2", 3, 2, false);
	objectPtr->setMessage(msgFear);
	myMap[3][2] = new MapPiece(OnlyLeftW, *objectPtr);

	objectPtr = new Object("Breeze", "Breeze3,3", 3, 3, false);
	objectPtr->setMessage(msgBreeze);
	myMap[3][3] = new MapPiece(OnlyFrontW, *objectPtr);

	objectPtr = new Object("Fear", "Fear3,4", 3, 4, false);
	objectPtr->setMessage(msgFear);
	myMap[3][4] = new MapPiece(OnlyBackW, *objectPtr);

	myMap[3][5] = new MapPiece(TopLeftC);
	myMap[3][6] = new MapPiece(CorridorH);

	NPCPtr = new NonPlayable("Sphinx", "Sphinx3,7", true, 10, 3, 7, false);
	NPCPtr->setMessage(msgEnemy);
	NPCPtr->setWillDropItem(true);
	NPCPtr->setItemHeld(itemDubiousWeapon);
	myMap[3][7] = new MapPiece(CorridorH, *NPCPtr);

	objectPtr = new Object("Pit", "Pit3,8", 3, 8, true);
	objectPtr->setMessage(msgPit);
	myMap[3][8] = new MapPiece(RightDeadEnd, *objectPtr);

	myMap[3][9] = new MapPiece(CorridorH);
}

// Column 4
{
	myMap[4][0] = new MapPiece(CorridorH);

	objectPtr = new Object("Fear", "Fear4,1", 4, 1, false);
	objectPtr->setMessage(msgFear);
	myMap[4][1] = new MapPiece(CorridorH, *objectPtr);

	myMap[4][2] = new MapPiece(CorridorH);

	objectPtr = new Object("Pit", "Pit4,3", 4, 3, true);
	objectPtr->setMessage(msgPit);
	myMap[4][3] = new MapPiece(RightDeadEnd, *objectPtr);

	NPCPtr = new NonPlayable("Sphinx", "Sphinx4,4", true, 10, 4, 4, false);
	NPCPtr->setMessage(msgEnemy);
	NPCPtr->setWillDropItem(true);
	NPCPtr->setItemHeld(itemHealthPotion);
	myMap[4][4] = new MapPiece(CorridorH, *NPCPtr);

	myMap[4][5] = new MapPiece(CorridorH);
	myMap[4][6] = new MapPiece(CorridorH);

	objectPtr = new Object("Fear", "Fear4,7", 4, 7, false);
	objectPtr->setMessage(msgFear);
	myMap[4][7] = new MapPiece(CorridorH, *objectPtr);

	myMap[4][8] = new MapPiece(BackDeadEnd);
	myMap[4][9] = new MapPiece(TopRightC);
}
```

```
	// Column 5
	{
		myMap[5][0] = new MapPiece(BottomRightC);

		objectPtr = new Object("Fear", "Fear5,1", 5, 1, false);
		objectPtr->setMessage(msgFear);
		myMap[5][1] = new MapPiece(OnlyFrontW, *objectPtr);

		myMap[5][2] = new MapPiece(BottomRightC);
		myMap[5][3] = new MapPiece(CorridorV);

		objectPtr = new Object("Fear", "Fear5,4", 5, 4, false);
		objectPtr->setMessage(msgFear);
		myMap[5][4] = new MapPiece(TopRightC, *objectPtr);

		myMap[5][5] = new MapPiece(CorridorH);

		objectPtr = new Object("Fear", "Fear5,6", 5, 6, false);
		objectPtr->setMessage(msgFear);
		myMap[5][6] = new MapPiece(CorridorH, *objectPtr);

		myMap[5][7] = new MapPiece(BottomRightC);
		myMap[5][8] = new MapPiece(CorridorV);
		myMap[5][9] = new MapPiece(TopLeftC); // EXIT
	}

// Column 6
	{
		myMap[6][0] = new MapPiece(LeftDeadEnd);

		NPCPtr = new NonPlayable("Sphinx", "Sphinx6,1", true, 10, 6, 1, false);
		NPCPtr->setMessage(msgEnemy);
		myMap[6][1] = new MapPiece(OnlyBackW, *NPCPtr);

		objectPtr = new Object("Fear", "Fear6,2", 6, 2, false);
		objectPtr->setMessage(msgFear);
		myMap[6][2] = new MapPiece(CorridorV, *objectPtr);

		myMap[6][3] = new MapPiece(CorridorV);
		myMap[6][4] = new MapPiece(CorridorV);
		myMap[6][5] = new MapPiece(TopRightC);

		NPCPtr = new NonPlayable("Sphinx", "Sphinx6,6", true, 10, 6, 6, false);
		NPCPtr->setMessage(msgEnemy);
		NPCPtr->setWillDropItem(true);
		NPCPtr->setItemHeld(itemDubiousWeapon);
		myMap[6][6] = new MapPiece(CorridorH, *NPCPtr);

		myMap[6][7] = new MapPiece(LeftDeadEnd);
		myMap[6][8] = new MapPiece(BottomLeftC);
		myMap[6][9] = new MapPiece(OnlyFrontW);
	}

	// Column 7
	{
		myMap[7][0] = new MapPiece(CorridorH);

		objectPtr = new Object("Fear", "Fear7,1", 7, 1, false);
		objectPtr->setMessage(msgFear);
		myMap[7][1] = new MapPiece(BottomRightC, *objectPtr);

		myMap[7][2] = new MapPiece(OnlyLeftW);
		myMap[7][3] = new MapPiece(CorridorV);
```

```
		myMap[7][4] = new MapPiece(OnlyLeftW);
		myMap[7][5] = new MapPiece(CorridorV);

		objectPtr = new Object("Fear", "Fear7,6", 7, 6, false);
		objectPtr->setMessage(msgFear);
		myMap[7][6] = new MapPiece(OnlyRightW, *objectPtr);

		myMap[7][7] = new MapPiece(TopRightC);

		objectPtr = new Object("Breeze", "Breeze7,8", 7, 8, false);
		objectPtr->setMessage(msgBreeze);
		myMap[7][8] = new MapPiece(CorridorH, *objectPtr);

		myMap[7][9] = new MapPiece(CorridorH);
}

// Column 8
{
		objectPtr = new Object("Breeze", "Breeze8,0", 8, 0, false);
		objectPtr->setMessage(msgBreeze);
		myMap[8][0] = new MapPiece(OnlyBackW, *objectPtr);

		objectPtr = new Object("Pit", "Pit8,1", 8, 1, true);
		objectPtr->setMessage(msgPit);
		myMap[8][1] = new MapPiece(FrontDeadEnd, *objectPtr);

		myMap[8][2] = new MapPiece(CorridorH);
		myMap[8][3] = new MapPiece(LeftDeadEnd);
		myMap[8][4] = new MapPiece(OnlyBackW);
		myMap[8][5] = new MapPiece(CorridorV);
		myMap[8][6] = new MapPiece(CorridorV);
		myMap[8][7] = new MapPiece(TopLeftC);

		objectPtr = new Object("Pit", "Pit8,8", 8, 8, true);
		objectPtr->setMessage(msgPit);
		myMap[8][8] = new MapPiece(RightDeadEnd, *objectPtr);

		myMap[8][9] = new MapPiece(CorridorH);
}

// Column 9
{
		objectPtr = new Object("Fear", "Fear9,0", 9, 0, false);
		objectPtr->setMessage(msgFear);
		myMap[9][0] = new MapPiece(BottomRightC, *objectPtr);

		NPCPtr = new NonPlayable("Sphinx", "Sphinx9,1", true, 10, 9, 1, false);
		NPCPtr->setMessage(msgEnemy);
		NPCPtr->setWillDropItem(true);
		NPCPtr->setItemHeld(itemHealthPotion);
		myMap[9][1] = new MapPiece(CorridorV, *NPCPtr);

		objectPtr = new Object("Fear", "Fear9,2", 9, 2, false);
		objectPtr->setMessage(msgFear);
		myMap[9][2] = new MapPiece(OnlyRightW, *objectPtr);

		myMap[9][3] = new MapPiece(OnlyRightW);

		objectPtr = new Object("Breeze", "Breeze9,4", 9, 4, false);
		objectPtr->setMessage(msgBreeze);
		myMap[9][4] = new MapPiece(OnlyRightW, *objectPtr);

		objectPtr = new Object("Pit", "Pit9,5", 9, 5, true);
		objectPtr->setMessage(msgPit);
		myMap[9][5] = new MapPiece(FrontDeadEnd, *objectPtr);
```

