Nonsuch High School for Girls

Candidate number: 9197  
Centre number: 14723  
Qualification Code:  
Date: 2022

REMEMBRIT: Text based adventure game

Alyssa Venter

Contents

[Problem – 3](#_Toc101343158)

[Identifying the End-User – 3](#_Toc101343159)

[Research – 3](#_Toc101343160)

[Existing Resources: 3](#_Toc101343161)

[Articles: 3](#_Toc101343162)

[Survey: 4](#_Toc101343163)

[Conclusion: 5](#_Toc101343164)

[Objectives – 5](#_Toc101343165)

[Documented Design: 7](#_Toc101343166)

[Structure Diagram – 7](#_Toc101343167)

[Creating the map – 7](#_Toc101343168)

[First design: 7](#_Toc101343169)

[New map design: 8](#_Toc101343170)

[Class Diagram – 9](#_Toc101343171)

[Flowchart – 10](#_Toc101343172)

[Pseudocode – 11](#_Toc101343173)

[Outputting the map: (\* means a room is locked) 11](#_Toc101343174)

[Changing the player’s position with the graph: 11](#_Toc101343175)

[File Design – 12](#_Toc101343176)

[Dialogue file design: 12](#_Toc101343177)

[User Interface Design: 13](#_Toc101343178)

[Technical Solution: 14](#_Toc101343179)

[Log: 14](#_Toc101343180)

[10/01/2022: 14](#_Toc101343181)

[12/01/2022 15](#_Toc101343182)

[21/02/2022 16](#_Toc101343183)

[22/2/2022 16](#_Toc101343184)

[23/02/2022 17](#_Toc101343185)

[24/02/2022 17](#_Toc101343186)

[27/02/2022 21](#_Toc101343187)

[8/03/2022 23](#_Toc101343188)

[09/03/2022 24](#_Toc101343189)

[10/03/2022 25](#_Toc101343190)

[11/03/2022 27](#_Toc101343191)

[16/03/2022 27](#_Toc101343192)

[17/03/2022 28](#_Toc101343193)

[23/03/2022 29](#_Toc101343194)

[25/03/2022 30](#_Toc101343195)

[29/03/2022 30](#_Toc101343196)

[4/4/2022 31](#_Toc101343197)

[06/04/2022 31](#_Toc101343198)

[Annotated code: 32](#_Toc101343199)

[Testing: 55](#_Toc101343200)

[Testing criteria and results: 55](#_Toc101343201)

[Screenshots: 59](#_Toc101343202)

[File contents: 69](#_Toc101343203)

[Evaluation: 71](#_Toc101343204)

[Meeting user objectives: 71](#_Toc101343205)

[Independent feedback: 73](#_Toc101343206)

[Evaluation of independent feedback: 74](#_Toc101343207)

[References 75](#_Toc101343208)

[Credit: 75](#_Toc101343209)

Analysis:

## Problem –

As a tutor, my students have expressed to me that they find written tests to typically be ‘boring’ and I find they do not encourage active participation. Moreover, video games are popular with my students, so I had the idea to combine the two. Merging computer games and testing/memorising has already been successful (for example Kahoot or Quizlet), however, the ‘games’ featured are often basic, like a simple matching game. Longer-style adventure games are also popular with my students, which is useful for learning as they feature storylines that draw the user in and allow them to feel a sense of accomplishment when the game is finished. This will mean I need to deal with large amounts of data in the form of dialogue and ASCII art, which will be stored in text files.

## Identifying the End-User –

The users of the program will be younger students trying to memorise or test something in an entertaining way. This would also be beneficial to older students who are trying to memorise keywords or dates.

## Research –

### Existing Resources:

#### Kahoot:

Kahoot is an online multiplayer game that allows users to play trivia games. Whilst it does test knowledge and help players retain information, it does not make it easy for educators to track a student’s progress as they are constantly in competition with other players in the quiz. The point system is also based on speed, which makes it a good game for a final test, but not for learning content.

#### Quizlet:

Quizlet is a revision website that allows users to create flashcards. This is an effective revision tool, however the games on the website like ‘Match’ (a classic pairing game) or ‘Gravity’ (which is a game of speed to destroy asteroids) do not encourage the same sense of accomplishment and repetition of questions that an adventure game would. Therefore, the game must be able to import ‘cards’ already created on Quizlet to the video game as a solution.

### Articles:

An article by Kurt Squire called ‘Video Games in Education’ (Squire, 2011) is particularly useful in justifying the way video games can help in education. A main view in his book is that ‘educational games provide an intrinsic desire for learning, enhanced citizenship and can develop strong partnerships within the educational systems and community groups’ (Pleasant, 2013).

He also discusses the ideas of Malone, who produced a set of criteria to make educational games ‘fun’ (Malone, 1980):

* clear goals that students find meaningful
* multiple goal structures and scoring to give students feedback on their progress
* multiple difficulty levels to adjust the game difficulty to learner skill
* random elements of surprise

### Survey:

However, most of this research is over 30 years old and may not be as relevant today, in a society where video games are extremely prevalent. Therefore, I did some research with some my students on Google Forms:

Chart, pie chart

Description automatically generated

Chart, pie chart

Description automatically generatedChart, pie chart

Description automatically generated

### Conclusion:

In an age range of 10 to 18, most students found that tests at school were boring and difficult to revise for, however most enjoy video games. A strong percentage found that websites like ‘kahoot’ were useful to their revision, meaning they find interactive learning – and will consequently find Remembrit - useful.

#### Similar Games:

* Minecraft in education
  + As an open world sandbox-style game, it allows for users to express their creativity.
  + Whilst this is useful for some forms of education, this is not good for revision materials or testing.
* ZORK: text-based adventure game
  + This game is difficult to input answers, especially for children.
  + It is easy to include revision materials

Therefore, a text-based game is the most appropriate, as it is easy to ask questions to users without distracting them from the questions.

## Objectives –

1. The user must be able to enter their own questions
   1. Users can enter either by loading a text file
   2. Or by writing their own questions using the console
      1. Questions should be able to be saved in a text file and then reused again
2. There must be enemies to fight against
   1. Diverse types of enemies should be in different areas
   2. They should be randomised to avoid repetition
   3. There should be different ‘levels’ of enemies, like ‘normal’ or ‘boss’ enemies
3. There must be a set map that the user can move around
   1. The map must have a certain number of rooms to allow for random generation of enemies
4. The game should have levels of difficulty in accordance with the ideas of Malone
   1. This should be in the form of enemies, where each ‘section’ of the map will have a different level of difficulty
5. The game must comply with the rules
   1. Boss characters will have 5 health points – answering each question will get rid of a random amount of health points between 1 and 3.
   2. Normal enemies will have 3 health points – each question will get rid of 1 point. It will deal 1 attack damage if you get it wrong.
   3. You must defeat the bosses and get ‘items’ they drop. If you have all three items this will unlock room 10, which will let you begin the ultimate boss battle.
   4. The Ultimate boss will have as many health points as the number of questions the user enters, therefore asking all the questions. It can deal 3 attack damage if you get a question wrong.
   5. The player will have 30 health points.
6. There must be feedback for students after they have finished the game
   1. This will be represented as points and the score will be shown throughout the game
7. The game should be easy to use as the end users are younger children who may not understand complicated inputs
   1. There should be predefined letters to input (with validation) so that users can make choices of where to go easily
   2. There should be a help option if the user is stuck

# Documented Design:

## Structure Diagram –

Timeline

Description automatically generated

## A picture containing text, whiteboard Description automatically generatedCreating the map –

### First design:

However, this design had too many rooms – it would take more than an hour to play which would not fit in lesson times.

### New map design:

Diagram

Description automatically generated with medium confidence

The new design for the map ensures that there are enough rooms to allow for randomisation (in accordance with objective 3i). The new design also has updates from the 70’s theme to an arcade theme to be more user friendly and less specific, especially for younger users.

This graph should be implemented using a **dictionary of dictionaries**:

*(pseudocode)*

1: { {2, 0}, {5, 1}, {7, 1}, {10, 1} }

2: { {1, 0}, {3, 0}, {4, 1}}

3: { {2, 0}, {4, 1} }

4: { {2, 0}, {3, 0} }

5: { {1, 0}, {6, 0} }

6: { {5, 0} }

7: { {1, 0}, {8, 0}, {9, 1} }

8: { {7, 0}, {9, 1} }

9: { {8, 0}, {7, 0} }

10:{ {1, 0} }

The use of a dictionary of dictionaries allows for the rooms that can be travelled to from a particular room to be stored, as well as whether they are locked or unlocked (represented using a ‘0’ or ‘1’). However, if the dictionary is being changed, you cannot use a dictionary in the same place it has been changed. I found an article (Shri, n.d.) that proposes the solution of placing the second dictionary in lists instead, so it is not iterating through something being changed.

## Class Diagram –

Diagram

Description automatically generatedText, letter

Description automatically generated

1 Initial design for the enemy with attributes and methods

This demonstrates the object-oriented programming that will be implemented in the program, with ‘Bosses’ and ‘FinalBoss’ being inherited from the Enemies class. This allows the program to be organised more effectively and reduces redundancy as it means that code does not have to be repeated.

