MENTAL RHYTHM DOCUMENTATION

TYREASE TEER, ZACHARY LLOYD

UNIVERSITY OF ADVANCING TECHNOLOGY

DESCRIPTION:

Mental Rhythm is a basic simulation of how well someone’s memorization skills are by generating a sequence of tones for them to input. The sequence generated by the program may have between 10 and 20 notes in which consist of the following notes:

* + C (5th Octave): resonating at a frequency of 523.2511Hz
  + D (5th Octave): resonating at a frequency of 587.3295Hz
  + E (5th Octave): resonating at a frequency of 659.2551Hz
  + F (5th Octave): resonating at a frequency of 698.4565Hz
  + G (5th Octave): resonating at a frequency of 783.9909Hz

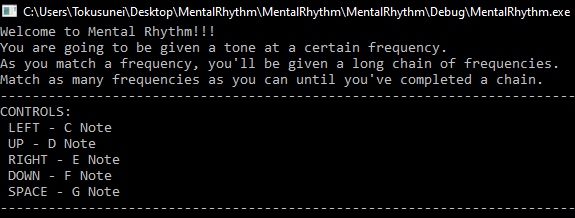
You are also capable of creating your own sequence, saving it as a .mrb file, and executing it through the program. You can also check what score you got on what sequence.

FEATURES:

The program consists of 10 programming features such as:

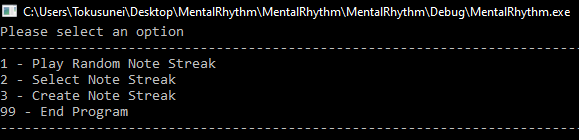
* An opening screen with a description of the application and instructions.
* A menu for the user to choose options.
* 4 classes
* Inheritance
* Dynamic Polymorphism
* Encapsulation
* File Input and Output Processing
* Iterators
* Exception Handling
* A Singleton Class (Design Pattern)
* The use of the Boost Library

OPENING SCREEN

“Welcome to Mental Rhythm!!! You are going to be given a tone at a certain frequency. As you match a frequency, you’ll be given a long chain of frequencies. Match as many frequencies as you can until you’ve completed a chain.” 

MENU

The first option, a note streak will randomly be generated for you. The second option will iterate through the default directory and output any existing files. The third option will prompt you to create a sequence. Inputting “<” will finish the recording of your streak, and your streak will be saved.

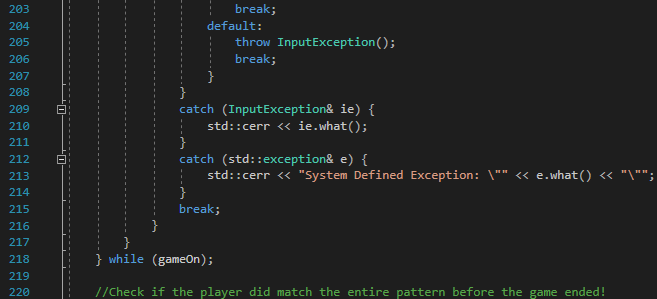
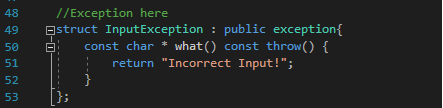


4 CLASSES / INHERITENCE

* ConsolePrint.h
  + class ConsolePrint
    - public ConsolePrint() constructor
    - public string DisplayText(string text) method
    - private string name property
  + class HelpPrint (derives from class ConsolePrint)
    - public HelpPrint() constructor
    - public void GiveTutorial() method
  + class Menu (derives from class ConsolePrint)
    - public Menu() constructor
    - public ~Menu() destructor
    - public int ShowMenu() method
* FileManager.h
  + class FileManager (singleton)
    - public void Read() method
    - public void Write() method
    - public static FileManager \* Get() method
    - private FileManager() contructor
    - private void init() method
    - private static atomic<FileManager\*> pinstance property
    - private static mutex m\_ property
  + class FileMenu (derives from FileManager and ConsolePrint)
    - public int ShowMenu() method
* MyRandom.h
  + class Random
    - public Random() constructor
    - public int GetNumberRange(int a, int b) method
* Sim.h
  + class Sim (singleton)
    - public ~Sim() destructor
    - public void Start() method
    - public bool MatchingStreak(std::string a, std::string b) method
    - public string Generate() method
    - public static Sim\* Get() method
    - public int inputVal property
    - private Sim() contructor
    - private void init() method
    - private static atomic<Sim\*> pinstance property
    - private static mutex m\_ property
    - private std::string noteStreak property

EXCEPTION HANDLING

This is snippets of code found in Sim.cpp:



DESIGN PATTERN  
Singleton -  
We choose to do a singleton implementation because we are used to it from our game classes. With the knowledge from there we realized it creates a single instance of an object. This is perfect for our project because all we need is one simulation running at run time. The simulation .cpp file is where it is located. This allows us to ensure that only one instance of the game can be running while you are playing.