```
                myMap[9][6] = new MapPiece(BackDeadEnd);
                myMap[9][7] = new MapPiece(OnlyRightW);
                myMap[9][8] = new MapPiece(CorridorV);
                myMap[9][9] = new MapPiece(TopRightC);
        }

        // Generating Riddle List- this list will be cycled through in the game-> NPCs will access this list to obtain their riddles to ask the PC
        RiddleRecordPtr = new RiddleRecordList();
        RiddleRecordPtr->addRiddle("What do you call a fish with no eyes?", "Myxine Glutinosa", "Blind", "I dunno... fsh?", 2);
        RiddleRecordPtr->addRiddle("At night they come without being fetched, and by day they are lost without being stolen. What are they?", "The stars", "Dreams", "Time", 0);
        RiddleRecordPtr->addRiddle("What is the meaning of life?", "Whatever which we choose to give it", "There's none", "E", 2);
        RiddleRecordPtr->addRiddle("Why is a raven like a writing desk?", "They're both not made of cheese", "Because Poe wrote on both", "I haven't the slightest idea", 2);
        RiddleRecordPtr->addRiddle("Why did the chicken cross the road?", "To get to the other side", "There was a car coming", "I don't know. Why?", 0);

        // SFML implementation
                //Loading a SoundBuffer object where the background audio is stored in
        sf::SoundBuffer buffer1;
        if (!buffer1.loadFromFile("HyruleCastle.wav"))
        {
                return -1;
        }



sf::Sound soundBackground;
        soundBackground.setBuffer(buffer1);

        //Loading annother SoundBuffer object to store the enemy audio
        sf::SoundBuffer buffer2;
        if (!buffer2.loadFromFile("Enemy.wav"))
        {
                return -1;
        }

        sf::Sound soundEnemy;
        soundEnemy.setBuffer(buffer2);

        // Game start
        cout << "WELCOME TO THE MAZE." << endl;
        cout << "======================" << endl;
        cout << "Press any key to start..." << endl;

        string anykey = "";
        cin >> anykey;

        soundBackground.play(); // Play background sound
        soundBackground.setLoop(true); // Set the audio to loop in the background

        cout << endl << "What is your name?:" << endl;
        cout << "Input: ";
        string myname = "";
        cin >> myname;

        myPC = new Playable(myname, "0001", true, 15, 0, 0); // Create a PC on the heap

// Some game description to the player
        cout << endl << "Welcome " << myname << ". Let us begin..." << endl << endl;
```

```
	// Cout game instructions
	cout << instruction << endl << instruction2 << endl << instruction3 << endl << instruction4 << endl << instruction5 << endl << hint << endl;
	cout << "=====================================================" << endl << endl;

	cout << "Press 'w' to take your first step!" << endl;

	// Game runtime loop:
	do
	{
		if ((myPC->getPosition().X == 5) & (myPC->getPosition().Y == 9)) // if PC position is at the end goal/exit of (5,9)
		{
			myPC->setMessage("I can see the light! This is it! I have reached the exit!");
			cout << *(myPC) << endl;

			userInput = "win game"; // End the game
		}

		else // else the game continues to take in an input
		{
			cout << "Input: ";
			cin >> userInput;
		}


		if (userInput == "w")  // Go UP
		{
			if (myMap[myPC->getPosition().X][myPC->getPosition().Y]->getWalls().FrontW == false) // If there is no wall in front
			{
				myPC->moveUp(1); // Proceed to move up by one step
			}
			else
			{
				cout << "Can't go up. There's a wall blocking me." << endl;
			}

			cout << "Current position :" << myPC->getPosition().X << ", " << myPC->getPosition().Y << endl; // Cout where the PC is
			userInput = "run"; // Force user input to continue in the loop
		}

		if (userInput == "a") // Go LEFT
		{
			if (myMap[myPC->getPosition().X][myPC->getPosition().Y]->getWalls().LeftW == false) // If there is no wall on left
			{
				myPC->moveLeft(1); // Proceed to move left by one step
			}
			else
			{
				cout << "Can't go left. There's a wall blocking me." << endl;
			}

			cout << "Current position :" << myPC->getPosition().X << ", " << myPC->getPosition().Y << endl; // Cout where the PC is
			userInput = "run";
		}

		if (userInput == "d") // Go RIGHT
		{
			if (myMap[myPC->getPosition().X][myPC->getPosition().Y]->getWalls().RightW == false) // If there is no wall on = right
```

```
                {
                        myPC->moveRight(1); // Proceed to move right by one step
                }
                else
                {
                        cout << "Can't go right. There's a wall blocking me." << endl;
                }

                cout << "Current position :" << myPC->getPosition().X << ", " <<
myPC->getPosition().Y << endl; // Cout where the PC is
                userInput = "run";
        }

        if (userInput == "s") // Go DOWN
        {
                if (myMap[myPC->getPosition().X][myPC->getPosition().Y]->getWalls().BackW ==
false) // If there is no wall behind
                {
                        myPC->moveDown(1); // Proceed to move down one step
                }
                else
                {
                        if ((myPC->getPosition().X == 0) & (myPC->getPosition().Y == 0)) // If at
starting position
                        {
                                myPC->setMessage("This is where I started from... I musn't go
backwards."); // Do not move out of map
                                cout << *(myPC) << endl;
                        }
                        else
                        {
                                cout << "Can't go down. There's a wall blocking me." << endl;
                        }

                }
                cout << "Current position :" << myPC->getPosition().X << ", " <<
myPC->getPosition().Y << endl; // Cout where the PC is
                userInput = "run";
        }

        if (userInput == "view")  // View the last five steps made by PC
        {
                cout << endl << "=======================" << endl;
                myPC->viewLastSteps(5);
                cout << endl << "=======================" << endl;

        }



if ((myMap[myPC->getPosition().X][myPC->getPosition().Y]->getObject() != nullptr) ||
(myMap[myPC->getPosition().X][myPC->getPosition().Y]->getNPC() != nullptr))
        { // If the PC is at a piece of the map that has a object or NPC (both enemies)

                if (soundEnemy.getStatus() != 2) // When the enemy audio is not playing
                {
                        soundBackground.stop(); // Stop the background audio
                        soundEnemy.play(); // Play the enemy audio
                }
        }

        else // If the PC is at a piece of the map without any NPPC enemies/objects that can induce death
        {
                soundEnemy.stop(); // Stop the enemy music
```

```
                        if (soundBackground.getStatus() != 2) // When the background audio is not playing
                        {
                                soundBackground.play(); // Play the background audio
                        }
                }

                if (myMap[myPC->getPosition().X][myPC->getPosition().Y]->getObject() != nullptr) // If the PC
is in the same position as a game object
                {
                        if
(myMap[myPC->getPosition().X][myPC->getPosition().Y]->getObject()->getInduceDeath() == true) // If the object
can induce death
                        {
                                myPC->Die(); // PC will die
                        }
                        cout << *(myMap[myPC->getPosition().X][myPC->getPosition().Y]->getObject()); //
Cout message of object
                }

                if (myMap[myPC->getPosition().X][myPC->getPosition().Y]->getNPC() != nullptr) // If the PC is
in the same position as an NPC (meeting an enemy)
                {
                        NPCPtr = (myMap[myPC->getPosition().X][myPC->getPosition().Y]->getNPC()); //
Point NPC pointer at current NPC

                        if (NPCPtr->getIsDead() == false) // If the NPC is not dead
                        {
                                cout << endl;
                                cout << *(NPCPtr); // Cout NPC message
                                cout << endl;
                                string userInputMeetEnemy = "";
                                cout << "What do I do?" << endl << "1) Approach with caution" << endl << "2)
Attack (-4 HP, if without weapon)" << endl << "3) Run (-3HP)";
                                cout << endl;
                                cout << endl << "Input: ";
                                cin >> userInputMeetEnemy;

                                if (userInputMeetEnemy == "1") // PC chooses to approach enemy
                                {
                                        NPCPtr->setMessage(RiddleRecordPtr->showRiddleAndChoices()); //
NPC gets a riddle from the riddle list

                                        cout << "Sphinx says: " << endl;
                                        cout << *(NPCPtr); // Outputs the question
                                        cout << endl;
                                        cout << "Input: ";
                                        int userInputRiddle;