## Flowchart –

Diagram

Description automatically generated

2 An overview of how the system will work.

## Pseudocode –

### Outputting the map: (\* means a room is locked)

List<> unlockedRoomList

String map ← mapfile

string newmap ← “”

IF unlockedRooms.Count > 0

FOR i ← 1 to map.Length()

string nextChar ← map[i + 1]

IF (nextChar = integer)

IF (map[i] = '\*' && unlockedRooms.Contains(nextCharInt))

newmap ← newmap + " "

ELSE

newmap ← newmap + map[i]

ENDIF

ELSE

newmap ← newmap + map[i]

ENDIF

ENDFOR

newmap += map[map.Length - 1];

Console.WriteLine(newmap)

ELSE

Console.WriteLine(map)

END IF

### Changing the player’s position with the graph:

bool found ← false

bool unlocked ← true

int counter ← 0

int PlayerPosition ← (PASSED IN)

int newPos ← (PASSED IN)

Dictionary<int, Dictionary<int, int>> map ← (PASSED IN)

Dictionary<int, int> valuePairs ← map[playerPosition]

FOREACH (int key IN valuePairs)

List<int> ← keys.Add(key.Key)

List<int> ← values.Add(key.Value)

END FOREACH

IF (newPos = 10 && inventory.Count >= 3)

PlayUltimateBoss()

unlocked ← false

ELSE

FOREACH (int locked IN keys)

IF (values[counter] = 0 && locked = newPos)

playerPosition ← newPos

found ← true

unlocked ← false

ELSEIF (values[counter] = 1 && locked = newPos)

unlocked ← CheckScore(newPos)

IF (unlocked = false)

OUTPUT ← “THIS DOOR IS LOCKED.”

found ← true

ELSE

found ← UserUnlockRoom(newPos)

ENDIF

counter ← counter + 1

ENDIF

END FOREACH

IF (!found && playerPosition != newPos)

OUTPUT ← "You cannot enter this area from here."

ENDIF

IF ((playerPosition = 4 || playerPosition = 6 || playerPosition = 9))

PlayBoss()

ENDIF

ENDIF

RETURN unlocked;

## File Design –

Diagram

Description automatically generated

As there are many files that will be loaded into the program (for less repetitive parts of dialogue, music files and art that will be displayed), the system needs to be organised. This will allow for an easy installation of the game if it is being used on multiple devices apart from the one I created it on.

### Dialogue file design:

Table

Description automatically generated

The way the dialogue can organised to be read from the files. This will allow it to be split into a dictionary, and a method can be created to go through this and output when needed.

## User Interface Design:

Easy to enter letters with clear explanation

A screenshot of a computer

Description automatically generated with medium confidence

Each menu option leads to its own sub-option, which allows the user to enter a file to load in or asks the user to input the questions they would like to be tested on one by one.

A screenshot of a computer

Description automatically generated

Map is shown with ASCII art which allows the user to visualise it. This will be helpful especially for engaging younger users. It will also remove the asterisks when a room becomes unlocked.

Clearly shows the number of points needed to get into another room

All sub-menus will also have the same system of letters to input for a particular option.

A computer screen capture

Description automatically generated with medium confidence

Different colours used for the dialogue and question which helps break up the text.

ASCII art to ensure the game is entertaining for younger players, which therefore means their learning will be more entertaining.

# Technical Solution:

## Log:

### 10/01/2022:

#### Creating the methods for reading and writing questions to the text file:

//Importing the questions with a file either from kahoot or pre-written using the function in the program

static void GetQuestionsFile(ref bool questionsLoaded)

{

string path;

if (!fileWritten)//if the user is entering their own

{

Console.WriteLine("Please enter your file name. It must be in the 'bin, debug' folder this program.");

path = Console.ReadLine();

}

else //when called using the WriteQuestions() method

{

path = "WriteLines.txt";

}

try

{

using (StreamReader sr = new StreamReader(path))

{

int lineNumber = 0;

int numberOfLines = File.ReadAllLines(path).Length;//number of lines using system.io

string[,] Questions = new string[2, numberOfLines];

// Console.WriteLine(numberOfLines);

while (sr.Peek() >= 0) //while not at end of file

{

string line = sr.ReadLine();

string question = "";

string answer = "";

int commaPosition = 0;

bool commaFound = false;

while (!commaFound)

{

if (line[commaPosition] == ',') //finds where the comma is in the line

{

commaFound = true;

}

else

{

commaPosition++;

}

}

for (int i = 0; i < line.Length; i++) //assigning the string before the comma to 'question' and after to 'answer'

{

if (i < commaPosition && i != commaPosition)

{

question += line[i];

}

else if (i > commaPosition && i != commaPosition)

{

answer += line[i];

}

}

//adding into 2d array

Questions[0, lineNumber] = question;

Questions[1, lineNumber] = answer;

lineNumber++;

}

Console.WriteLine("Questions added!");

questionsLoaded = true;

}

}

catch (Exception e) //validation

{

Console.WriteLine("The file could not be read:");

Console.WriteLine(e.Message);

questionsLoaded = false; //cannot play game if this is false

}

}

### 12/01/2022

#### Method created for the menu and writing the text from file:

static void WriteQuestions(ref bool questionsLoaded)

{

Console.WriteLine("How many questions would you like to enter?");

int questionNumbers = Convert.ToInt32(Console.ReadLine());

string text = "";

for (int i = 0; i < questionNumbers; i++)

{

Console.WriteLine("Question " + (i + 1) + ": ");//asks for the questions

text += Console.ReadLine() + ",";

if (i != questionNumbers - 1)

{

Console.WriteLine("Answer " + (i + 1) + ": ");

text += Console.ReadLine() + Environment.NewLine;//asks for answers

}

else

{

Console.WriteLine("Answer " + (i + 1) + ": "); //last line without an empty line in the array

text += Console.ReadLine();

}

}

using (StreamWriter outputFile = new StreamWriter("WriteLines.txt"))//writes to a file to be loaded in LoadFile()

{

outputFile.WriteLine(text);

}

fileWritten = true;

GetQuestionsFile(ref questionsLoaded);

}

static void Menu()

{

bool endGame = false;

bool questionsLoaded = false;

while (!endGame)//continues until close game is selected

{

Console.WriteLine("---------------------Welcome to the game!---------------------");

Console.WriteLine("Play game (P), Load file (L), Write questions (W), Close game (C)");

string input = Console.ReadLine();

if (input == "L" || input == "C" || input == "W" || input == "P")//checks if valid input

{

char answer;

answer = char.ToUpper(Convert.ToChar(input));

if (answer == 'L')

{

GetQuestionsFile(ref questionsLoaded);

}

if (answer == 'C')

{

Console.WriteLine("Thank you for playing the game!");

endGame = true;

}

if (answer == 'W')

{

WriteQuestions(ref questionsLoaded);

}

if (answer == 'P')

{

if (questionsLoaded == true)

{

PlayGame();

}

else

{

Console.WriteLine("Please enter your questions before beginning the game!");

}

}

}

else

{

Console.WriteLine("Not a valid input. Please try again.");

}

}

}

### 21/02/2022

#### Creating the map and beginning of ‘PlayGame’, as well as some class diagrams for the implementation of the enemies.

### 22/02/2022

#### Encountered a problem with the implementation of the map – how does it recognise which direction it is? Cannot use a dictionary with 3 values. Program individually?

static void SetUpMap()

{

map = new Dictionary<int, Dictionary<int, int>>

{

{ 1 , new Dictionary<int, int>

{ {2, 0}, {5, 1}, {7, 1}, {9, 1} }

},

{ 2 , new Dictionary<int, int>

{ {1, 0}, {3, 0}, {4, 1}}

},

{ 3 , new Dictionary<int, int>

{ {2, 0}, {4,1 } }

},

{ 4 , new Dictionary<int, int>

{ {2, 0}, {3, 0} }

},

{ 5 , new Dictionary<int, int>

{ {1, 0}, {6, 0} }

},

{ 6 , new Dictionary<int, int>

{ {5, 0} }

},

{ 7 , new Dictionary<int, int>

{ {1, 0}, {8, 0}, {9, 1} }

},

{ 8 , new Dictionary<int, int>

{ {7, 0}, {9, 1} }

},

{ 9 , new Dictionary<int, int>

{ {8, 0}, {7, 0} }

},

{ 10 , new Dictionary<int, int>

{ {1, 0 } }

}

};

}

#### Representation of map as dictionary of dictionaries

### 23/02/2022

#### Solving the problem of implementing a map as a dictionary: instead of just updating a private global variable, there could be a ‘teleport’ method

### 24/02/2022

static void UpdatePlayerPos(int newPos)

{

int currentPos = playerPosition;

foreach (var locked in map[newPos])

{

if (locked.Value == 0 && locked.Key == newPos)

{

playerPosition = locked.Key;

}

else if (locked.Value == 1 && locked.Key == newPos)

{

Console.WriteLine("This door is locked. Please come back later.");

}

else

{

Console.WriteLine("You cannot enter this area from here.");

}

}

}

A screenshot of a computer

Description automatically generated

Figure 1:Trying from an area that cannot be accessed

A screenshot of a computer

Description automatically generated

Figure 2: Trying from an area that is locked

A screenshot of a computer

Description automatically generated

Figure 3: If you can access the area, it outputs nothing

#### UPDATED CODE:

static void UpdatePlayerPos(int newPos)

{

int currentPos = playerPosition;

bool found = false;

foreach (var locked in map[currentPos])

{

if (locked.Value == 0 && locked.Key == newPos)

{

playerPosition = locked.Key;

found = true;

}

else if (locked.Value == 1 && locked.Key == newPos)

{

Console.WriteLine("This door is locked. Please come back later.");

}

}

if (!found)

{

Console.WriteLine("You cannot enter this area from here.");

}

}

#### MOVED INTO SECTIONS CLASS, STARTED DIALOGUE FROM FILES

class Sections

{

private static Dictionary<int, Dictionary<int, int>> map;

private static int playerPosition = 1;

public Dictionary<string,string> ReadDialogueFile(int area)

{

string path = "area" + area + ".txt";

Dictionary<string, string> dialogue = new Dictionary<string, string>();

try

{

using (StreamReader sr = new StreamReader(path))

{

while (sr.Peek() >= 0)

{

int commaCount = 1;

int commaPosition = 0;

string line = sr.ReadLine();

while (commaCount < 2)

{

bool commaFound = false;

while (!commaFound)

{

if (line[commaPosition] == ',')

{

commaFound = true;

commaCount++;

}

else

{

commaPosition++;

}

string key = "";

string value = "";

for (int i = 0; i < commaPosition; i++)

{

key = Convert.ToString(line[i]);

}

for (int i = commaPosition; i < line.Length; i++)

{

value = Convert.ToString(line[i]);

}

dialogue.Add(key, value);

}

sr.ReadLine();

}

}

}

}

catch (Exception e)

{

Console.WriteLine("There was an error: ", e.ToString());

}

return dialogue;