                                        cin >> userInputRiddle; // PC inputs answer choice

                                        if ((userInputRiddle - 1) != RiddleRecordPtr->correctAns()) // If
answer choice is wrong
                                        {
                                                myPC->setMessage("Oh no, I'm wrong. RUN. (-3HP)");
                                                cout << *(myPC) << endl;

                                                myPC->decreaseHP(3); // Decrease PC HP by 3
                                        }

                                        else // If answer choice is correct
                                        {
                                                NPCPtr->setMessage("Sphinx says: You are right. You may
pass.");

                                                cout << *(NPCPtr);
                                        }
```

```
                                        RiddleRecordPtr->NextRiddle(); // Choose the next riddle
                                        NPCPtr->setMessage(msgEnemy); // And reset the message for the
next enemy
                                }

                                else if (userInputMeetEnemy == "2") // PC chooses to Attack enemy
                                {
                                        if ((myPC->getHoldingItem() == true) & (myPC->getItemHeld() ==
itemDubiousWeapon)) // If PC is holding an item 'weapon'
                                        {
                                                NPCPtr->Die(); // The NPC dies
                                                cout << "The Sphinx died." << endl;
                                                myPC->dropItem(); // The PC drops the wepon- weapons are
single use only

                                                string droppeditem = NPCPtr->DropItem(); // NPC drops an
item. Store dropped item into variable

                                                cout << "The Sphinx dropped " << droppeditem << "! Put to
inventory? (yes/no)" << endl; // Ask if PC wants to take dropped item
                                                string inputChoice = "";

                                                cin >> inputChoice;

                                                if (inputChoice == "yes") // If PC chooses to take dropped
item
                                                {
                                                        myPC->addItem(droppeditem); // Add it into
inventory. If inventory is full it will automatically discard it
                                                        inputChoice = "no";
                                                }
                                        }

                                        else // If PC is not holding an item but still chooses to attack the NPC
                                        {
                                                NPCPtr->decreaseHP(NPCPtr->getMaxHP() / 2); // Decrease
NPC health by half
                                                {
                                                        if (NPCPtr->getCurrentHP() <= 0) // If NPC current
HP is zero or less
                                                        {
                                                                NPCPtr->Die(); // Then it dies
                                                                cout << "The Sphinx died." << endl;

                                                                string droppeditem =
NPCPtr->DropItem(); // NPC drops an item. Store dropped item into variable

                                                                cout << "The Sphinx dropped " <<
droppeditem << "! Put to inventory? (yes/no)" << endl; // Ask if PC wants to take dropped item
                                                                string inputChoice = "";

                                                                cin >> inputChoice;

                                                                if (inputChoice == "yes") // If PC chooses
to take dropped item
                                                                {
                                                                        myPC->addItem(droppeditem);
// Add it into inventory. If inventory is full it will automatically discard it
                                                                        inputChoice = "no";
                                                                }
                                                        }

                                                        else // If NPC current HP is not yet zero
                                                        {
```

```
                                                                myPC->decreaseHP(4); // Decrease PC
HP by 4
                                                                myPC->setMessage("Oh nawh, he's not
dead yet... RUN. (-4HP)");
                                                                cout << *(myPC);
                                                        }
                                                }
                                        }
                                }

                                else // If PC chooses to run from the Enemy
                                {
                                        myPC->decreaseHP(3); // Decrease PC HP by 3
                                        myPC->setMessage("Nope, imma run. (-3HP)");
                                        cout << *(myPC);
                                }
                        }

                        else // If the NPC is dead
                        {
                                NPCPtr->setMessage("Oh... the Sphinx is dead. Man, sure feels unsettling
around here.");
                                cout << *(NPCPtr);
                                userInput = "run";
                        }
                        userInput = "run"; // Force userInput to be back in loop
                }

                if (userInput == "bag") // View PC inventory
                {
                        cout << endl << endl;
                        cout << "Opening inventory:" << endl;
                        myPC->showInventory(); // Show iventory items
                        cout << endl;
                        cout << endl;

                        cout << "Select item? (yes/no)" << endl;
                        string inputChoice = "";
                        string inputScroll = "";

                        cin >> inputChoice;

                        if (inputChoice == "yes") // If PC chooses to select items
                        {
                                cout << "Scroll inventory: (a/d). Press 's' to use item, 'w' to drop item, press 'x' to
close inventory." << endl;
                                myPC->showInventory();



do
                                {
                                        cout << endl;
                                        cout << "Input: ";
                                        cin >> inputScroll;

                                        if (inputScroll == "a") // Scroll left
                                        {
                                                myPC->inventoryPrevItem(); // Iterate backwards
                                        }

                                        if (inputScroll == "d") // Scroll right
                                        {
                                                myPC->inventoryNextItem(); // Iterate forwards
```

```
                                        }

                                        if (inputScroll == "s") // Select Item
                                        {
                                                myPC->takeItem(); // PC to take item in hand

                                                if (myPC->getItemHeld() == itemHealthPotion) // If the item
is a health potion
                                                {
                                                        string usePotion = "";

                                                        cout << "Use health potion? (+4HP) (yes/no)" <<
endl;
                                                        cin >> usePotion;

                                                        if (usePotion == "yes") // If user chooses to use
health potion
                                                        {
                                                                myPC->increaseHP(4); // PC health
increases by 4
                                                                myPC->dropItem(); // Potion is discarded
                                                                cout << "Health increased by 4 HP!" <<
endl;
                                                        }

                                                        else // If user chooses to not use health potion
                                                        {
                                                                myPC->addItem(itemHealthPotion); //
Store item back into inventory
                                                        }
                                                }

                                                myPC->showInventory(); // Show inventory contents
                                        }

                                        if (inputScroll == "w") // Drop item
                                        {
                                                if (myPC->getHoldingItem() == true) // If PC is currently
holding an item
                                                {
                                                        cout << myPC->getItemHeld() << " has been
dropped!" << endl;
                                                        myPC->dropItem();
                                                }
                                        }

                                        if (inputScroll == "x") // Close the inventory
                                        {
                                                inputChoice = "no";
                                        }

                                } while (inputChoice == "yes");
                        }

                        else // Close the inventory
                        {
                                cout << "Inventory closed." << endl;
                        }
                }

                if (userInput == "stats") // Show player stats: name, id, position, HP etc
                {
                        cout << endl;
                        cout << "====================" << endl;
```

```
                cout << "Player name: " << myPC->getName() << "(ID: " << myPC->getID() << ")" <<
endl;
                cout << "Current Position: (" << myPC->getPosition().X << "," <<
myPC->getPosition().Y << ")" << endl;
                cout << "HP:" << myPC->getCurrentHP() << "/" << myPC->getMaxHP() << endl;
                cout << "Total number steps taken: " << myPC->getStepsTaken() << endl;

                cout << "Item held: " << myPC->getItemHeld() << endl;
                cout << endl;

                cout << "Type 'trace' to trace back the last 5 steps taken." << endl;
                cout << "Type 'bag' to view inventory contents" << endl;
                cout << "====================" << endl;
                cout << endl;
        }

        if (userInput == "trace") // Trace back the ast five steps made by PC
        {
                cout << "====================" << endl;
                myPC->viewLastSteps(5);
                cout << "====================" << endl;
                cout << endl;
        }

        if ((myPC->getCurrentHP() <= 0) || (myPC->getIsDead() == true)) // If NPC dies
        {
                cout << "You died... game over. Do you want to retry? (yes/no)" << endl;
                string inputChoice = "";
                cout << "Input: ";
                cin >> inputChoice;

                if (inputChoice == "yes")
                {
                        userInput = "e";
                }

                else // Exit program
                {
                        userInput = "exit game";
                }
        }


        if (userInput == "win game") // If user wins the game (successfully reach exit)
        {
                cout << "Congratulations, you have survived the maze!" << endl << "Total steps taken: "
<< myPC->getStepsTaken() << endl << endl << "Replay ? (yes / no)" << endl;
                cin >> userInput;

                if (userInput == "yes") // If user chooses to replay
                {
                        // restart
                }

                else // If user chooses to exit
                {
                        userInput = "exit game"; // Exit game
                }

        }

        if (userInput == "menu") // Access main menu
        {
                cout << "MAIN MENU:" << endl;
                cout << "Choose... (Press 'x' or any key to close)" << endl;
```

```
                        cout << "1) Help" << endl;
                        cout << "2) Exit" << endl;
                        cout << "3) Restart" << endl;
                        cout << "Input: " << endl;

                        string userChoice;
                        cin >> userChoice;

                        if (userChoice == "1") // Show game help (intructions)
                        {
                                cout << "=======================================" << endl;
                                cout << instruction2 << endl << instruction3 << endl << instruction4 << endl
<< instruction5 << endl << endl << hint << endl;
                                cout << "=======================================" << endl;

                                userInput = "run";
                        }

                        if (userChoice == "2") // Exit game
                        {
                                userInput = "exit game";
                        }

                        else if (userChoice == "3") // Restart game
                        {
                                //restart
                        }

                        else // Close main menu
                        {
                                userInput = "run";
                        }
                }

                if (userInput == "exit game") // Exit game
                {
                        cout << "Are you sure you want to exit? (yes/no)" << endl;
                        cin >> userInput;

                        if (userInput == "yes") // If user confirms to exit
                        {
                                userInput = "end"; // End game
                        }

                        else
                        {
                                userInput = "run"; // Go back to game loop
                        }

                }

        } while (userInput != "end");

        cout << "Exiting Program..." << endl;


        // Delete objects created on heap (memory management)
        delete myPC;
        delete RiddleRecordPtr;
        int fRow = 10;
        int fCol = 10;

        for (int i = 0; i < fRow; i++)
        {
                for (int j = 0; j < fCol; j++)
```

```
                {
                        delete myMap[i][j];
                }
        }
        return 0;
}
```

// Entity.h

```
#pragma once


#include <iostream>
#include <string>

using namespace std;

struct Position
{
        int X, Y;
};

class Entity
{

private:

protected:
        string fName; // Character name
        string fID; // Character ID
        Position fPosition; // (x, y) Position of character

        string fMessage; // Message dialogue

public:
        Entity(); //Default constructor
        Entity(string name, string id, int posx, int posy);


        //Declare Setters
        void setName(string name); // Setter for fName
        void setID(string id); // Setter for fIDS
        void setPositionX(int x); // Setter for x position
        void setPositionY(int y); // Setter for y position

        void setMessage(string msg); // Setter for fMessage


        //Declare Getters
        string getName(); // Getter for fName
        string getID(); // Getter for fID
        Position getPosition(); // Getter for x and y position

        string getMessage(); // Getter for fMessage

        //Friend Operator returns input stream
        friend istream& operator>>(istream& aIstream, Entity& aUnit);

        //Friend Operator returns output stream
        friend ostream& operator<<(ostream& aOstream, Entity& aUnit);

        void listen(string msg);
```

```
        void tell();


        //Destructor
        virtual ~Entity();
};
```

// Entity.cpp

```
#include "Entity.h"
#include <iostream>
#include <string>

using namespace std;

Entity::Entity() //Default constructor
{
        fName = "Player01";
        fID = "L00000000";
        fPosition.X = 0; // x position
        fPosition.Y = 0; // y position


        fMessage = "";
}

Entity::Entity(string name, string id, int posx, int posy)
{
        fName = name;
        fID = id;
        fPosition.X = posx; // x position
        fPosition.Y = posy; // y position

        fMessage = "";
}


// Declare Setters

void Entity::setName(string owner) // Setter for fName
{
        fName = owner;
}

void Entity::setID(string id) // Setter for fID
{
        fID = id;
}


void Entity::setPositionX(int x) // Setter for x position
{
        fPosition.X = x;
}

void Entity::setPositionY(int y) // Setter for y position
{
        fPosition.Y = y;
}

void Entity::setMessage(string msg) // Setter for fMessage
```

```
{
	fMessage = msg;
}


//Declare Getters

string Entity::getName() // Getter for name
{
	return fName;
}

string Entity::getID() // Getter for ID
{
	return fID;
}

Position Entity::getPosition() // Getter for x and y position
{
	return fPosition;
}


string Entity::getMessage() // Getter for fMessage
{
	return fMessage;
}


void Entity::listen(string msg)
{
	fMessage = msg;
}

void Entity::tell()
{
	cout << fMessage << endl;
}


// Friend Operator returns input stream
istream& operator>>(istream& aIstream, Entity& aUnit)
{
	getline(aIstream, aUnit.fMessage); //get a line of string
	return aIstream;
}

// Friend Operator returns output stream
ostream& operator<<(ostream& aOstream, Entity& aUnit)
{
	aOstream << aUnit.fMessage << endl;
	return aOstream;
}


// Declare Destructor
Entity::~Entity()
{
}
```

// Object.h

```
#pragma once
#include "Entity.h"

#include <iostream>
#include <string>

using namespace std;

class Object : public Entity
{
private:
protected:
	bool fInduceDeath;

public:
	Object();
	Object(string name, string id, int posx, int posy, bool cancausedeath);

	//Declare Setters
	void setInduceDeath(bool choice);

	//Declare Getters
	bool getInduceDeath();

	~Object();
};
```

// Object.cpp

```
#include "Object.h"
#include <iostream>
#include <string>

using namespace std;

Object::Object()
{
	fInduceDeath = false;
}

Object::Object(string name, string id, int posx, int posy, bool cancausedeath) :Entity(name, id, posx, posy)
{
	fInduceDeath = cancausedeath;
}


// Declare Setters
void Object::setInduceDeath(bool choice)
{
	fInduceDeath = choice;
}

// Declare Getters
bool Object::getInduceDeath()
{
	return fInduceDeath;
}

Object::~Object()
{}
```

// Character.h

```
#pragma once
#include <iostream>
#include <string>
#include "Entity.h"

using namespace std;

class Character : public Entity
{
private:
protected:
	int fMaxHP; // Max healthpoint
	int fCurrentHP; // Current healthpoint
	bool fCanDie; // Can the character die?
	bool fIsDead; // Is character dead? (will be = true if fCurrentHP = 0
	                                    // if fCanDie = false, fIsDead is permanently = false)

	//When char fight decrease hp by half

public:
	Character();
	Character(string name, string id, bool candie, int maxhp, int posx, int posy);


	//Declare Setters
	void setMaxHP(int maxhp); // Setter for fMaxHP
	void setCurrentHP(int currenthp); // Setter for fCurrentHP

	void setCanDie(bool candie); // Setter for fCanDie
	void setIsDead(bool isdead); // Setter for fIsDead


	//Declare Getters
	int getMaxHP(); // Getter for fMaxHP
	int getCurrentHP(); // Getter for fCurrentHP

	bool getCanDie(); // Getter for fCanDie
	bool getIsDead(); // Getter for fIsDead

	//Declare functions
	void increaseHP(int increaseby); // Function to increase HP
	void decreaseHP(int decreaseby); // Function to decrease HP

	virtual void Die();

	virtual void moveLeft(int x); // Move left by x amount
	virtual void moveRight(int x); // Move right by x amount
	virtual void moveUp(int y); // Move up by y amount
	virtual void moveDown(int y); // Move down by y amount

	//Destructor
	virtual ~Character();

};
```

// Character.cpp

```
#include "Character.h"
#include <iostream>
```

```
Character::Character()
{
	fMaxHP = 100;
	fCurrentHP = 100; // Character will start off with full health

	fCanDie = true;
	fIsDead = false; // Character will start off being not dead
}

Character::Character(string name, string id, bool candie, int maxhp, int posx, int posy):Entity(name, id, posx, posy)
{
	fMaxHP = maxhp;
	fCurrentHP = maxhp; // Character will start off with full health

	fCanDie = candie;
	fIsDead = false; // Character will start off being not dead
}


// Declare Setters
void Character::setMaxHP(int maxhp) // Setter for fMaxHP
{
	fMaxHP = maxhp;
}


void Character::setCurrentHP(int currenthp) // Setter for fCurrentHP
{
	fCurrentHP = currenthp;
}

void Character::setCanDie(bool candie) // Setter for fCanDie
{
	fCanDie = candie;
}

void Character::setIsDead(bool isdead) // Setter for fIsDead
{
	if (fCanDie == false) // If fCanDie is false (meaning the character does not die)
	{
		fIsDead = false; // Then the character will never be dead
	}

	else
	{
		fIsDead = isdead;
	}
}


// Declare Getters
int Character::getMaxHP() // Getter for fMaxHP
{
	return fMaxHP;
}

int Character::getCurrentHP() // Getter for fCurrentHP
{
	return fCurrentHP;
}

bool Character::getCanDie() // Getter for fCanDie
{
```

```
	return fCanDie;
}

bool Character::getIsDead() // Getter for fIsDead
{
	return fIsDead;
}


//Declare functions
void Character::increaseHP(int increaseby) // Function to increase HP
{
	if (fCurrentHP < fMaxHP)
	{
		fCurrentHP = fCurrentHP + increaseby; // Increase character HP

		if (fCurrentHP > fMaxHP) // If current HP exceeds max HP
		{
			fCurrentHP = fMaxHP; // Character HP has been maxed out
		}
	}
}


void Character::decreaseHP(int decreaseby) // Function to decrease HP
{
	fCurrentHP = fCurrentHP - decreaseby; // Decrease character HP

	if (fCurrentHP <= 0) // If current HP decreases below zero or is zero
	{
		Die(); // Then character dies. Call die function
	}