}

#### Making a dictionary in order to split dialogue by its direction in a certain room

public Dictionary<string, string> ReadDialogueFile(int area)

{

string path = "area" + area + ".txt";

Dictionary<string, string> dialogue = new Dictionary<string, string>();

try

{

using (StreamReader sr = new StreamReader(path))

{

while (sr.Peek() >= 0)

{

int commaPosition = 0;

string line = sr.ReadLine();

bool commaFound = false;

while (!commaFound)

{

if (line[commaPosition] == ',')

{

commaFound = true;

}

else

{

commaPosition++;

}

}

string key = "";

string value = "";

for (int i = 0; i < line.Length; i++)

{

if (i < commaPosition && i != commaPosition)

{

key += line[i];

}

else if (i > commaPosition && i != commaPosition)

{

value += line[i];

}

}

dialogue.Add(key, value);

}

}

}

catch (Exception e)

{

Console.WriteLine("There was an error: ", e.ToString());

}

return dialogue;

}

### 27/02/2022

#### Ran into a problem with the UpdateMap() method as it will not allow me to change the 2d dictionary more than once in the foreach loop

Graphical user interface, text, application

Description automatically generated

Figure 4: Error message for UpdateMap()

#### SOLVED: instead of changing in the foreach loop, I instead found the rooms which had the ‘locked’ room in, and removed then readded them in a for loop:

A picture containing graphical user interface

Description automatically generated

5Figure 5: outputting the whole of the dictionary of dictionaries in order to see if the locked value had changed, value inputted was ‘9’

public void UpdateMap(int roomUnlocked)

{

int[] roomstounlock = new int[10];

int counter = 0;

foreach (var room in map)

{

foreach (var subdict in map[room.Key])

{

if (subdict.Key == roomUnlocked && subdict.Value == 1)

{

{

roomstounlock[counter] = room.Key;

counter++;

}

}

}

}

for (int i = 0; i < counter; i++)

{

map[roomstounlock[i]].Remove(roomUnlocked);

map[roomstounlock[i]].Add(roomUnlocked, 0);

}

}

#### 28/02/2022

#### Writing the method to check if a room needs to be moved to from the dialogue files.

public void CheckIfMove(Dictionary<string,string> dialogue, string direction)

{

string key = direction + playerPosition;

if (dialogue[key].Contains("@"))

{

string value = dialogue[key];

if (value.Length == 2)

{

UpdatePlayerPos(Convert.ToInt32(value[2]));

}

else

{

UpdatePlayerPos(10);

}

}

}

### 08/03/2022

#### Problems to fix:

#### Indexing issue with room 1 or starting room – crashes as key does not present in dictionary

#### Problem with outputting the text

#### Fixed issue with outputting the text:

A screenshot of a computer

Description automatically generated

Figure 6: Used room number instead of player position when checking the key, causing it to output that the key was not in the dictionary

public bool UpdatePlayerPos(int newPos)

{

int currentPos = playerPosition;

bool found = false;

bool unlocked = true;

foreach (var locked in map[currentPos])

{

if (locked.Value == 0 && locked.Key == newPos)

{

playerPosition = newPos;

found = true;

unlocked = false;

}

else if (locked.Value == 1 && locked.Key == newPos)

{

Console.WriteLine("This door is locked. Please come back later.");

found = true;

}

}

if (!found)

{

Console.WriteLine("You cannot enter this area from here.");

}

return unlocked;

}

#### Fixed issues with the starting room:

A screenshot of a computer

Description automatically generated

Figure 7: Moves back to the starting room if the playerPosition is 1, solving indexing issues

if (playerPosition == 1)

{

area = 0; //setting area to 0 to fix indexing issues

}

### 09/03/2022

#### How to solve puzzles: create an object with a getter method solved

#### Create object in sections, pass it into the locked method and if its true then

#### To solve the issue with complexity of the dictionary of dictionaries, I need to change the key of the second dictionary to contain the room AND the direction – can use concatenation. However, to get the result of the room out, I would need to switch the way its indexed to room + direction instead (so I can use [0] to get the room number).

#### Start on Puzzles class:

class Puzzles

{

private bool solved;

private int code;

List<int> numbers = new List<int>();

private Puzzles()

{

solved = false;

}

public bool Solved()

{

return solved;

}

public void CreatePuzzle()

{

Random random = new Random();

code = random.Next(900) + 100;

string codeAsString = Convert.ToString(code);

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

{

numbers.Add(Convert.ToInt32(codeAsString[i]));

}

}

public void CheckSolved(string userInput)

{

if (userInput == Convert.ToString(code))

{

solved = true;

}

}

}

### 10/03/2022

#### Changing mechanics (again) – allow the user to move, look or display map. The look will output dialogue from the text file or allow them to encounter enemies. Moving will be done using the UpdatePlayerPos() method.

#### UPDATED MECHANICS:

public void StartArea()

{

bool repeat = true;

while (repeat == true)

{

playerPosition = 1;

Console.WriteLine("----You are in the starting room!----");

Console.WriteLine("Please enter up/down/left/right to enter each area.");

Console.WriteLine(" ");

string input = Console.ReadLine().ToLower();

if (input == "up")

{

repeat = UpdatePlayerPos(7);

}

else if (input == "down")

{

repeat = UpdatePlayerPos(5);

}

else if (input == "left")

{

repeat = UpdatePlayerPos(10);

}

else if (input == "right")

{

area = 1;

repeat = UpdatePlayerPos(2);

}

else

{

Console.WriteLine("Invalid input. Please try again.");

}

}

}

public void MoveInArea()

{

if (playerPosition == 1)

{

StartArea();

}

else

{

bool move = false;

Console.WriteLine(" ");

Console.WriteLine("Info --- Room: " + (playerPosition) + " Health: ");

Console.WriteLine(" ");

while (move == false && playerPosition != 1)

{

Console.WriteLine("Either (M)ove, (L)ook or (D)isplay map?");

string choice = Console.ReadLine().ToLower();

if (choice == "m")

{

Console.WriteLine("Which room would you like to move to? Please enter the number from the map.");

string moveChoice = Console.ReadLine().ToLower();

int roomNumber;

if (Int32.TryParse(moveChoice, out roomNumber))

{

UpdatePlayerPos(roomNumber);

move = true;

};

}

else if (choice == "l")

{

Dictionary<string, string> areaDialogue = ReadDialogueFile(area);

Console.WriteLine("Look up/down/left/right?");

string lookChoice = Console.ReadLine().ToLower();

Console.WriteLine(OutputDialogue(area, lookChoice, areaDialogue));

}

else if (choice == "d")

{

DisplayMap();

}

else

{

Console.WriteLine("You have entered an invalid input. Try again.");

}

}

}

}

#### CHANGED MECHANICS AGAIN: instead of a puzzle, you find items. Each item will be related and added to a list of items. If you have all the items, you can open the room.

### 11/03/2022

#### Changing the OOP – Making a map class, and then an area class where enemies can randomly be created

### 16/03/2022

#### Making the enemies class and changing the data structure that stores the questions to be dynamic instead of static.

class Enemies

{

protected int damage;

protected int health;

protected string name;

List<string> questions = new List<string>();

List<string> answers = new List<string>();

public Enemies(string name, int health, int damage, string[,] Questions)

{

this.health = health;

this.damage = damage;

this.name = name;

Random rnd = new Random();

int length = Questions.GetLength(1);

for (int i = 0; i < health; i++)

{

foreach (var item in Questions)

{

Console.WriteLine(item);

}

int j = rnd.Next(4);

questions.Add(Questions[j, 0]);

answers.Add(Questions[j, 1]);

}

}

public virtual void Fight(Enemies enemy, int userHealth)

{

Console.WriteLine("You have encountered an enemy: " + name);

Console.WriteLine("Answer these questions to defeat them!");

while (health > 0)

{

Console.WriteLine(questions[health - 1]);

string userAnswer = Console.ReadLine().ToUpper();

if (userAnswer == questions[health - 1])

{

Console.WriteLine("That is correct! " + name + " took 1HP damage.");

health--;

questions.Remove(questions[health - 1]);

}

else

{

Console.WriteLine("That is incorrect! You took "+ damage + "HP damage.");

userHealth--;

}

if (health == 0 && questions.Count == 0)

{

Console.WriteLine("The enemy is defeated!");

enemy = null;

}

}

}

}

//class Bosses : Enemies

//{

//}

//class FinalBoss : Enemies

//{

//}

### 17/03/2022

#### Updates: changing mechanics – if you interact with a normal locked room, you must have a certain score to unlock it. To unlock room number 10 (or the ultimate boss room) you must have all the items from the ‘bosses’ to enter (using a list possibly?). Boss positions will be stored in the text files.

#### Score, health and game over updated with some (minor) user interface updates:

private static int score = 0;

private static int userHealth = 30;

public static void PlayGame()

{

Console.WriteLine("Welcome to the game! You have arrived from the future to save our planet from an evil robot takeover.");

Console.WriteLine("You must travel to each area, solving the puzzles along the way and defeating any enemies you come across with your knowledge.");

Console.ReadLine();

Map map = new Map();

map.DisplayMap();

bool gameover = false;

while (!gameover)

{

if (userHealth > 0)

{

map.MoveInArea(ref score, ref userHealth);

}

else

{

Console.WriteLine("You died! Your score: " + score + "point(s)!");

gameover = true;

}

}

}

#### Also working on unlocking the rooms and ran into an error – in C# you cannot use a dictionary in the same place it has been changed. Found an article (Shri, n.d.) that explains how to do it with a list instead so technically it is not iterating through something being changed.