}


void Character::Die()
{
	fCurrentHP = 0; // Character HP has been minimised
	fIsDead = true; // Character dies
}


void Character::moveLeft(int x) // Move left by x amount
{
	fPosition.X = fPosition.X - x;
}

void Character::moveRight(int x) // Move right by x amount
{
	fPosition.X = fPosition.X + x;
}

void Character::moveUp(int y) // Move up by y amount
{
	fPosition.Y = fPosition.Y + y;
}

void Character::moveDown(int y) // Move down by y amount
{
	fPosition.Y = fPosition.Y - y;

}

// Declare Destructor
```

```
Character::~Character()
{
}
```

// Playable.h

```
#pragma once

#include <iostream>
#include <string>
#include "Character.h"
#include "Iterator1D.h" //MUST INCLUDE TO USE ITERATOR OBJECT
#include "StepRecordList.h"
using namespace std;


class Playable : public Character
{
private:
protected:

	string fInventory[3];		// has an accessable inventory

	bool fHoldingItem; // is PC holding an item
	string fItemHeld;

	int fStepsTaken; // Number of moves the character walked
	int fStepCounter; // Counts a 'cycle' of steps; reset at a certain value to carry out something
	//last step taken and number of steps

	Iterator1D* fInventoryPtr;

	StepRecordList* fStepListPointer;

public:
	Playable();
	Playable(string name, string id, bool candie, int maxhp, int posx, int posy);

	// Declare Setters
	void setHoldingItem(bool choice); // Setter for fHoldingItem
	void setItemHeld(string item);

	// Declare Getters
	int getStepsTaken(); // Getter for fStepsTaken
	int getStepCounter(); // Getter for fStepCounter
	bool getHoldingItem();
	string getItemHeld(); // Getter for fHoldingItem

	// Polymorphism
	void moveLeft(int x); // Move left by x amount
	void moveRight(int x); // Move right by x amount
	void moveUp(int y); // Move up by y amount
	void moveDown(int y); // Move down by y amount



	// Declaring functions
	void resetStepCount(int stepnumber); // Reset number of steps

	void showInventory(); // show inventory contents
```

```
        void inventoryNextItem(); //Increment inventory item using iterator
        void inventoryPrevItem(); //Decrement inventory item using iterator

        void addItem(string itemname); // Add item into inventory

        Iterator1D* getInventoryItem(); // Get inventory item (pointer to element)

        void takeItem(); // Get inventory item void GetItem(Iteraor* iterator ptr, return fItemHeld);
        void dropItem(); // Drop inventory item

        void viewLastSteps(int limit); // Cycle back from doubly linked list and obtain last steps

        //Destructor
        virtual ~Playable();
};
```

// Playable.cpp

```
#include "Playable.h"
#include <iostream>
#include <string>

//Perhaps use stack to store record of position- allow a function to backtrack 5 steps
//use node to keep track of total number of steps taken

Playable::Playable()
{
        fInventory[0] = "";
        fInventory[1] = "";
        fInventory[2] = "";

        fHoldingItem = false;
        fItemHeld  = ""; // no value, no item currently held
        fInventoryPtr = new Iterator1D(fInventory, 3); // has an accessable inventory, size 3
        fStepsTaken = 0;
        fStepCounter = 0;

        fStepListPointer = new StepRecordList();
}


Playable::Playable(string name, string id, bool candie, int maxhp, int posx, int posy):Character(name, id, candie,
maxhp, posx, posy)
{
        fInventory[0] = "";
        fInventory[1] = "";
        fInventory[2] = "";

        fHoldingItem = false;
        fItemHeld = ""; // no value, no item currently held
        fInventoryPtr = new Iterator1D(fInventory, 3); // has an accessable inventory, size 3
        fStepsTaken = 0;
        fStepCounter = 0;

        fStepListPointer = new StepRecordList();
}

// Declare Setters
void Playable::setHoldingItem(bool choice) // Setter for fHoldingItem
{
        fHoldingItem = choice;
```

```
}

void Playable::setItemHeld(string item)
{
	fItemHeld = item;
}


// Declare Getters

int Playable::getStepsTaken()
{
	return fStepsTaken;
}

int Playable::getStepCounter()
{
	return fStepCounter;
}

bool Playable::getHoldingItem() // Getter for fHoldingItem
{
	return fHoldingItem;
}

string Playable::getItemHeld() //
{
	return fItemHeld;
}



// Declare functions
void Playable::moveLeft(int x) // Move left by x amount
{
	fPosition.X = fPosition.X - x;
	fStepsTaken++;
	fStepCounter++;

	fStepListPointer->addStep("Left", fStepsTaken, fPosition.X, fPosition.Y);
}

void Playable::moveRight(int x) // Move right by x amount
{
	fPosition.X = fPosition.X + x;
	fStepsTaken++;
	fStepCounter++;

	fStepListPointer->addStep("Right", fStepsTaken, fPosition.X, fPosition.Y);

}

void Playable::moveUp(int y) // Move up by y amount
{
	fPosition.Y = fPosition.Y + y;
	fStepsTaken++;
	fStepCounter++;

	fStepListPointer->addStep("Up", fStepsTaken, fPosition.X, fPosition.Y);

}

void Playable::moveDown(int y) // Move down by y amount
{
	fPosition.Y = fPosition.Y - y;
```

```
		fStepsTaken++;
		fStepCounter++;

		fStepListPointer->addStep("Down", fStepsTaken, fPosition.X, fPosition.Y);

}


void Playable::resetStepCount(int stepnumber) // Reset number of steps when it hits a certain number of steps
{
		if (fStepCounter == stepnumber)
		{
			fStepCounter = 0;
		}
}



void Playable::showInventory() // show inventory contents
{
		for (int i = 0; i < 3; i++)
		{
			cout << " " << fInventory[i] << " ";
		}
}

void Playable::inventoryNextItem() //Increment iterator to next item
{
		if ((*fInventoryPtr) == (fInventoryPtr->end()))
		{
			fInventoryPtr->end();

			//Don't do anything if at max end of inventory
		}

		else
		{
			++(*fInventoryPtr);
		}

		for (int i = 0; i < 3; i++)
		{
			if (fInventoryPtr->getIndex() == i)
			{
				cout << " [" << fInventory[i] << "] ";
			}

			else
			{
				cout << " " << fInventory[i] << " ";
			}
		}
}

void Playable::inventoryPrevItem() //Decrement iterator to previous item
{
		if ((*fInventoryPtr) == (fInventoryPtr->begin()))
		{
			fInventoryPtr->begin();
			//Don't do anything if at min beginning of inventory
		}

		else
		{
			--(*fInventoryPtr);
```

```
		}

		for (int i = 0; i < 3; i++)
		{
			if (fInventoryPtr->getIndex() == i)
			{
				cout << " [" << fInventory[i] << "] ";
			}

			else
			{
				cout << " " << fInventory[i] << " ";
			}
		}

}


void Playable::addItem(string itemname) // Add item into inventory
{
		bool itemadded = false;


		for (int i = 0; i < 3; i++)
		{
			if ((fInventory[i] == "") & (itemadded == false))
			{
				fInventory[i] = itemname;
				itemadded = true;
				cout << itemname << " added into inventory!" << endl;
			}
		}

		if (itemadded == false) // Means the inventory is full
		{
			cout << "Inventory is full! Unable to take." << endl;
		}

		if (fHoldingItem == true)
		{
			fHoldingItem = false;
			fItemHeld = "";
		}

}

Iterator1D* Playable::getInventoryItem() // Obtain inventory items
{
		return fInventoryPtr;
}

void Playable::takeItem() // Get inventory item void GetItem(Iterator* iterator ptr, return fItemHeld);
{
		fHoldingItem = true;
		fItemHeld = *(*fInventoryPtr);

		for (int i = 0; i < 3; i++)
		{
			if (fInventoryPtr->getIndex() == i)
			{
				fInventory[i] = "";
			}
		}

		cout << fItemHeld << " has been selected!" << endl;
```

```
}


void Playable::dropItem() // Drop inventory item
{
	fItemHeld = ""; // make value void- 'delete' item
	fHoldingItem = false;
}

void Playable::viewLastSteps(int limit) // Cycle back from doubly linked list and obtain last steps
{
	cout << fStepListPointer->showLastSteps(limit);
}

// Declare Destructor
Playable::~Playable()
{
	delete fInventoryPtr;
}
```

// NonPlayable.h

```
#pragma once

#include <iostream>
#include <string>
#include "Character.h"

using namespace std;

class NonPlayable : public Character
{
private:
protected:
	bool fWillDropItem;
	string fItemHeld;
	bool fIsFriendly; // is this NPC friendly? if false, will attack PC

public:
	NonPlayable();
	NonPlayable(string name, string id, bool candie, int maxhp, int posx, int posy, bool isfriendly);

	string DropItem(); // Return item dropped by the enemy

	// Declare Setters
	void setWillDropItem(bool choice); // Setter for fWillDropItem
	void setItemHeld(string itemname); // Setter for fItemHeld
	void setIsFriendly(bool choice); // Setter for fIsFriendly

	// Declare Getters
	bool getWillDropItem(); // Getter for fWillDropItem
	string getItemHeld(); // Getter for fItemHeld
	bool getIsFriendly(); // Getter for fIsFriendly

	// Declare functions
	void die();

	//Destructor
	virtual ~NonPlayable();
};
```

```
// NonPlayable.cpp
#include "NonPlayable.h"
#include <iostream>

NonPlayable::NonPlayable()
{
	fWillDropItem = false;
	fItemHeld = "";
	fIsFriendly = true;
}


NonPlayable::NonPlayable(string name, string id, bool candie, int maxhp, int posx, int posy, bool
isfriendly) :Character(name, id, candie, maxhp, posx, posy)
{
	fWillDropItem = false;
	fItemHeld = "";
	fIsFriendly = isfriendly;

}

string NonPlayable::DropItem()
{
	string drop = "";

	if ((fWillDropItem == true) & (fIsDead == true))
	{
		drop = fItemHeld;
	}

	return drop;
}


//Declare Setters

void NonPlayable::setWillDropItem(bool choice)
{
	fWillDropItem = choice;
}

void NonPlayable::setItemHeld(string itemname)
{
	fItemHeld = itemname;
}

void NonPlayable::setIsFriendly(bool choice)
{
	fIsFriendly = choice;
}


//Declare Getters

bool NonPlayable::getWillDropItem()
{
	return fWillDropItem;
}

string NonPlayable::getItemHeld()
{
	return fItemHeld;
```

```
}

bool NonPlayable::getIsFriendly()
{
	return fIsFriendly;
}


// Declare Functions
void NonPlayable::die()
{
	fCurrentHP = 0; // NPCharacter HP has been minimised
	fIsDead = true; // NPCharacter dies

	if (fWillDropItem == true)
	{
		DropItem();
	}

}


// Declare Destructor
NonPlayable::~NonPlayable()
{
}
```

// Iterator.h

```
//Base class
#pragma once

#include <iostream>
using namespace std;


class Iterator
{
protected:

	string* fArrayElements;
	int fIndex;

public:
	Iterator();

	Iterator(string aArray[], int aIndex = 0);

	string operator*();

	Iterator& operator++(); //prefix operator

	Iterator operator++(int); //postfix operator

	Iterator& operator--(); //prefix operator

	Iterator operator--(int); //postfix operator

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

	bool operator!=(const Iterator& aOther) const;
```

```
        virtual Iterator begin() const;

        virtual Iterator end() const;


        virtual int getIndex();

        virtual ~Iterator();
};
```

// Iterator.cpp

```
#include <iostream>
#include "Iterator.h"

using namespace std;

Iterator::Iterator()
{
        fArrayElements = NULL;
        fIndex = 0;
}

Iterator::Iterator(string aArray[], int aIndex)
{
        fArrayElements = aArray;
        fIndex = aIndex;
}

string Iterator::operator*()
{
        return fArrayElements[fIndex];
}

Iterator& Iterator::operator++()
{
        //prefix operator
        fIndex++;
        return *this;
}

Iterator Iterator::operator++(int)
{
        //postfix operator
        Iterator temp = *this;
        fIndex++;
        return temp;
}

Iterator& Iterator::operator--()
{
        //prefix operator
        fIndex--;
        return *this;
}

Iterator Iterator::operator--(int)
{
        //postfix operator
        Iterator temp = *this;
        fIndex--;
        return temp;
}
```

```
bool Iterator::operator==(const Iterator& aOther) const
{
	return (fIndex == aOther.fIndex); //Return true or false
}

bool Iterator::operator!=(const Iterator& aOther) const
{
	return (fIndex != aOther.fIndex); //Return true or false
}

Iterator Iterator::begin() const
{
	return Iterator();
}

Iterator Iterator::end() const
{
	return Iterator();
}


int Iterator::getIndex()
{
	return fIndex;
}


//Destructor
Iterator::~Iterator()
{
}
```

// Iterator1D.h

```
#pragma once

#include <iostream>
#include "Iterator.h"

using namespace std;

class Iterator1D :
	public Iterator
{

private:
	int fEndIndex;

public:
	Iterator1D();

	Iterator1D(string aArray[], int aArrSize, int aIndex = 0); //apparently default parameter should be at end of parameter list

	Iterator begin() const;

	Iterator end() const;

	~Iterator1D();
};
```

// Iterator1D.cpp

```
#include <iostream>
#include "Iterator1D.h"

using namespace std;

Iterator1D::Iterator1D()
{
	fArrayElements = NULL;
	fIndex = 0;
	fEndIndex = 0;
}


Iterator1D::Iterator1D(string aArray[], int aArrSize, int aIndex) : Iterator(aArray, aIndex)//, fArrayElements(aArray)
{
	fArrayElements = aArray;
	fEndIndex = aArrSize - 1; //Show end of Array
}

Iterator Iterator1D::begin() const
{
	int lArrSize = fEndIndex + 1;
	return Iterator1D(fArrayElements, lArrSize);

}

Iterator Iterator1D::end() const
{
	int lArrSize = fEndIndex + 1;
	return Iterator1D(fArrayElements, lArrSize, fEndIndex);

}

//Destructor
Iterator1D::~Iterator1D()
{
}
```

// RiddleNode.h

```
#pragma once // Guards against repeated inclusion
#include <iostream>
#include <string>

using namespace std;

class RiddleNode // Doubly-Linked Node to represent Steps taken
{

private:
	string fQuestion; // Records the direction taken
	string fAnsChoices[3];

	int fCorrectAnsIndex;

	RiddleNode* fNext; // Pointer to the next node

public:
```

```
        RiddleNode(); // Default constructor
        RiddleNode(string question, string choice1, string choice2, string choice3, int correctindex); // Constructor,
where direction can be: LEFT, RIGHT, UP, DOWN

        // Declaring Getters
        RiddleNode* getNextNode(); // Get pointer to the next node

        string getQuestion();  // Getter for fQuestion
        string getAnsChoices(); // Getter for fAnsChoices
        int getCorrectAnsIndex();

        // Declaring Setters
        void setQuestion(string ques);  // Setter for fQuestion
        void setAnsChoices(int ansindex, string ans); // Getter for fAnsChoices

        // Declaring functions
        void prependRiddle(RiddleNode& riddle);
        void removeRiddle(); // Remove current node from links



        //Destructor
        virtual ~RiddleNode();
};
```

// RiddleNode.cpp

```
#pragma once // Guards against repeated inclusion
#include "RiddleNode.h"
#include <iostream>
#include <string>