### 23/03/2022

#### Updated using lists instead to loop through whilst the dictionary is being changed – now working

A screenshot of a computer

Description automatically generated

Figure 8: room 4, previously locked, is now where the player is.

public void DisplayMap()

{

Console.WriteLine("Locked: \*");

Console.WriteLine(" ");

Console.WriteLine("Scores needed to unlock each door:");

Console.WriteLine("Door 4: 2 points \n Door 5: 3 points \n Door 7: 4 points \n Door 9: 6 points \n Door 10: All bosses defeated");

string map = (".......... MAP: ............\n.........| 8 |---|\*9 |............\n...........|......................\n.........|\*7 | ...................\n...........|......................\n.|\*10|---| 1 |---| 2 |--| 3 |.....\n...........|.......|......|.......\n.........|\*5 | ..|\*4 |----|.......\n...........|......................\n.........| 6 | ...................");

if (unlockedRooms.Count > 0)

{

foreach (var room in unlockedRooms)

{

for (int i = 0; i < map.Length; i ++)

{

if (map[i] == '\*' && map[i+1] == room)

{

map[i] = ' ';

}

}

}

}

else

{

Console.WriteLine(map);

}

Console.WriteLine(" ");

}

}

#### Attempted to redo map with “locks” as \* which changes.

#### Some validation attempted

A screenshot of a computer

Description automatically generated

Figure 9: Does not output anything - possibly stuck in a loop? Another issue may be the use of lists

FIXED, needed to instantiate variable outside of the loop

### 25/03/2022

#### TASKS LEFT:

#### Fix map

#### Key graphic

#### Write dialogue

#### Write different enemies for each area

#### Write boss class

#### Dropped items add to inventory

#### Write ultimate boss class

#### Issues with outputting the text?

#### Issues with question mechanic – doesn’t work sometimes, step through to solve

### 29/03/2022

#### Creating a skip function in the fight:

}

case "S":

{

Console.WriteLine("If you skip a question, you will lose 3 health. Proceed? (y/n)");

string skipChoice = Console.ReadLine().ToUpper();

if (skipChoice == "Y")

{

questions.Remove(questions[health - 1]);

userHealth -= 3;

}

break;

}

A screenshot of a computer

Description automatically generated

Figure 10: question is removed and enemy health decreases

Text

Description automatically generated

Figure 11: player health is decreased by 3

#### Notes:

#### Fixed where the bosses are created

#### Ascii art implemented

#### Help/explanation added

#### Fixed problem where the boss battle went to -1 health.

### 04/04/2022

#### Organised files better

#### Music added

#### Fix boss fight as the randomly generated dialogue was not working

#### Make sure random works

#### Console.Clear()s fixed, ensured there were enough read lines.

### 06/04/2022

#### Finished code and annotations added

## Annotated code:

001: using System;

002: using System.Collections.Generic;

003: using System.IO;

004: using System.Linq;

005: using System.Text;

006: using System.Threading.Tasks;

007:

008: namespace Coursework

009: {

010:     class Enemies

011:     {

012:         //objective 2

013:         // creating variables/attributes

014:         protected int damage;

015:         protected int health;

016:         protected string name;

017:         protected bool defeated;

018:         protected int linesOfDialogue;

019:         protected List<string> dialogue = new List<string>();

020:         protected List<string> questions = new List<string>();

021:         protected List<string> answers = new List<string>();

022:

023:         public Enemies(string name, string[,] Questions , int health, int damage)

024:         {

025:             //assigns using parameters when the object is instantiated

026:             defeated = false;

027:             this.name = name;

028:             this.health = health;

029:             this.damage = damage;

030:             SetUpEnemy(Questions);

031:         }

032:         protected virtual void SetUpEnemy(string[,] Questions)

033:         {

034:             int j;

035:             Random rnd = new Random();

036:             int length = Questions.GetLength(1); //random question picked

037:             for (int i = 0; i < health; i++)//adds random question into the lists

038:             {

039:                 j = rnd.Next(0, length);

040:                 questions.Add(Questions[0, j]);

041:                 answers.Add(Questions[1, j]);

042:             }

043:             SetUpDialogue();

044:         }

045:         protected void SetUpDialogue()

046:         {

047:             using (StreamReader sr = new StreamReader("Dialogue/" + name + ".txt")) //reads from file and adds lines of dialogue to a list

048:             {

049:                 linesOfDialogue = 0;

050:                 while (sr.Peek() >= 0)

051:                 {

052:                     dialogue.Add(sr.ReadLine());

053:                     linesOfDialogue++;

054:                 }

055:             }

056:         }

057:         protected void SayDialogue()

058:         {

059:             Console.ForegroundColor = ConsoleColor.Blue; //changes to blue to allow the user to distinguish between other text (as this is quite a text heavy sequence)

060:             Random random = new Random();

061:             int line = random.Next(1, linesOfDialogue);

062:             Console.WriteLine("\n" + name + ": " + dialogue[line]);//outputs dialogue

063:             Console.ResetColor();

064:         }

065:

066:         protected void DisplayAsciiArt() //reads art from file and outputs to console

067:         {

068:             try

069:             {

070:                 using (StreamReader sr = new StreamReader("Art/" + name + "Art.txt"))

071:                 {

072:                     Console.WriteLine(sr.ReadToEnd());

073:                 }

074:             }

075:             catch (Exception e)

076:             {

077:                 Console.WriteLine("There was an error: " + e.ToString());

078:             }

079:         }

080:

081:         public virtual bool Fight(ref int userHealth)

082:         {

083:             DisplayAsciiArt();

084:             Console.WriteLine("\n----------------------------------------");

085:             Console.WriteLine("YOU HAVE ENCOUNTERED AN ENEMY: " + name);

086:             Console.ForegroundColor = ConsoleColor.Blue; //changes to blue

087:             Console.WriteLine(name + ": " + dialogue[0]);//outputs first line of dialogue

088:             Console.ResetColor();

089:             Console.WriteLine("Answer these questions to defeat them!\n");

090:             while (health >= 0 && !Program.dead)

091:             {

092:                 SayDialogue();

093:                 if (health != 0 && questions.Count != 0 && !Program.dead)

094:                 {

095:                     Console.ForegroundColor = ConsoleColor.Red;

096:                     Console.WriteLine("\nYour question: " + questions[health - 1] + "\n"); //asks question in red

097:                     Console.ResetColor();

098:                     Console.WriteLine("The enemy has " + health + "HP left. You have " + Program.userHealth + " HP left.");

099:                     Console.WriteLine("\n(F)ight or (S)kip?");

100:                     string choice = Console.ReadLine().ToUpper();

101:                     switch (choice)

102:                     {

103:                         case "F": //if the fight option is picked

104:                             {

105:                                 Console.WriteLine("Your answer: ");

106:                                 string userAnswer = Console.ReadLine().ToUpper();

107:                                 if (userAnswer == answers[health - 1].ToUpper()) //if the user inputs the correct answer

108:                                 {

109:                                     Console.WriteLine("That is correct! " + name + " took 1HP damage.");

110:                                     questions.Remove(questions[health - 1]); //removes the question and answer from the lists

111:                                     answers.Remove(answers[health - 1]);

112:                                     health--;

113:                                 }

114:                                 else //user takes damage

115:                                 {

116:                                     Console.WriteLine("That is incorrect! You took " + damage + "HP damage.");

117:                                     userHealth--;

118:                                 }

119:                                 break;

120:                             }

121:                         case "S"://if skip is picked

122:                             {

123:                                 Console.WriteLine("If you skip a question, you will lose 3 health. Proceed? (y/n)");

124:                                 string skipChoice = Console.ReadLine().ToUpper();

125:                                 if (skipChoice == "Y")

126:                                 {

127:                                     questions.Remove(questions[health - 1]); //removes the question

128:                                     answers.Remove(answers[health - 1]);

129:                                     userHealth -= 3; //decreases the health by 3

130:                                     health--;

131:                                 }

132:                                 break;

133:                             }

134:                         default: //validation

135:                             {

136:                                 Console.WriteLine("You have entered an invalid letter. Please try again.");

137:                                 break;

138:                             }

139:                     }

140:

141:                 }

142:                 else //when the enemy has 1 hp left and the last question has been entered

143:                 {

144:                     Console.WriteLine("The enemy is defeated!");

145:                     health--;

146:                     defeated = true;

147:                 }

148:

149:                 if (userHealth <= 0) //sets the global variable to true to stop the game.

150:                 {

151:                     Program.dead = true;

152:                 }

153:

154:             }

155:             return defeated;

156:

157:         }

158:

159:     }

160:     class Bosses : Enemies //inherited class for 'Bosses' which includes a drop

161:     {

162:         public string drop;

163:         public Bosses(string name, string[,] Questions, string drop , int health, int damage) : base(name, Questions , health, damage)

164:         {

165:             this.drop = drop;

166:         }

167:

168:     }

169:     class FinalBoss : Enemies //final boss inherited class

170:     {

171:         public FinalBoss(string name, string[,] Questions , int health, int damage) : base(name, Questions , health, damage)

172:         {

173:             SetUpEnemy(Questions);

174:             Console.WriteLine("You use Zeus' lightning bolt to charge the Pacman controller. \nThe crown is placed on your head, as you become the king of the robots!");

175:         }

176:         protected override void SetUpEnemy(string[,] Questions) //adds all questions as a final test

177:         {

178:             int length = Questions.GetLength(1);

179:             for (int i = 0; i < length; i++)

180:             {

181:                 questions.Add(Questions[0, i]);

182:                 answers.Add(Questions[1, i]);

183:             }

184:             SetUpDialogue();

185:         }

186:

187:     }

188:

189:     class Map

190:     {

191:         private Dictionary<int, Dictionary<int, int>> map; //decalaring a dictionary called map, used throughout to check if rooms are connected or locked

192:         public int playerPosition = 1;

193:         private int area = 0;

194:         public List<string> inventory = new List<string>();//contains all items dropped by the bosses

195:         private List<int> unlockedRooms = new List<int>();//checks when outputting map

196:         public Dictionary<string, string> ReadDialogueFile(int area) //reads all dialogue from the files into the dictionary

197:         {

198:             string path = "Area/area" + area + ".txt";

199:             Dictionary<string, string> dialogue = new Dictionary<string, string>();

200:             try

201:             {

202:                 using (StreamReader sr = new StreamReader(path))

203:                 {

204:                     while (sr.Peek() >= 0)

205:                     {

206:                         int commaPosition = 0;

207:                         string line = sr.ReadLine();

208:                         bool commaFound = false;

209:                         while (!commaFound)

210:                         {

211:                             if (line[commaPosition] == ',') //finds comma

212:                             {

213:                                 commaFound = true;

214:                             }

215:                             else

216:                             {

217:                                 commaPosition++;

218:                             }

219:                         }

220:                         string key = "";

221:                         string value = "";

222:                         for (int i = 0; i < line.Length; i++)

223:                         {

224:                             if (i < commaPosition && i != commaPosition)

225:                             {

226:                                 key += line[i]; //splits all before the comma into the key

227:                             }

228:                             else if (i > commaPosition && i != commaPosition)

229:                             {

230:                                 value += line[i]; //splits all after the comma into the value

231:                             }

232:

233:                         }

234:

235:                         dialogue.Add(key, value); //adds to dictionary

236:

237:                     }

238:                 }

239:             }

240:             catch (Exception e)

241:             {

242:                 Console.WriteLine("There was an error: ", e.ToString());

243:             }

244:             return dialogue;

245:         }

246:

247:

248:         public void UnlockRoom(int roomUnlocked) //unlocks the room in the dictionary

249:         {

250:

251:             int[] roomsToUnlock = new int[10];

252:             int counter = 0;

253:

254:             foreach (var room in map)

255:             {

256:                 foreach (var subdict in map[room.Key])

257:                 {

258:                     if (subdict.Key == roomUnlocked && subdict.Value == 1) //finds all instances of the room that needs to be unlocked in the dictionary

259:                     {

260:                         {

261:                             roomsToUnlock[counter] = room.Key;

262:                             counter++;

263:

264:                         }

265:                     }

266:                 }

267:             }

268:             for (int i = 0; i < counter; i++)

269:             {

270:                 map[roomsToUnlock[i]].Remove(roomUnlocked); //removes and adds a new value with '0' meaning unlocked

271:                 map[roomsToUnlock[i]].Add(roomUnlocked, 0);

272:             }

273:             unlockedRooms.Add(roomUnlocked);

274:

275:         }

276:

277:         public void StartArea() //allows the user to travel to each 'area' of the map

278:         {

279:             Program.PlayMusic("gamemusic");

280:             Console.WriteLine("Push enter to continue.");

281:             Console.ReadLine();

282:             Console.Clear();

283:             bool repeat = true;

284:             while (repeat == true) //keeps asking if the player is not moved

285:             {

286:                 playerPosition = 1;

287:                 DisplayMap();

288:                 Console.WriteLine("\nYou are in the starting room!\nFrom here you can travel in your time machine to each area.");

289:                 Console.WriteLine("Please enter up/down/left/right to enter each area.");

290:                 string input = Console.ReadLine().ToLower();

291:                 switch (input)

292:                 {

293:                     case "up":

294:                         {

295:                             Program.PlayMusic("arcademusic");

296:                             repeat = UpdatePlayerPos(7); //moves the player and returns if it is possible

297:                             area = 3;

298:                             break;

299:                         }

300:                     case "down":

301:                         {

302:                             Program.PlayMusic("medievalmusic");

303:                             area = 2;

304:                             repeat = UpdatePlayerPos(5);

305:                             break;

306:                         }

307:                     case "left":

308:                         {

309:                             Program.PlayMusic("robotmusic");

310:                             area = 4;

311:                             repeat = UpdatePlayerPos(10);

312:                             break;

313:                         }

314:                     case "right":

315:                         {

316:                             area = 1;

317:                             Program.PlayMusic("greekmusic");

318:                             repeat = UpdatePlayerPos(2);

319:                             break;

320:                         }

321:                     default: //validation

322:                         {

323:                             Console.WriteLine("Invalid input. Please try again. Push enter.");

324:                             Console.ReadLine();

325:                             break;

326:                         }

327:                 }

328:             }

329:         }

330:

331:         public void MoveInArea() //allows the user to move from room to room, look, show the map or get help

332:         {

333:             while (!Program.dead && Program.userHealth > 0)

334:             {

335:                 if (playerPosition == 1) //if they are in room 1

336:                 {

337:                     StartArea();

338:                 }

339:                 else

340:                 {

341:                     bool move = false;

342:                     Dictionary<string, string> areaDialogue = ReadDialogueFile(area);

343:                     while (move == false && playerPosition != 1 && Program.dead == false)

344:                     {

345:                         Console.Clear();

346:                         Console.WriteLine("Info --- Room: " + playerPosition + " -  Health: " + Program.userHealth + " - Score: " + Program.score + " ---"); //displays information

347:                         Console.WriteLine(areaDialogue[Convert.ToString(playerPosition)]);

348:                         Console.WriteLine("\nEither (M)ove, (L)ook, (D)isplay your map or (H)elp?");

349:                         string choice = Console.ReadLine().ToLower();

350:

351:                         //example of objective 7(i), however it is achieved throughout the code

352:

353:                         switch (choice)

354:                         {

355:                             case "m":

356:                                 {

357:                                     Console.WriteLine("\nWhich room would you like to move to? Please enter the number from the map.");

358:                                     string moveChoice = Console.ReadLine();

359:                                     if (Int32.TryParse(moveChoice, out int roomNumber))

360:                                     {

361:                                         UpdatePlayerPos(roomNumber); //moves player

362:                                         move = true;

363:                                     }

364:                                     else //validation

365:                                     {

366:                                         Console.WriteLine("Please enter a valid input.");

367:                                     };

368:                                     break;

369:                                 }

370:                             case "l":

371:                                 {

372:                                     Console.WriteLine("Look up/down/left/right?");

373:                                     string lookChoice = Console.ReadLine().ToLower();

374:                                     string output = OutputDialogue(area, lookChoice, areaDialogue  ); //outputs dialogue when looking

375:                                     Console.WriteLine(output);

376:                                     if (!Program.dead) //stops repeated output if dead

377:                                     {

378:                                         Console.WriteLine("Press enter to continue.");

379:                                         Console.ReadLine();

380:                                     }

381:                                     break;

382:                                 }

383:                             case "d": //display map

384:                                 {

385:                                     DisplayMap();

386:                                     Console.WriteLine("Press enter to continue.");

387:                                     Console.ReadLine();

388:                                     break;

389:                                 }

390:                             case "h": //help, objective 7(ii)

391:                                 {

392:                                     Console.WriteLine("Select a room to move to. From there, you can look around and search for enemies.\nYou will only find enemies when looking around, not moving.\nEach enemy increases your score by 1 point, and each boss increases it by 2. Look at the score guide on your map to find out how many points it takes to unlock a certain room.");

393:                                     Console.WriteLine("Press enter to continue.");

394:                                     Console.ReadLine();

395:                                     break;

396:                                 }

397:                             default: //validation

398:                                 {

399:                                     Console.WriteLine("You have entered an invalid input. Try again.");

400:                                     Console.WriteLine("Press enter to continue.");

401:                                     Console.ReadLine();

402:                                     break;

403:                                 }

404:

405:                         }

406:                     }

407:                 }

408:             }

409:         }

410:

411:

412:         public void PlayEnemy( ) //used to instantiate enemies based on areas

413:         {

414:             // objective 2(i)

415:             // objective 2(iii)

416:             Enemies enemy = null;

417:             if (area == 1)

418:             {

419:                 enemy = new Enemies("Cyclops", Question.Questions , 3, 1); //objective 5(ii)

420:             }

421:             else if (area == 2)

422:             {

423:                 enemy = new Enemies("Armour stand", Question.Questions , 3, 1);

424:             }

425:             else if (area == 3)

426:             {

427:                 enemy = new Enemies("Ghost", Question.Questions , 3, 1);

428:             }

429:             bool defeated = enemy.Fight(ref Program.userHealth);

430:             if (defeated && Program.userHealth > 0)

431:             {

432:                 Program.score++;

433:                 enemy = null; //object lifecycle

434:             }

435:

436:         }

437:

438:         private bool CheckBossDefeated(Bosses enemy) //checks if the user has defeated the boss of the area by checking if they have its drop in their inventory

439:         {

440:             //objective 5(iii)

441:             bool itemLeft = false;

442:             if (inventory.Contains(enemy.drop))

443:             {

444:                 itemLeft = true;

445:             }

446:             return itemLeft;

447:         }

448:

449:         public void PlayBoss(ref int score) //instantiates bosses based on area

450:         {

451:             // objective 2(i)

452:             // objective 2(iii)

453:             Bosses enemy = null;

454:             Random random = new Random(); //objective 4(i), 5(i)

455:             int damage = random.Next(3);

456:             if (playerPosition == 4)

457:             {

458:                 enemy = new Bosses("Zeus", Question.Questions, "Lightning bolt" , 5, damage);

459:             }

460:             else if (playerPosition == 6)

461:             {

462:                 enemy = new Bosses("King", Question.Questions, "Crown" , 5, damage);

463:             }

464:             else if (playerPosition == 9)

465:             {

466:                 enemy = new Bosses("Pacman", Question.Questions, "Game controller" , 5, damage);

467:             }

468:

469:             if (false == CheckBossDefeated(enemy)) //playing the boss and adding their drop to the inventory

470:             {

471:                 bool defeated = enemy.Fight(ref Program.userHealth);

472:                 if (defeated == true && Program.userHealth > 0)

473:                 {

474:                     Console.WriteLine("\nYou have found a " + enemy.drop + ". It has been added to your inventory. Push enter to continue.");  //objective 5(iii)

475:                     inventory.Add(Convert.ToString(enemy.drop));

476:                     score += 2;

477:                     Console.ReadLine();

478:                 }

479:

480:             }

481:             enemy = null; //object lifecycle

482:         }

483:         private string OutputDialogue(int area, string direction, Dictionary<string, string> dialogue  )

484:         {

485:             Random random = new Random();

486:             string output = "";

487:             int chance = random.Next(4); //chance of enemy appearing

488:             direction = direction.ToLower();

489:             if (direction == "up" || direction == "down" || direction == "left" || direction == "right")

490:             {

491:                 if (chance == 1) //random enemy appears, , objective 2(ii)

492:                 {

493:                     PlayEnemy();

494:                 }

495:                 else //outputs dialogue from dictionary

496:                 {

497:                     string key = direction + playerPosition;

498:                     foreach (var value in dialogue)