RiddleNode::RiddleNode() // Default constructor. Create a new skill node
{
        fQuestion = "";
        fAnsChoices[0] = "";
        fAnsChoices[1] = "";
        fAnsChoices[2] = "";

        fCorrectAnsIndex = 0;

        // Current skill node is not pointing to other nodes
        fNext = NULL;
}

RiddleNode::RiddleNode(string question, string choice1, string choice2, string choice3, int correctindex) //
Constructor. Create a new skill node
{
        fQuestion = question;
        fAnsChoices[0] = choice1;
        fAnsChoices[1] = choice2;
        fAnsChoices[2] = choice3;

        // Keyed in integer -1 for index
        fCorrectAnsIndex = correctindex;

        // Current skill node is not pointing to other nodes
        fNext = NULL;
}
```

```
// Declaring Getters

RiddleNode* RiddleNode::getNextNode()
{
	return fNext;
}


string RiddleNode::getQuestion() // Getter for fQuestion
{
	return fQuestion;
}

string RiddleNode::getAnsChoices() // Getter for fAnsChoices
{
	string choices = "";

	for (int i = 0; i < 3; i++)
	{
		choices = choices + to_string(i + 1) + ") " + fAnsChoices[i] + '\n';
	}
	return choices;
}


int RiddleNode::getCorrectAnsIndex()
{
	return fCorrectAnsIndex;
}




// Declaring Setters
void RiddleNode::setQuestion(string ques) // Setter for fQuestion
{
	fQuestion = ques;
}

void RiddleNode::setAnsChoices(int ansindex, string ans) // Getter for fAnsChoices
{
	fAnsChoices[ansindex] = ans;
}


// Declaring functions
void RiddleNode::prependRiddle(RiddleNode& riddle) // Add new node at the beginning of list
{
	this->fNext = &riddle; // Prepend node before riddle
}


void RiddleNode::removeRiddle() // Remove current node from links, only works at the top
{
	this->fNext = NULL;
}



// Declare Destructor
RiddleNode::~RiddleNode()
{
}
```

// RiddleNodeList.h

```
#pragma once
#include "RiddleNode.h"
#include <string>
using namespace std;


class RiddleRecordList
{

private:
	// Linked List of skill nodes
	RiddleNode* headNode; // Pointer to head of list of skills
	RiddleNode* tailNode; // Pointer to tail of list of skills
	RiddleNode* riddlePtr; // Pointer to any skill in the list

public:

	RiddleRecordList();
	void addRiddle(string question, string choice1, string choice2, string choice3, int correctchoice);

	// Declare functions to manage skill list
	void removeTop();  // Remove Riddle at the front of list

	string showRiddleAndChoices(); // Output Riddle and its choices

	int correctAns(); // Obtain Integer for correct answer

	void NextRiddle(); // Point at next riddle

	// Declare Getters
	RiddleNode* getRiddlePtr();

	~RiddleRecordList();
};
```

// RiddleRecordList.cpp

```
#pragma once // Guards against repeated inclusion
#include "RiddleRecordList.h"
#include "RiddleNode.h"
#include <iostream>
#include <string>

RiddleRecordList::RiddleRecordList()
{
	tailNode = new RiddleNode("", "", "", "", 0); // Create sentinel tail node on heap
	headNode = tailNode; // also poiint at end

	riddlePtr = NULL; // Don't point at anything yet
}

void RiddleRecordList::addRiddle(string question, string choice1, string choice2, string choice3, int correctchoice) //
Add skill (learn new skill)
```

```
{
	riddlePtr = new RiddleNode(question, choice1, choice2, choice3, correctchoice);

	riddlePtr->prependRiddle(*headNode);
	headNode = riddlePtr; // point at beginning of list
}

// Declare functions to manage skill list
void RiddleRecordList::removeTop() // Remove Riddle at the front of list
{
	riddlePtr = headNode->getNextNode(); // Point at next node
	headNode->removeRiddle(); // Remove riddle
	headNode = riddlePtr; // Point at new head of list
}

string RiddleRecordList::showRiddleAndChoices() // Output Riddle and its choices
{
	//and delete riddles already asked
	string qna = "";

	qna = riddlePtr->getQuestion() + '\n' + riddlePtr->getAnsChoices();

	return qna;
}

void RiddleRecordList::NextRiddle() // Obtain Integer for correct answer
{
	//After adding riddle riddlePtr will definitely be at the head of list

	riddlePtr = riddlePtr->getNextNode(); // Point at head of list
	// Note that skillPtr is now pointing at our current node that we may want to delete

	cout << "next riddle!" << endl;
	if (riddlePtr == tailNode)
	{
		riddlePtr = headNode; // Go back to the head node
	}
}

// Declare getters
RiddleNode* RiddleRecordList::getRiddlePtr() // Getter for riddlePtr
{
	return riddlePtr;
}


int RiddleRecordList::correctAns()
{
	int ansindex = riddlePtr->getCorrectAnsIndex(); // Point at next riddle

	return ansindex;
}



RiddleRecordList::~RiddleRecordList()
{
	riddlePtr = headNode; // Point at head of list
	// Note that skillPtr is now pointing at our current node that we may want to delete

	while (riddlePtr != tailNode) // While not at end of the skills list
	{
		headNode = headNode->getNextNode();

		riddlePtr->removeRiddle(); //Start deleting from beginning of list
```

```
                delete riddlePtr;

                riddlePtr = headNode;
        } // Iterate process when end of list is not reached

        delete tailNode;
}
```

// StepNode.h

```
#pragma once // Guards against repeated inclusion
#include <iostream>
#include <string>

using namespace std;

class StepNode // Doubly-Linked Node to represent Steps taken
{

private:
        string fDirection; // Records the direction taken
        int fPosX; // Position x
        int fPosY; // Position y
        int fStepNumber; // nth step

        StepNode* fNext; // Pointer to the next node
        StepNode* fPrevious; // Pointer to a previous node

public:

        StepNode(); // Default constructor
        StepNode(string direction, int stepnumber, int posx, int posy); // Constructor, where direction can be: LEFT,
RIGHT, UP, DOWN

        // Declaring Getters

        StepNode* getPrevNode(); // Get pointer to the previous node
        StepNode* getNextNode(); // Get pointer to the next node

        string getDirection();  // Getter for fSkillName
        int getPosX(); // Getter for fPosX
        int getPosY(); // Getter for fPosY
        int getStepNumber(); // Getter for fStepNumber

        // Declaring Setters
        void setDirection(string direction); // Setter for fSkillName


        // Declaring functions
        void prependStep(StepNode& skill); // Add in a skill before current skill node
        void appendStep(StepNode& skill); // Add in a skill after current skill node
        void removeStep(); // Remove current node from links



        //Destructor
        virtual ~StepNode();
};
```

```cpp
// StepNode.cpp
#pragma once // Guards against repeated inclusion
#include "StepNode.h"
#include <iostream>
#include <string>

StepNode::StepNode() // Default constructor. Create a new skill node
{
	fDirection = " ";
	fPosX = 0;
	fPosY = 0;

	// Current skill node is not pointing to other nodes
	fNext = NULL;
	fPrevious = NULL;
}

StepNode::StepNode(string direction, int stepnumber, int posx, int posy) // Constructor. Create a new skill node
{
	fDirection = direction;
	fPosX = posx;
	fPosY = posy;

	// Current skill node is not pointing to other nodes
	fNext = NULL;
	fPrevious = NULL;
}



// Declaring Getters

StepNode* StepNode::getPrevNode()
{
	return fPrevious;
}

StepNode* StepNode::getNextNode()
{
	return fNext;
}

string StepNode::getDirection()
{
	return fDirection; // Getter for fSkillName
}

int StepNode::getPosX() // Getter for fPosX
{
	return fPosX;
}

int StepNode::getPosY() // Getter for fPosY
{
	return fPosY;
}

int StepNode::getStepNumber() // Getter for fStepNumber
{
	return fStepNumber;
}
```

```
// Declaring Setters

void StepNode::setDirection(string direction) // Setter for fSkillName
{
	fDirection = direction;
}