499:                     {

500:                         if (value.Key == key)

501:                         {

502:                             output = value.Value;

503:                         }

504:                     }

505:                 }

506:             }

507:             else //validation

508:             {

509:                 Console.WriteLine("You have entered an invalid input. Push enter.");

510:             }

511:             return output;

512:         }

513:         public bool CheckScore(int newPos, int score) //checks if door can be unlocked

514:         {

515:             bool unlock = false;

516:             if ((newPos == 4 && score >= 2) || (newPos == 5 && score >= 3) || (newPos == 7 && score >= 4) || (newPos == 9 && score >= 6))

517:             {

518:                 unlock = true;

519:             }

520:

521:             return unlock;

522:         }

523:

524:         private void PlayUltimateBoss() //instantiates ultimate boss

525:         {

526:             FinalBoss robot = new FinalBoss("Robot", Question.Questions, (Question.Questions.Length)/2 , 3); //objective 5(iv)

527:             bool defeated = robot.Fight(ref Program.userHealth);

528:             if (defeated == true && Program.userHealth > 0)

529:             {

530:                 Console.WriteLine("Your remaining health was added to your point score.");

531:                 Program.score += Program.userHealth;

532:                 Program.userHealth = 0; //sets the health to zero to break out of the game loop

533:             }

534:         }

535:

536:         public bool UpdatePlayerPos(int newPos) //moves player to a room if it is not locked

537:         {

538:             bool found = false;

539:             bool unlocked = true;

540:             Dictionary<int, int> valuePairs = new Dictionary<int, int>(map[playerPosition]); //sub-dictionary of the main map

541:             List<int> keys = new List<int>(); //split into lists as the dictionary has been changed

542:             List<int> values = new List<int>();

543:

544:             foreach (var key in valuePairs) //adds to list

545:             {

546:                 keys.Add(key.Key);

547:                 values.Add(key.Value);

548:             }

549:             int counter = 0;

550:             if (newPos == 10 && inventory.Count >= 3) //checks if they are moving to the final ultimate boss room

551:             {

552:                 Console.WriteLine("You have all of the pieces to defeat the robot!");

553:                 PlayUltimateBoss();

554:                 unlocked = false;

555:             }

556:             else

557:             {

558:                 foreach (var locked in keys)//goes through each value in the list

559:                 {

560:                     if (values[counter] == 0 && locked == newPos) //checks if the room is unlocked and is the room that the player wants to move to

561:                     {

562:                         playerPosition = newPos;

563:                         found = true;

564:                         unlocked = false;

565:                     }

566:                     else if (values[counter] == 1 && locked == newPos) //checks if the room is locked and is the new position

567:                     {

568:                         unlocked = CheckScore(newPos, Program.score);

569:                         if (unlocked == false)

570:                         {

571:                             Console.WriteLine("This door is locked. Please come back later.");

572:                             found = true;

573:                         }

574:                         else

575:                         {

576:                            found = UserUnlockRoom(newPos); //asks the user if they want to unlock the room

577:                         }

578:                     }

579:                     counter++;

580:

581:                 }

582:                 if (!found && playerPosition != newPos) //if there is not a path

583:                 {

584:                     Console.WriteLine("You cannot enter this area from here.");

585:                     Console.ReadLine();

586:                 }

587:                 if ((playerPosition == 4 || playerPosition == 6 || playerPosition == 9)) //if moving to a boss battle room

588:                 {

589:                     PlayBoss(ref Program.score);

590:                 }

591:             }

592:             return unlocked;

593:         }

594:

595:         private bool UserUnlockRoom(int newPos)//asks user if they want to unlock the room

596:         {

597:             bool found = false;

598:             Console.WriteLine("You can unlock this room as your score is " + Program.score + ". Unlock? (y/n)");

599:             if (Console.ReadLine().ToUpper() == "Y")

600:             {

601:                 UnlockRoom(newPos);

602:                 DisplayLockArt();

603:                 found = true;

604:                 Console.ReadLine();

605:             }

606:             else

607:             {

608:                 Console.WriteLine("I don't see why you wouldn't, but come back later if you want to. Push enter to continue.");

609:                 Console.ReadLine();

610:             }

611:             return found;

612:         }

613:

614:         private void DisplayLockArt()//reads lock from file

615:         {

616:             using (StreamReader sr = new StreamReader("lockopen.txt"))

617:             {

618:                 Console.WriteLine(sr.ReadToEnd());

619:             }

620:             Console.WriteLine("Room unlocked! You may now move to this room. (Push enter to continue.)");

621:         }

622:         public Map()//defined graph as dictionary of dictionaries set in constructor

623:         {

624:             //objective 3(i)

625:

626:             map = new Dictionary<int, Dictionary<int, int>>

627:             {

628:                 { 1 , new Dictionary<int, int>

629:                     { {2, 0}, {5, 1}, {7, 1}, {10, 1} }

630:                 },

631:                 { 2 , new Dictionary<int, int>

632:                     { {1, 0}, {3, 0}, {4, 1}}

633:                 },

634:                 { 3 , new Dictionary<int, int>

635:                     { {2, 0}, {4, 1} }

636:                 },

637:                 { 4 , new Dictionary<int, int>

638:                     { {2, 0}, {3, 0} }

639:                 },

640:                 { 5 , new Dictionary<int, int>

641:                     { {1, 0}, {6, 0} }

642:                 },

643:                 { 6 , new Dictionary<int, int>

644:                     { {5, 0} }

645:                 },

646:                 { 7 , new Dictionary<int, int>

647:                     { {1, 0}, {8, 0}, {9, 1} }

648:                 },

649:                 { 8 , new Dictionary<int, int>

650:                     { {7, 0}, {9, 1} }

651:                 },

652:                 { 9 , new Dictionary<int, int>

653:                     { {8, 0}, {7, 0} }

654:                 },

655:                 { 10 , new Dictionary<int, int>

656:                     { {1, 0 } }

657:                 }

658:

659:             };

660:

661:         }

662:         public void DisplayMap() //displays the map depending on which rooms are unlocked in the list 'unlockedRooms'

663:         {

664:             Console.Clear();

665:             Console.WriteLine("-.-.-.-.-.- YOUR MAP: -.-.-.-.-.-");

666:             Console.WriteLine("\nYou can access the map by choosing (D)isplay when asked.\nLocked: \*");

667:             Console.WriteLine("Scores needed to unlock each door:");

668:             Console.WriteLine(" - Door 4: 2 points \n - Door 5: 3 points \n - Door 7: 4 points \n - Door 9: 6 points \n - Door 10: All bosses defeated\n");

669:             using (StreamReader sr = new StreamReader("map.txt"))

670:             {

671:                 string map = sr.ReadToEnd();

672:                 string newmap = "";

673:                 if (unlockedRooms.Count > 0)

674:                 {

675:                     for (int i = 0; i < map.Length - 1; i++)

676:                     {

677:                         string nextChar = Convert.ToString(map[i + 1]);

678:                         if (int.TryParse(nextChar, out int nextCharInt))

679:                         {

680:                             if (map[i] == '\*' && unlockedRooms.Contains(nextCharInt) == true)

681:                             {

682:                                 newmap += " ";

683:                             }

684:                             else

685:                             {

686:                                 newmap += map[i];

687:                             }

688:                         }

689:                         else

690:                         {

691:                             newmap += map[i];

692:                         }

693:                     }

694:                     newmap += map[map.Length - 1];

695:                     Console.WriteLine(newmap);

696:                 }

697:                 else

698:                 {

699:                     Console.WriteLine(map);

700:                 }

701:             }

702:         }

703:     }

704:     class Question

705:     {

706:         private static bool fileWritten = false;

707:         public static string[,] Questions;

708:         //Importing the questions with a file either from kahoot or pre-written using the function in the program

709:         private static void GetQuestionsFile(ref bool questionsLoaded) //objective 1(i)

710:         {

711:             string path;

712:

713:             if (!fileWritten)//if the user is entering their own

714:             {

715:                 Console.WriteLine("Please enter your file name. It must be in the 'bin, debug' folder of this program.");

716:                 path = Console.ReadLine();

717:             }

718:             else //when called using the WriteQuestions() method

719:             {

720:                 path = "WriteLines.txt";

721:             }

722:             try

723:             {

724:                 using (StreamReader sr = new StreamReader(path))

725:                 {

726:

727:                     int lineNumber = 0;

728:                     int numberOfLines = File.ReadAllLines(path).Length;//number of lines using system.io

729:                     Questions = new string[2, numberOfLines];

730:                     while (sr.Peek() >= 0) //while not at end of file

731:                     {

732:                         string line = sr.ReadLine();

733:                         string question = "";

734:                         string answer = "";

735:                         int commaPosition = 0;

736:                         bool commaFound = false;

737:                         while (!commaFound)

738:                         {

739:                             if (line[commaPosition] == ',') //finds where the comma is in the line

740:                             {

741:                                 commaFound = true;

742:                             }

743:                             else

744:                             {

745:                                 commaPosition++;

746:                             }

747:                         }

748:                         for (int i = 0; i < line.Length; i++) //assigning the string before the comma to 'question' and after to 'answer'

749:                         {

750:                             if (i < commaPosition && i != commaPosition)

751:                             {

752:                                 question += line[i];

753:                             }

754:                             else if (i > commaPosition && i != commaPosition)

755:                             {

756:                                 answer += line[i];

757:                             }

758:

759:                         }

760:                         //adding into 2d array

761:                         Questions[0, lineNumber] = question;

762:                         Questions[1, lineNumber] = answer;

763:                         lineNumber++;

764:

765:                     }

766:                     Console.WriteLine("Questions added! Push enter to continue.");

767:                     Console.ReadLine();

768:                     questionsLoaded = true;

769:                 }

770:             }

771:             catch (Exception e) //validation

772:             {

773:                 Console.WriteLine("The file could not be read:");

774:                 Console.WriteLine(e.Message + ". Push enter.");

775:                 questionsLoaded = false; //cannot play game if this is false

776:                 Console.ReadLine();

777:             }

778:

779:

780:         }

781:

782:         private static void WriteQuestions(ref bool questionsLoaded) //objective 1(ii)

783:         {

784:             Console.WriteLine("How many questions would you like to enter?");

785:             string initalInput = Console.ReadLine();

786:             int questionNumbers;

787:             if (int.TryParse(initalInput, out questionNumbers))

788:             {

789:                 string text = "";

790:                 for (int i = 0; i < questionNumbers; i++)

791:                 {

792:                     Console.WriteLine("Question " + (i + 1) + ": ");//asks for the questions

793:                     text += Console.ReadLine() + ",";

794:                     if (i != questionNumbers - 1)

795:                     {

796:                         Console.WriteLine("Answer " + (i + 1) + ": ");

797:                         text += Console.ReadLine() + Environment.NewLine;//asks for answers

798:                     }

799:                     else

800:                     {

801:                         Console.WriteLine("Answer " + (i + 1) + ": "); //last line without an empty line in the array

802:                         text += Console.ReadLine();

803:                     }

804:

805:                 }

806:                 using (StreamWriter outputFile = new StreamWriter("WriteLines.txt"))//writes to a file to be loaded in LoadFile() [objective 1(iia)

807:                 {

808:                     outputFile.WriteLine(text);

809:                 }

810:                 fileWritten = true;

811:                 GetQuestionsFile(ref questionsLoaded);

812:

813:             }

814:             else //validation

815:             {

816:                 Console.WriteLine("You have entered an invalid number. Push enter.");

817:                 Console.ReadLine();

818:             }

819:         }

820:         public bool Menu(ref bool endGame, ref bool questionsLoaded) //start menu

821:         {

822:             Program.PlayMusic("gamemusic");

823:             bool playGame = false;

824:             Console.Clear();

825:             Console.WriteLine("---------------------Welcome to the game!---------------------");

826:             Console.WriteLine("Play game (P), Load file (L), Write questions (W), Close game (C)");

827:             string input = Console.ReadLine().ToUpper();

828:             switch (input)

829:             {

830:                 case "L": //loads questions

831:                     {

832:                         GetQuestionsFile(ref questionsLoaded);

833:                         break;

834:                     }

835:                 case "C": //closes program

836:                     {

837:                         Console.WriteLine("Thank you for playing the game!");

838:                         endGame = true;

839:                         break;

840:                     }

841:                 case "W": //write questions

842:                     {

843:                         WriteQuestions(ref questionsLoaded);

844:                         break;

845:                     }

846:                 case "P": //plays game if there are questions in the dictionary

847:                     {

848:                         if (questionsLoaded == true)

849:                         {

850:                             playGame = true;

851:                         }

852:                         else

853:                         {

854:                             Console.WriteLine("Please enter your questions before beginning the game!");

855:                             Console.ReadLine();

856:                         }

857:                         break;

858:                     }

859:                 default://validation

860:                     {

861:                         Console.WriteLine("Not a valid input. Please try again.");

862:                         Console.ReadLine();

863:                         break;

864:                     }

865:

866:             }

867:             return playGame;

868:         }

869:     }

870:

871:     internal class Program

872:     {

873:         //global variables

874:         public static int score;

875:         public static int userHealth;

876:         public static bool dead;

877:

878:         public static void PlayMusic(string filename) //plays music from file

879:         {

880:             System.Media.SoundPlayer player = new System.Media.SoundPlayer();

881:             player.SoundLocation = ("Music/" + filename + ".wav");

882:             player.Play();

883:         }

884:         private static void PlayGame() //plays the game

885:         {

886:             Console.WriteLine("Welcome to the game! You have arrived from the future to save our planet from an evil robot takeover.\nSelect a room to move to. From there, you can look around and search for enemies.\nYou will only find enemies when looking around, not moving.");

887:             Console.WriteLine("You must travel to each area, solving the puzzles along the way and defeating any enemies you come across with your knowledge.");

888:             Map map = new Map(); //sets up map

889:

890:             //sets variables up for new game

891:             userHealth = 30; //objective 5(v)

892:             score = 0; //objective 6(i)

893:             dead = false;

894:             map.inventory.Clear();

895:             map.playerPosition = 1;

896:

897:

898:             while (!dead && userHealth >= 0) //loops until dead

899:             {

900:                 if (userHealth > 0)

901:                 {

902:                     map.MoveInArea();

903:                 }

904:                 else

905:                 {

906:                     Console.WriteLine("You have won the game! Well done, the world is now safe from the robots.\nYour score was: " + score + "."); //objective 6(i)

907:                     Console.ReadLine();

908:                     userHealth--; //breaks out of the loop

909:                 }

910:

911:             }

912:             Console.Clear();

913:             if (dead == true)

914:             {

915:                 Console.WriteLine("You died! Your score: " + score + " point(s)!"); //objective 6(i)

916:                 Console.ReadLine();

917:             }

918:

919:         }

920:         static void Main(string[] args)

921:         {

922:             bool gameover = false;

923:             bool playGame;

924:             bool questionsLoaded = false;

925:             Question questions = new Question();

926:             while (!gameover)//continues until close game is selected

927:             {

928:                 playGame = questions.Menu(ref gameover, ref questionsLoaded);

929:                 if (playGame == true)

930:                 {

931:                     PlayGame();

932:                 }

933:             }

934:             Console.ReadLine();

935:         }

936:     }

937: }

# Testing:

## Testing criteria and results:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Section** | **Test no.** | **Purpose** | **Input** | **Expected output** | **Test type** | **Pass/Fail** |
| Menu | 1 | To check that only specific characters can be input | 123 | Tell the user that it is “not a valid input” | Erroneous | Pass |
|  | 2 | To check that the user has inputted the questions that will be asked during the game | “P” (without questions loaded) | Asks the user to input their questions before beginning | Erroneous | Pass |
|  | 3 | To check if the program can load questions into the game | “L” then “doesntexist.txt”  (with a file that does not exist) | Asks the user to input the name of a file in the debug folder. The user inputs a file that does not exist, so it outputs that the file cannot be read. | Erroneous | Pass |
|  | 4 | ^ | “L” then “questions.txt” (with a file that exists) | Asks the user to input the name of a file in the debug folder. It outputs that the questions have been successfully added. | Normal | Pass |
|  | 5 | To check that the program allows the user to write questions into the game | “W” then “ABC” (invalid number of questions) | Asks the user how many questions they would like to answer. Output that they have “entered an invalid number of questions.” | Erroneous | Pass |
|  | 6 | ^ | “W”  “2”  “a”  “a”  “b”  “b” | Asks the user how many questions they want to enter, then asks them to input them one at a time. | Normal | Pass |
|  | 7 | To check that the program closes game | “C” | Says “thank you for playing” and stops the program. | Normal | Pass |
|  | 8 | To check the program can start the game | “P” (with questions loaded in) | Outputs the starting dialogue, then clears the console to output the map. | Normal | Pass |
| Starting Room | 1 | Verify that entering an invalid input does not crash the program | “Forward” | Outputs that the user has made an invalid choice. | Erroneous | Pass |
|  | 2 | Checking that a user cannot enter a locked room | “Down”  or  “Up”  or  “Left” | Outputs that the room is locked. | Normal | Pass |
|  | 3 | Verifying that the program can move a player to an unlocked room | “Right” | Menu switches to the room menu | Normal | Pass |
| Room menu | 1 | Verify that entering an invalid input does not crash the program | “ABC” | Outputs that the user has made an invalid choice. | Erroneous | Pass |
|  | 2 | Verify that the program outputs help with the game | “H” | Outputs help dialogue | Normal | Pass |
|  | 3 | Check the program can display the map | “D” | Displays the map to the user with the correct rooms locked/unlocked | Normal | Pass |
|  | 4 | Verifying that the program can move a player to an unlocked room | “M” (is unlocked and there is a path) | Moves to the correct room and displays the room menu for that room | Normal | Pass |
|  | 5 | ^ | “M” (is unlocked but there is not a path) | Outputs that you cannot move to the specified room. | Erroneous | Pass |
|  | 6 | Checks that the program moves to the look menu | “L” | Moves to the look menu | Normal | Pass |
|  | 7 | Checks that if the user has the correct score and has moved to a boss room, the | “4” with a score of 2 | Moves to the room and starts the Boss fight | Normal | Pass |
| Look menu | 1 | To check that if the user inputs a direction, the program outputs the correct piece of dialogue | Room 3 – “down” | Outputs correct dialogue | Normal | Pass |
|  | 2 | Verify that entering an invalid input does not crash the program | “forward” | Outputs that the user has made an invalid choice. | Erroneous | Pass |
|  | 3 | Verify that if an enemy is encountered the program switches to an enemy fight | “up” | Outputs the enemy ASCII art and switches to the fight sequence | Normal | Pass |
| Fight sequence  (This is the same for bosses, enemies, and the ultimate boss) | 1 | Verify that the user loses 1HP if they get a question wrong | “F” then “wronganswer” | Outputs that the user is incorrect and shows their new total health | Normal | Pass |
|  | 2 | Verify that if the user defeats an enemy, their score increases by 1 for a normal enemy, 2 for a boss and their remaining total health for the ultimate boss | “F” then correct answers in order given | Outputs that the user is correct and increases their score in the move menu | Normal | Pass |
|  | 3 | Verify that if Skip is chosen, the users health is decreased by 3, and the question is removed | “S” then “y” | Asks the user if they are willing to lose 3 health, then shows their new health | Normal | Pass |
|  | 4 | To verify that if the user has 0HP left they die, and the program stops | “F” then continuous incorrect answers of “a” | Outputs the users final score and moves back to the main menu | Normal | Pass |
|  | 5 | Verify that the correct dialogue is output for each enemy | Randomly encountering an enemy | Outputs the correct dialogue from a file | Normal | Pass |