// Declaring Functions

void StepNode::prependStep(StepNode& skill) // Add in a skill before current skill node
{
	skill.fNext = this; // Added skill is placed left of (before) current node
	// The right/forward pointer of new node is pointing towards current node

	if (fPrevious != NULL) // If current skill is not the first node in list
	{
		skill.fPrevious = fPrevious; // The left/backward pointer of new node is pointing to the previous node of the current node
		fPrevious->fNext = &skill; // The left/backward pointer of previous node is then made to point to new node
		// ^ Read as: previousnode's fNext made to point to new node
	}

	fPrevious = &skill; // The left/backward pointer of current node is made to point at new node
}


void StepNode::appendStep(StepNode& skill) // Add in a skill after current skill node
{
	skill.fPrevious = this; // Added skill is placed right of (after) current node
	// The left/backward pointer of new node is pointing towards current node

	if (fNext != NULL) // If current skill is not the last node in list
	{
		skill.fNext = fNext; // The right/forward pointer of new node is pointing to the next node of the current node
		fNext->fPrevious = &skill; // The  right/forward pointer of next node is then made to point to new node
		// ^ ` as: nextnode's fPrevious made to point to new node
	}

	fNext = &skill; // The right/forward pointer of current node is made to point at new node
}

void StepNode::removeStep() //Remove current skill from list of nodes
{
	if (fPrevious != NULL)
	{
		fPrevious->fNext = fNext;
		// ^ Read as: previousnode's fNext made to point to nextnode (next node of current node)
	}

	if (fNext != NULL)
	{
		fNext->fPrevious = fPrevious;
		// ^ Read as: nextnode's fPrevious made to point to previousnode (previous node of current node)
	}

	// Isolate current node
	fPrevious = NULL;
	fNext = NULL;

	delete this; //Destruct after isolating
```

```
}

// Declare Destructor
StepNode::~StepNode()
{
}
```

// StepRecordList.h

```
#pragma once
#include "StepNode.h"
#include <string>
using namespace std;


class StepRecordList
{

private:
	// Linked List of skill nodes
	StepNode* headNode; // Pointer to head of list of skills
	StepNode* tailNode; // Pointer to tail of list of skills
	StepNode* stepPtr; // Pointer to any skill in the list

public:

	StepRecordList();

	// Declare functions to manage skill list
	StepNode* findStep(int stepnumber); // Find step based on step number, return pointer


	void addStep(string direction, int stepnumber, int x, int y); // Add skill (learn new skill)
	void removeStep(int stepnumber); // Remove step (unlearn) by name and level

	string showLastSteps(int limit); // View character steps-> print last limit steps?


	~StepRecordList();
};
```

// StepRecordList.cpp

```
#pragma once // Guards against repeated inclusion
#include "StepRecordList.h"
#include "StepNode.h"
#include <iostream>
using namespace std;


StepRecordList::StepRecordList()
{

	headNode = new StepNode("SentinelStart", NULL, NULL, NULL); // Create sentinel head node on heap
	tailNode = new StepNode("SentinelEnd", NULL, NULL, NULL); // Create sentinel tail node on heap

	headNode->appendStep(*tailNode); // Link the head and tail sentinel nodes together

	stepPtr = headNode; // Point at head of list
```

```
}


// Declare functions to manage skill list

StepNode* StepRecordList::findStep(int stepnumber) // Find if a step at a certain number exists
{
	int number = 0;
	bool stepfound = false;
	StepNode* temp = nullptr;

	stepPtr = headNode; // Point to head of list
	stepPtr->getNextNode(); // Point to next step. Since we start at headNode, we want to point at the next one

	while (stepPtr != tailNode) // While not at end of the step list
	{
		if ((stepPtr->getStepNumber() == stepnumber))
		{
			stepfound = true;
			temp = stepPtr;
		}
		stepPtr = stepPtr->getNextNode(); // Point to next step and continue through the list
	}

	if (temp != nullptr)
	{
		stepPtr = temp;
	}

	else
	{
		stepPtr = NULL;
	}

	return stepPtr;
}



void StepRecordList::addStep(string direction, int stepnumber, int posx, int posy) // Add step
{
	stepPtr = new StepNode(direction, stepnumber, posx, posy); // Create new step on the heap

	tailNode->prependStep(*stepPtr); // Prepend the newly added step to the list (add in before sentinel tail
node)
	stepPtr = tailNode->getPrevNode(); // Point at current (newly added) step
}

void StepRecordList::removeStep(int stepnumber) // Remove step
{
	stepPtr = headNode; // Point to head of list
	stepPtr->getNextNode(); // Point to next skill. Since we start at headNode, we want to point at the next one

	while (stepPtr != tailNode) // While not at end of the step list
	{
		if (stepPtr->getStepNumber() == stepnumber)
		{
			stepPtr->removeStep(); // Remove the step
			stepPtr = headNode; // Point back to headNode again
		}
		stepPtr = stepPtr->getNextNode(); // Point to next step. Since we start at headNode, we want to
point at the next one
	}
}
```

```
string StepRecordList::showLastSteps(int limit) // View character steps
{
	string outputlist = "Showing the last " + to_string(limit) + " steps:" + '\n';
	int n = 0;

	stepPtr = tailNode; // Point to beginning of step list
	 // Point to next step. Since we start at headNode, we want to point at the next one

	stepPtr = stepPtr->getPrevNode();

	for (int i = 0; i < limit; i++)
	{
		if (stepPtr == headNode)
		{
			break;
		}

		outputlist = outputlist + to_string(i+1) + ") " + stepPtr->getDirection() + '\n';
		stepPtr = stepPtr->getPrevNode();
	}

	return outputlist;
}


StepRecordList::~StepRecordList()
{
	StepNode* temp; // Create a temporary pointer
	stepPtr = headNode; // Point at head of step list

	stepPtr = headNode; // Point to beginning of step list
	stepPtr = stepPtr->getNextNode(); // Point to next step. Since we start at headNode, we want to point at the
next one

	// Note that stepPtr is now pointing at our current node that we may want to delete

	while (stepPtr != tailNode) // While not at end of the steps list
	{
		stepPtr = stepPtr->getNextNode(); // Point stepPtr at the next node
		temp = stepPtr->getPrevNode();  // Point temp at the 'current' node

		delete temp; // Destruct the step node
	} // Iterate process when end of step list is not reached

	delete headNode;
	delete tailNode;
}
```

// MapPiece.h

```
#pragma once // Guards against repeated inclusion
#include <iostream>
#include <string>
#include "Entity.h"
#include "Object.h"
#include "Playable.h"
#include "NonPlayable.h"
using namespace std;

// Declare struct Walls
struct Walls
{
```

```
        bool LeftW, RightW, FrontW, BackW;
};

class MapPiece
{
private:

protected:
        Walls fWalls; // Walls struct
        Object* ObjectPtr; // Pointer to Object object
        NonPlayable* NPCPointer; // Pointer to NonPlayable object

public:

        // Constructors
        MapPiece();

        MapPiece(Walls walls);
        MapPiece(Walls walls, Object& object);
        MapPiece(Walls walls, NonPlayable& npc);


        // Declare Getters
        Walls getWalls(); // Getter for fWalls
        Object* getObject(); // Getter for ObjectPtr
        NonPlayable* getNPC(); // Getter for NPCPointer

        // Destructor
        ~MapPiece();

};
```

// MapPiece.cpp

```
#pragma once // Guards against repeated inclusion
#include "MapPiece.h"
#include <iostream>
#include <string>

MapPiece::MapPiece() // Default constructor
{
        fWalls.LeftW = false;
        fWalls.RightW = false;
        fWalls.FrontW = false;
        fWalls.BackW = false;

        ObjectPtr = nullptr;
        NPCPointer = nullptr;
}

MapPiece::MapPiece(Walls walls)
{
        fWalls = walls;

        ObjectPtr = nullptr;
        NPCPointer = nullptr;
}

MapPiece::MapPiece(Walls walls, Object& object)
{
        fWalls = walls;

        ObjectPtr = &object;
```

```
	NPCPointer = nullptr;
}

MapPiece::MapPiece(Walls  walls, NonPlayable& npc)
{
	fWalls = walls;

	ObjectPtr = nullptr;
	NPCPointer = &npc;
}

Walls MapPiece::getWalls()  // Getter for fWalls
{
	return fWalls;
}

Object* MapPiece::getObject()
{
	return ObjectPtr;
}

NonPlayable* MapPiece::getNPC()
{
	return NPCPointer;
}

//Destructor
MapPiece::~MapPiece()
{
	if (NPCPointer != nullptr)
	{
		delete NPCPointer;
	}

	if (ObjectPtr != nullptr)
	{
		delete ObjectPtr;
	}
}
```