## Screenshots:

|  |  |  |
| --- | --- | --- |
| Section | Test number | Result(s) |
| Menu | 1 | Graphical user interface, application  Description automatically generated |
|  | 2 | Graphical user interface, application, Word  Description automatically generated |
|  | 3 | Graphical user interface, application, PowerPoint  Description automatically generated |
|  | 4 | A screenshot of a computer  Description automatically generated  IN GAME USE:  Text  Description automatically generated |
|  | 5 | Graphical user interface, application  Description automatically generated |
|  | 6 | Graphical user interface, application  Description automatically generated  IN GAME USE:  Text  Description automatically generated |
|  | 7 | A screenshot of a computer  Description automatically generated |
|  | 8 | Graphical user interface, application  Description automatically generated  Graphical user interface, application  Description automatically generated |
| Starting room | 1 | A screenshot of a computer  Description automatically generated |
|  | 2 | Graphical user interface, application  Description automatically generated  Graphical user interface, application, Word  Description automatically generated  Graphical user interface, application, Word  Description automatically generated |
|  | 3 | A screenshot of a computer  Description automatically generated  A screenshot of a computer  Description automatically generated |
| Room menu | 1 | Graphical user interface, application  Description automatically generated |
|  | 2 | Graphical user interface, application  Description automatically generated |
|  | 3 | A screenshot of a computer  Description automatically generated  3 Output  A screenshot of a computer  Description automatically generated  4 Input |
|  | 4 | A screenshot of a computer  Description automatically generated  5 Input  A screenshot of a computer  Description automatically generated  6 Output |
|  | 5 | Graphical user interface, application  Description automatically generated |
|  | 6 | A screenshot of a computer  Description automatically generated |
|  | 7 | 7 Input    8 Output |
| Look Menu | 1 | Graphical user interface, application, Excel  Description automatically generated  Graphical user interface, text, application  Description automatically generated  9 area1.txt - contents of the file |
|  | 2 | Graphical user interface, application  Description automatically generated |
|  | 3 | Graphical user interface, application  Description automatically generated |
| Fight | 1 | Graphical user interface, application, Word  Description automatically generated |
|  | 2 | A screenshot of a computer  Description automatically generated with medium confidence |
|  | 3 | A screenshot of a computer  Description automatically generated with medium confidence |
|  | 4 | Text  Description automatically generated  10 Input  A picture containing text  Description automatically generated  11 Output |
|  | 5 | Text  Description automatically generated  File contents:  Graphical user interface, application  Description automatically generated |

## File contents:

Example area:

Graphical user interface, text, application

Description automatically generated

Example dialogue:

Graphical user interface, text, application

Description automatically generated

Boss text

Example question file:

Graphical user interface, text, application

Description automatically generated

# Evaluation:

## Meeting user objectives:

1. **The user must be able to enter their own questions**
   1. **Users can enter either by loading a text file**
   2. **Or by writing their own questions using the console**
      1. **Questions should be able to be saved in a text file and then reused again**

1i.) The objective has been met, as users can enter their own text file, which is done by asking the user to enter a file name that they have already placed in the debug folder. This is a good solution as it allows the user to import questions from sites such as ‘Quizlet’, which has a built-in function to export a card set in the same way that the program reads it. It is also useful for giving a test to multiple students at the same time, as the questions can be transferred easily.

1ii.) The objective has been met, as users can choose to write their own questions. The program asks them to enter the number of questions and then allows them to enter their question and answer them one by one. The questions are then saved in a file called “WriteLines.txt” which meets objective (a). This solution is especially useful for younger users who may not understand how to import files or only want to test a few smaller questions.

1. **There must be enemies to fight against**
   1. **Diverse types of enemies should be in different areas**
   2. **They should be randomised to avoid repetition**
   3. **There should be different ‘levels’ of enemies, like ‘normal’ or ‘boss’ enemies**

2i.) This objective has been met well, as there is a specific type of ‘boss’ and ‘enemy’ for each area, with different ASCII art and dialogue as well. This means that the game is not repetitive, as the user is fighting different types of enemies that are in line with the theme of each section. This was also an asset when my students tested the game, as I found that having different types of enemies motivated them to find them all, and thus made them answer more questions.

2ii.) This objective has been met as every enemy is instantiated based on a random number generator. The number generated is between 0 and 4, meaning the chance of an enemy appearing is 1 in 5. This means that no enemy is ever in the same place every game, meaning it can be replayed without getting tedious.

2iii.) There are three different levels of enemies: normal, boss and the ultimate enemy. This meets the objective and is done with inheriting classes – meaning there is no unnecessarily repeated code and the fighting mechanics of the game do not change (which is helpful for younger users).

1. **There must be a set map that the user can move around** 
   1. **The map must have a certain number of rooms to allow for random generation of enemies**

3i.) The map has 1o rooms in total, which allows different types of enemies to be placed in different areas, without the map being too big or taking too long to play through. The implementation of the map (graph) as a dictionary of dictionaries is also a good solution, as it has a big O of O(1), meaning the lookup time is fast.

1. **The game should have levels of difficulty in accordance with the ideas of Malone**
   1. **This should be in the form of enemies, where each ‘section’ of the map will have a different level of difficulty**

4i.) This objective could have been met more effectively; whilst there are different levels of difficulty with ‘bosses’ and normal enemies, each area does not increase with difficulty until you move to area 4, or the ‘ultimate boss’ battle. One way to achieve this would be to add more questions for each enemy depending on the area or allow for only a certain number of ‘skips.’ However, I feel the current level of difficulty is acceptable considering the randomly generated change in how much damage the user takes in a boss fight. This, in combination with the previously mentioned ‘levels’ of enemies, still meets the requirements of Malone.

1. **The game must comply with the rules**
   1. **Boss characters will have 5 health points – answering each question will get rid of a random amount of health points between 1 and 3.**
   2. **Normal enemies will have 3 health points – each question will get rid of 1 point. It will deal 1 attack damage if you get it wrong.**
   3. **You must defeat the bosses and get ‘items’ they drop. If you have all three items this will unlock room 10, which will let you begin the ultimate boss battle.**
   4. **The Ultimate boss will have as many health points as the number of questions the user enters, therefore asking all the questions. It can deal 3 attack damage if you get a question wrong.**
   5. **The player will have 30 health points.**

5.) The game fully complies with all rules set. All enemies are instantiated with the correct number of health points and attack damage. The ‘drop’ element of the game is also well implemented, as it uses a list to check if all items have been collected and therefore all of the ‘bosses’ have been defeated – which is efficient.

1. **There must be feedback for students after they have finished the game**
   1. **This will be represented as points and the score will be shown throughout the game**

6i.) This objective has been met; however, I feel as if I could improve the project in this area. At present, the score is shown throughout the game in the ‘move’ menu and when the user wins/dies. This feedback could be more personalised, such as telling them the questions they got wrong the most and giving them a new question file to play based on this. This would allow the user to gain even more familiarity with the questions and show them which ones they know well/need to work on.

1. **The game should be easy to use as the end users are younger children who may not understand complicated inputs**
   1. **There should be predefined letters to input (with validation) so that users can make choices of where to go easily**
   2. **There should be a help option if the user is stuck**

7i.) This objective has been met, as all inputs have been validated thoroughly to ensure that younger users can easily input their choices. Moreover, the game is not case sensitive, which is useful as it makes the inputs less specific.

7ii.) This objective has also been met, as there is a specific option in the move menu that gives the user more information on how the game works if they are stuck. However, a useful addition could be a menu when inputting files that also explains to the user how to do so if they are unsure.

## Independent feedback:

To see how well the program worked with my end-users, I asked one of my students to do an interview with me on how they found the game. I asked him about each individual requirement of the objectives as well as the general usability of the program.

He said he found writing questions that he wanted to be asked (1ii) was clear and easy to do. However, he found loading the file (1i) quite difficult and thought that there should have been more help/explanation to get his question set from ‘Quizlet’ into the game. He suggested the idea of a separate part of the game to type specific questions and get answers from the program.

In terms of the enemies, he thought that there was a good range (2i) and enjoyed discovering different areas and the types of enemies associated with them. He also enjoyed the different levels (2iii) of enemies (which I found kept him entertained as he was trying much harder to defeat ‘bosses’) and the story of defeating the final boss with items from the bosses made him feel like he had accomplished something aside from learning his key French words. Initially, he did struggle slightly with moving around the map (3i) however after reading the help section he understood this (6ii). As a younger user, he managed to complete the game with no external help – apart from pointing out the help menu – which was the main aim of objective 6.

He also agreed with my previous assessment of objective 4, as he thought that there could have been more help with questions that he got wrong/right. He found that certain words (like pomegranate) were more difficult for him to remember and said that he would have liked a list of words he could relearn for the test.

Despite his suggestions, he said that overall, he loved the game and found it much more entertaining than the typical type of test that I usually give him and particularly enjoyed the music and art as well. He thought the layout was clear and the game was useful for remembering and was easy to use. My favourite comment from him, though, was that he hoped he could use the game again for more tests!

## Evaluation of independent feedback:

I do agree with most of the points mentioned by my student, as I also found the same issues within my program. His comments regarding personalised feedback could be easily implemented, as I could write a method that stores the questions that have been answered incorrectly in a 2D array that stores the question, answer and the number of times the player has gotten the question wrong. From that point, it could be written into a separate file (in a place that the user specifies) so that it can either be used again in the game or as a record for them to relearn.

I also found his suggestions about a help menu to be useful, as this could help younger users or players who are less knowledgeable about technology. It can be easily implemented as another choice in the main menu, however specific help (such as a question-and-answer database) may require too much processing for only a small number of questions that can be asked.

My final conclusions on the project are that I am extremely happy with the program I have built, as it is an effective solution to an issue that I have faced, and I can see myself using it again with other students to encourage active participation when I am teaching.

# References

Malone, T., 1980. *What makes things fun to learn? A study of intrinsically motivating computer games. ,* Palo Altao, CA: Xerox Palo Alto Research Center.

Pleasant, R., 2013. Book Review: Video Games and Learning. *International Journal of Gaming and Computer-Mediated Simulations,* Volume 5(4), p. 100.

Shri, n.d. *Stack Overflow.* [Online]   
Available at: https://stackoverflow.com/questions/604831/collection-was-modified-enumeration-operation-may-not-execute

Squire, K., 2011. *Video Games in Education.* s.l., Massachusetts Institute of Technology.

## Credit:

‘Blossom’ music used in main menu and start room – Mukul Mehra